Grado en Ingeniería Electrónica Industrial y Automática

Curso 2019 – 2020

Septiembre 2020

Trabajo Fin de Grado

“MODELADO Y SIMULACIÓN DEL PUERTO DE SANTA CRUZ”

Sofía Paula Techera Biraben

Tutor

Iván Castilla Rodríguez

Agradecimientos.

A Iván, tutor del proyecto, por estar a disposición siempre ante cualquier duda o inconveniente.

A Víctor Morín, trabajador del puerto de Santa Cruz, por toda la información aportada.

A las personas que han formado parte de esta experiencia y especialmente a mi familia, por todo.

RESUMEN.

En el puerto de Santa Cruz de Tenerife, como en la mayoría de puertos, se realiza una gestión y planificación de las operaciones que se deben realizar a lo largo de la jornada laboral. Esto incluye las acciones de carga y descarga de contenedores desde el barco, así como el traslado de los mismos hacia el patio del puerto. Esta gestión y planificación de recursos y actividades es muy importante, ya que se debe coordinar el proceso para evitar, en su mayor parte, conflictos y/o retrasos en la actividad planificada que pueden dar lugar al no cumplimiento de los objetivos del día.

El fin principal de este proyecto es tener la posibilidad de realizar simulaciones sobre la actividad mar-patio del puerto para así, tener la capacidad de optimizar recursos y tiempo, minimizando la probabilidad de cualquier tipo de interrupción y/o parada en las actividades diarias.

Para llevar a cabo una gestión de recursos de manera eficiente y una simulación lo más realista posible, se ha realizado un estudio previo en las actividades mar-patio del Puerto de Santa Cruz de Tenerife gracias a la ayuda de uno de los transportistas del muelle, Víctor Morín, que cuenta con años de experiencia en las actividades que se realizan en estas zonas.

El proyecto se ha realizado en lenguaje Java utilizando la herramienta PSIGHOS, creada por el personal docente de la Universidad de La Laguna, que permite realizar simulación de eventos discretos. PSIGHOS nos permiten crear experimentos, dichos experimentos contienen simulaciones y dichas simulaciones están formadas por recursos, elementos, actividades, grupos de trabajo, ciclos de tiempo...

Para facilitar la utilización del programa, se ha creado una interfaz gráfica de usuario, en la que éste podrá definir los parámetros del modelo y obtener los resultados de la simulación de una manera clara. Para ello, contamos con una ventana en la que podemos seleccionar el contenedor que nos interese y ver en qué momento se está utilizando cada recurso, complementando esto, se nos abre otra ventana con el porcentaje de utilización de cada recurso, que nos ayuda a ver qué recursos no nos hacen falta para llevar a cabo las actividades, así como los recursos que se pueden pasar de tiempo de utilización, ocasionando demoras.

Este proyecto deja abierta la posibilidad de unificar las actividades de las distintas zonas del puerto con el fin de aportar accesibilidad y así servir de apoyo a los supervisores.

ABSTRACT.

In the port of Santa Cruz de Tenerife, as in most ports, there is a process focused on managing and planning the operations that must be done throughout the work day. This process includes both loading and unloading containers from the ship, and transferring such containers to the port's courtyard. This planning and management of resources and activities is quite important, as the process should be coordinated to avoid, mostly, conflicts and delays on the planned activities that could cause unfulfillment of the daily objectives.

The main goal of this project is to have the possibility of running simulations of the sea-courtyard activities of the port in order to obtain the ability to optimize resources and time, minimizing the probability of any type of interruption and/or stoppage of daily actions. In order to manage resources efficiently and to carry out realistic simulations, a preliminary study was made about the sea-courtyard activities of the Santa Cruz de Tenerife Port, thanks to the assistance of one of its workers, Victor Morín, who has years of experience related to the aforementioned activities.

The project was developed using Java and the PSIGHOS library, created by the teaching staff at Universidad de La Laguna, which allows the simulation of discrete events. PSIGHOS enables the creation of experiments, which contain simulations, and said simulations are comprised of resources, workgroups, elements, activities, time cycles, etc.

To simplify the use of the program, a graphical user interface has been created, where the parameters of the model can be defined and the results of the simulation can be observed clearly.

This project opens up the possibility of unifying the activities of different zones of the port, as well as adapting it to mobile devices, with the goal of adding accessibility and supporting supervisors.

CONTENTS

1. Introducción. 1
   1.1. Puerto de contenedores 1
   1.2. El contenedor 2
   1.3. Equipos de trabajo 4
      1.3.1. Grúas pórtico (Grúas barco) 4
      1.3.2. Grúas de patio 4
      1.3.3. Camiones con plataforma 5
      1.3.4. Otros equipos 6
   1.4. Sistemas de la terminal 7
      1.4.1. Carga y descarga de contenedores 7
      1.4.2. Almacenamiento 7
      1.4.3. Entrega y recepción 7
      1.4.4. Transporte interno 8
   1.5. Objetivos del proyecto 8
   1.6. Estructura de la memoria 9
2. Detalles del proyecto. 10
   2.1. Simulación 10
      2.1.1. Fases en el estudio de la simulación 11
      2.1.2. Simulación de eventos discretos 12
   2.2. Estructura y funcionamiento del Puerto de Santa Cruz 12
3. Fases del estudio de la simulación 14
   3.1. Objetivos el sistema 14
   3.3. Modelo informático 16
      3.3.1. Implementación del modelo 17
   3.4. Experimentos 21
   3.5. Resultados 24
      3.5.1. Ejemplo 1 24
      3.5.2. Ejemplo 2 26
4. Conclusiones 29
   4.1. Conclusions 30
5. Bibliografía 31
6. ANEXOS 32

1. INTRODUCCIÓN.

1.1. Puerto de contenedores

Un puerto es un lugar donde los barcos pueden llevar a cabo acciones de embarque, desembarque, carga y descarga de contenedores. En las zonas marítimas, se incluyen construcciones para proteger los barcos, así como dársenas para las operaciones o permanencia del barco. [1]

Las terminales marítimas permiten conectar el trasporte marítimo y terrestre, funcionando como un centro logístico donde se interconectan varios sistemas que forman parte del funcionamiento normal de una terminal. En las terminales de contenedores, se combinan varios tipos de trasporte para trasladar los contenedores desde su lugar de origen, que en este caso será el barco, hasta el patio, donde se colocan estos contenedores [2].

Ilustración 1: Vista aérea puerto de contenedores

https://twitter.com/mexicoxport/status/1067822337989464064

Un puerto se considera terminal de contenedores portuaria (TCP) si cuenta con las instalaciones adecuadas para el manejo de los diferentes tipos de contenedores. Para ello existen diferentes zonas:

- Zona de operación: en esta zona se realizan las cargas y descargas de contenedores a los correspondientes barcos.

Para disminuir el tiempo de espera del buque en puerto, se debe tener en cuenta la disponibilidad de recursos que tenga la terminal para que el tiempo de demora sea el mínimo posible.

- Zona de almacenamiento: lo que entendemos como patio, aquí es donde se van a trasladar los contenedores.

  Este almacenamiento suele ser por poco tiempo, ya que el patio es una zona descubierta. Los contenedores de frío suelen tener una zona asignada para mantener ese sistema de enfriado, pero tampoco es una zona cubierta.
- Zona de cambio de transporte: donde se llevan a cabo las actividades de carga y descarga, pero vía terrestre.
- Zona de servicio: donde se sitúan todas los talleres, oficinas, aduanas y demás servicios que puede prestar el puerto.

1.2. El contenedor

Los contenedores cuentan con enganches en las 8 esquinas, 4 en la parte superior del contenedor y 4 en la inferior, para poder introducir el instrumento de la grúa que nos permite elevarlo (spreader). Además, los contenedores deben cumplir las siguientes características:

- Que permita su uso reiterado
- Que contenga dispositivos que permitan su traslado de manera cómoda
- Debe resultar fácil cargarlo y descargarlo
- Su volumen interior tiene que ser como mínimo de un metro cúbico

Esta definición pertenece a la norma UNE 49-751, la cual define también los tipos de contenedores según su funcionalidad:

- Contenedores estándar
- Contenedores para líquidos o gases
- Contenedores refrigerados
- Contenedores ventilados

Para este trabajo se considera que los contenedores son estándar, siendo estos los más comunes, en concreto los de 40 y 20 pies [3].

Las dimensiones de los contenedores tienen que cumplir con la norma ISO-6346 [4], la mayoría de estos son de acero, pero existen otros materiales como el acero o la madera contrachapada reforzada. Todos estos contenedores deben tener enganches en el exterior para poder ser trasladados por las grúas.

20 PIES STANDARD (DRY CARGO) 20´X 8´X 6´

Tara: 2210 - 2400 kg / Carga Máxima 21700 - 28240 kg / Capacidad Cubica 33.3m3

MEDIDAS	EXTERNA		INTERNA		PUERTA ABIERTA	
	Metros	Pies	Metros	Pies	Metros	Pies
LARGO	6.05	20´	5.90	19´4"		
ANCHO	2.43	8´	2.34	7´8"	2.33	7´8"
ALTO	2.59	8´6"	2.40	7´10"	2.29	7´6"

40 PIES STANDARD (DRY CARGO) 40´X 8´X 6´

Tara: 3630-3740kg / Carga Máxima 2674 - 226850kg / Capacidad Cubica 67.7m3

MEDIDAS	EXTERNA		INTERNA		PUERTA ABIERTA	
	Metros	Pies	Metros	Pies	Metros	Pies
LARGO	12.19	40´	12.03	39´6"		
ANCHO	2.43	8´	2.34	7´8"	2.33	7´8"
ALTO	2.59	8´6"	2.40	7´10"	2.29	7´6"

40 PIES HIGH CUBE STANDARD (DRY CARGO) 40´X 8´X 9´6"

Tara: 3880 - 3900kg / Carga Mínima 26580-26600kg / Capacidad Cubica 76.5m3

MEDIDAS	EXTERNA		INTERNA		PUERTA ABIERTA	
	Metros	Pies	Metros	Pies	Metros	Pies
LARGO	12.19	40´	12.03	39´6"		
ANCHO	2.43	8´	2.34	7´8"	2.33	7´8"
ALTO	2.89	8´11"	2.59	8´6"	2.29	7´6"

Ilustración 2: Detalle de los contenedores más utilizados

http://www.iso-house21.com/

Los contenedores siempre pasan por una inspección para verificar el estado interior y exterior del mismo, comprobar el número de contenedor, comprobar el estado de las puertas, verificar el estado de los pisos, paredes y techos de los contenedores, para evitar cualquier tipo de problema debido al estado del mismo, así como la documentación correspondiente asociada a cada contenedor.

1.3. Equipos de trabajo

1.3.1. Grúas pórtico (Grúas barco)

Las grúas pórtico deben su nombre a la estructura en forma de pórtico de las mismas, ya que tienen cuatro columnas y dos vigas, suelen estar hechas de acorde alta resistencia.

Estas grúas realizan la tarea de carga y descarga de los contenedores desde los barcos. Son, por tanto, el equipo de manipulación principal en estas terminales y suelen trabajar con una carga de trabajo desde las 40 a las 120 toneladas.

Ilustración 3: Grúas pórtico en muelle

https://www.liebherr.com/es/esp/productos/gruas-maritimas/equipamiento-de-puerto/gruas-portico-para-la-carga-de-contenedores/gruas-de-contenedores-buque-a-costa.html

El proyecto se centrará en la acción de descarga del contenedor, para que, una vez estando en tierra, se coloque sobre los camiones portacontenedores y se traslade hasta la zona de patio. Esta descarga la ejecuta un operario, siempre siguiendo el plan de actuación establecido.

1.3.2. Grúas de patio

Las grúas de patio o grúas pórtico de almacenamiento, son un equipo presente en las zonas de almacenamiento de los muelles, permiten la carga y descarga de los contenedores, tanto en la zona correspondiente como en los camiones. Las más comunes son las de carril montado (RMG) y las grúas pórticos sobre neumáticos (RTG). [5]

Ilustración 4: Imagen aérea de grúas de patio

http://puertosaduanasssassas.blogspot.com/2015/10/tipos-de-puertos.html

En este proyecto, la acción a destacar es la de descarga desde el camión, que trasladó el contenedor hacia el patio, y la de posicionamiento en el patio.

1.3.3. Camiones con plataforma

Los camiones con plataforma son unos de los elementos más utilizados en los muelles y se utilizan únicamente para el traslado de contenedores en medio terrestre. En este caso, el camión recogerá el contenedor desde el barco, gracias a la grúa pórtico, trasladándolo hasta la zona de patio asignada para el contenedor. Cuando haya acabado y hay contenedores a la espera de ser descargados, este vuelve a la zona de descarga de contenedores del barco, en caso de que ya haya terminado los recorridos diarios, volverá a la zona destinada para los mismos.

Ilustración 5: Camión con plataforma para el trasporte de contenedores

https://www.gruasyaparejos.com/grua-portico-para-contenedores/gruas-de-patio/

1.3.4. Otros equipos

En la actividad portuaria de carga y descarga de contenedores, también se utilizan otros equipos:

- Equipos de manipulación frontal: entre los más comunes están las grúas apiladoras y los cargadores frontales.
- Vehículos guiados automáticamente: tienen la misma función que los camiones con plataforma, pero sin la necesidad de conductor, en la actualidad su uso ha ido en aumento.
- Grúas polivalentes: tienen la capacidad de manejar distintos tipos de carga, pero su rendimiento es muy bajo comparado con otros equipos. En la actualidad ya no quedan grúas polivalentes en los grandes puertos.
- Remolque portacontenedor autocarga: estos remolques poseen un sistema que carga y descarga contenedores de forma autónoma, pudiendo colocar de una a dos alturas

Ilustración 6: Remolque portacontenedor de autocarga

https://www.pinterest.com.au/pin/505036545687910145/

Ilustración 7: Grúa apiladora de contenedores

http://www.sic-lazaro.es/manutencion.htm

1.4. Sistemas de la terminal

Podemos considerar el sistema general del puerto de contenedores como varios subsistemas, los cuales deberán trabajar conjuntamente para que la actividad portuaria se realice de manera segura y eficiente. [6]

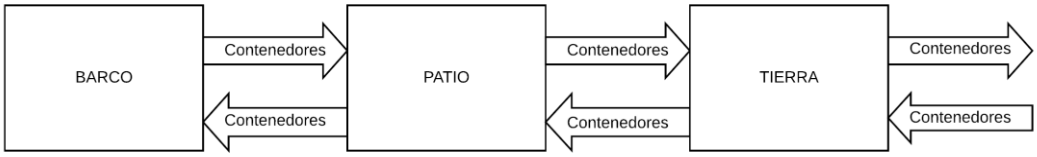


Ilustración 8: Flujo de contenedores en un puerto

1.4.1. Carga y descarga de contenedores

Este subsistema abarca la carga y descarga de los contenedores entre las zonas de mar y tierra. Vemos involucrados a una serie de profesionales dentro de esta tarea:

- Coordinadores de la terminal: preparan las labores diarias y coordinan el trabajo que se está llevando a cabo
- Estibadores: encargados de realizar la actividad de carga/descarga
- La autoridad portuaria: encargados del atraque y desatraque del barco en el muelle.
- Planificadores: encargados de organizar las secuencias de carga y descarga de la terminal teniendo en cuenta diversos factores como peso y destino del contenedor.

1.4.2. Almacenamiento

Zona en la terminal dónde el contenedor es almacenado temporalmente a la espera de ser entregados. Esta zona recibe el nombre de patio o campa.

1.4.3. Entrega y recepción

Este subsistema es la conexión entre la zona de almacenamiento y los sistemas de transporte terrestre. Este trasporte terrestre puede ser tanto camiones como en algunos casos, ferrocarriles. Abarca el proceso de recepción y control de la carga, así como el de carga y descarga en los distintos vehículos.

1.4.4. Transporte interno

Como su nombre indica, abarca el transporte de contenedores y mercancía dentro del puerto, concretamente desde la zona del muelle hasta la zona de almacenamiento y viceversa, aunque a veces se puede incluir el traslado a la zona de entrega y recepción.

1.5. Objetivos del proyecto

Ante la dificultad de coordinar las diversas tareas que surgen en un muelle de contenedores y teniendo como objetivo la optimización de tiempo y de recursos, surge la idea de implementar un programa en el que se pueda obtener información sobre estas actividades, para poder gestionar de manera mucho más eficiente las tareas a realizar en la jornada laboral.

Una elección correcta de la utilización de recursos minimizará lo máximo posible los retrasos, aportando al cliente un buen servicio y ayudando a mantener así, un buen nivel de calidad. Para esto, debemos de tener en cuenta variables como: los costes, la capacidad de carga de cada vehículo, la cantidad de recursos disponibles para realizar las tareas, la frecuencia y el tiempo de las acciones, etc.

El principal objetivo de este proyecto, es, por lo tanto, la utilización de un programa para realizar experimentos y así obtener un modelo que simule el funcionamiento de la carga y descarga de contenedores en la zona mar-patio del puerto de Santa Cruz.

La simulación presenta muchas ventajas como:

- Permite experimentar de forma artificial con el sistema.
- Coste menos elevado que ejecutando las acciones de forma física.
- Se puede utilizar muchas veces.
- No afecta al sistema real.
- Simular operaciones que ocurren en horas o días en segundos.

Para realizar este proyecto adecuadamente, debemos informarnos para así conocer el funcionamiento del puerto y los recursos que emplean para realizar las actividades anteriormente nombradas. Para ello se ha obtenido datos a través de internet y mediante un contacto, Víctor Morín, trabajador del puerto de Santa Cruz.

Se ha creado una interfaz gráfica de usuario para que la utilización del programa sea lo más cómoda posible, en la que se podrá definir los parámetros del modelo y obtener los resultados de la simulación de una manera muy clara.

1.6. Estructura de la memoria

Se describe a continuación la estructura de la memoria del proyecto:

CAPÍTULO II: ***Detalles del proyecto****.* En este capítulo se explica en qué consiste una simulación, se profundiza en la estructura del puerto de Santa Cruz, se desarrollan los objetivos del proyecto y se explica el experimento en cuestión.

CAPÍTULO III: ***Resultados****.* Información sobre resultados y ejemplos.

CAPÍTULO IV: ***Conclusiones***.

CAPÍTULO V: ***Bibliografía***.

CAPÍTULO VI: ***Anexos***.

2. DETALLES DEL PROYECTO.

2.1. Simulación

Para poder estudiar un sistema, la mayoría de las veces se hace uso de un modelo matemático, el cual podrá ser implementado en el caso de que sea lo suficientemente sencillo para que, a la hora de resolverlo, los resultados obtenidos sea lo más coherentes y exactos posibles. En el caso de que el modelo sea muy complejo o necesitemos de varias representaciones de éste, se recurre a la simulación de este sistema. [7]

La complejidad en estos sistemas viene dada básicamente por dos motivos o causas:

- En sistemas continuos: existen variables de estado que representan a otras variables, pudiendo aparecer así ecuaciones diferenciales no lineales que complican enormemente la resolución analítica.
- En los sistemas discretos, pueden aparecer aleatoriamente fenómenos que únicamente pueden ser representados en términos de probabilidad. En estos modelos pueden llegar a coincidir ecuaciones diferenciales complejas con variables aleatorias, lo que complica más aún la resolución.

Se puede destacar la utilización de simulaciones en el estudio de:

- Comunicaciones: correos, teléfonos, redes informáticas...
- Diseño de instalaciones.
- Eficiencia en producción.
- Diseño de actividades: trabajadores, puestos de trabajo...
- Análisis de proyectos.
- Gestión de inventarios.
- Análisis de inversiones.

2.1.1. Fases en el estudio de la simulación

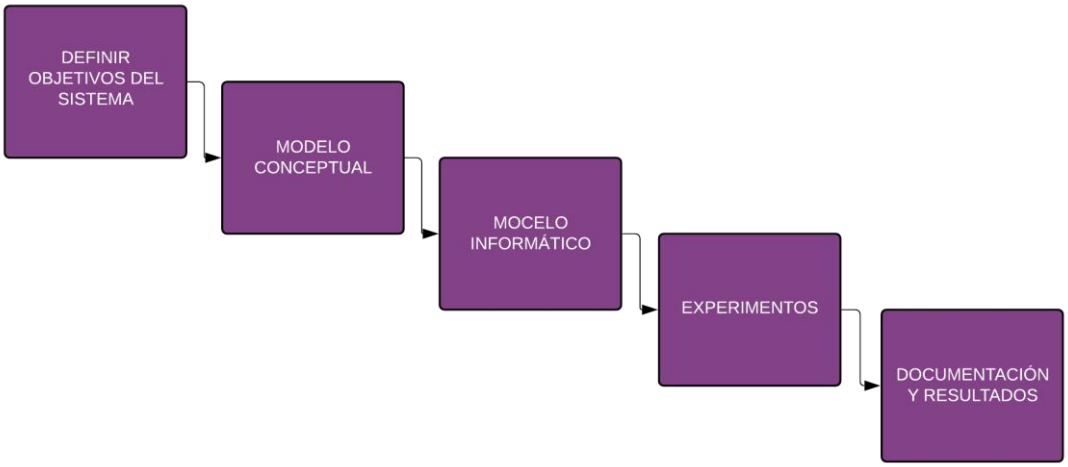


Ilustración 9: Etapas en el estudio de simulación

Primero se comienza especificando los objetivos del sistema y definiendo los elementos que forman parte de este sistema que se estudiará. A continuación, se elabora un modelo conceptual, diseñado a partir de los objetivos propuestos, es recomendable que el modelo sea sencillo pero lo más realista posible. En esta etapa habría que tener en cuenta las diferentes variables y si fuera posible obtener algún dato que nos permita validar el modelo, sirviendo de gran ayuda la opinión de personas que tengan conocimiento amplio del sistema que se representa. Una vez este modelo ya está validado, se selecciona el lenguaje de programación que se va a utilizar a la hora de desarrollar el proyecto, basado en las características tanto del modelo como del leguaje en cuestión. Definido el lenguaje de programación, se comenzará con la programación del mismo y con los experimentos que exija el modelo. Para cada uno de estos experimentos, se deberá dar valor a las condiciones iniciales y tras la ejecución de estos, determinar si los resultados son lógicos para confirmar el modelo. Para finalizar queda elaborar la documentación.

Ya se ha comentado anteriormente que las simulaciones son una técnica cada vez más eficiente en el estudio de sistemas complejos. Gran parte de los sistemas no se pueden ajustar o medir mediante un modelo matemático. Por esto, la simulación es el único método posible de estudio en estos casos. La simulación estudia el comportamiento del con unas condiciones operativas definidas previamente. Se pueden definir distintos diseños para elegir cuál es el que cumple de manera más eficiente con los objetivos. Además, la simulación permite tener un mejor control sobre las condiciones del experimento.

2.1.2. Simulación de eventos discretos

La simulación de eventos discretos, que nos permite implementar PSIGHOS, es, por lo tanto, una herramienta de análisis para aquellos eventos que cambian el estado en instantes espaciados en el tiempo, pudiendo modelar muchos de las actividades de administración de procesos productivos

La mayoría de los sistemas cumplen con las características de los sistemas discretos, ya que los cambios de estado (recepción de contenedores, inicio y finalización de la transporte, entradas y salidas del patio…) se producen en instantes de tiempo determinados y separados entre sí.

2.2. Estructura y funcionamiento del Puerto de Santa Cruz

En este proyecto nos hemos centrado en el funcionamiento y la distribución del Puerto de Santa Cruz de Tenerife. La información ha sido obtenida a través de la página del Puerto y del contacto, anteriormente nombrado, que trabaja en la zona mar-patio del propio puerto. [8]

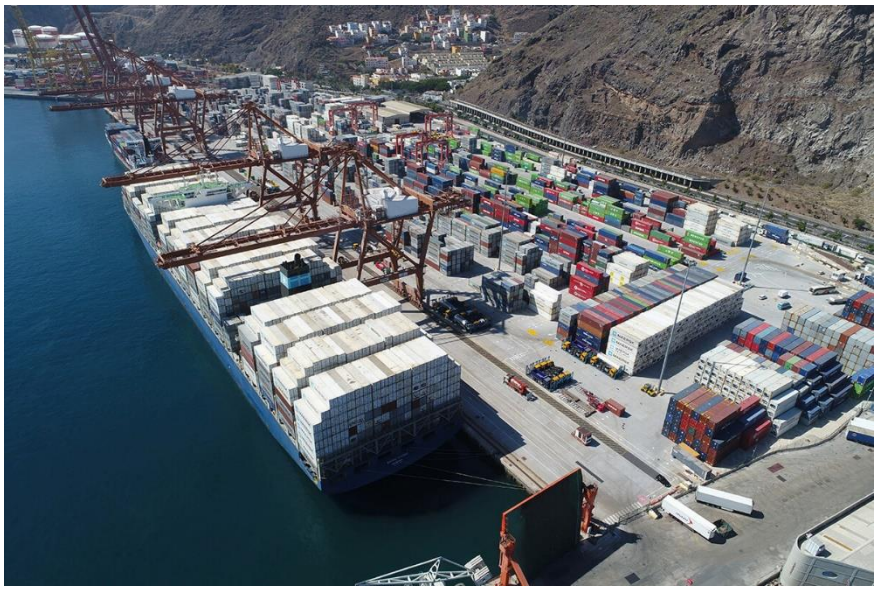


Ilustración 10: Puerto de contenedores de Santa Cruz de Tenerife

El puerto de Santa Cruz, es el principal puerto de la isla de Tenerife y uno de los tres principales puertos del mudo para el tráfico de cruceros, siendo también un punto de escala entre los principales puertos de África, Europa y América.

En la *ilustración 11* podemos observar el flujo del puerto de Santa Cruz, que sigue el modelo general de flujos en puertos de contenedores.

En la zona de mar-tierra de este puerto, se sigue el mismo flujo de trabajo, los barcos llegan al muelle con la carga, las grúas pórtico del muelle se encargan de bajar estos contenedores del barco para poder colocarlos en los camiones con plataforma. Los camiones se encargan de trasladar el contenedor desde el barco hasta la zona del patio que le corresponda y de regresar en el caso de todavía se requiera, si no quedara ningún contenedor para su descarga, el camión volvería a la zona de espera asignada para estos.

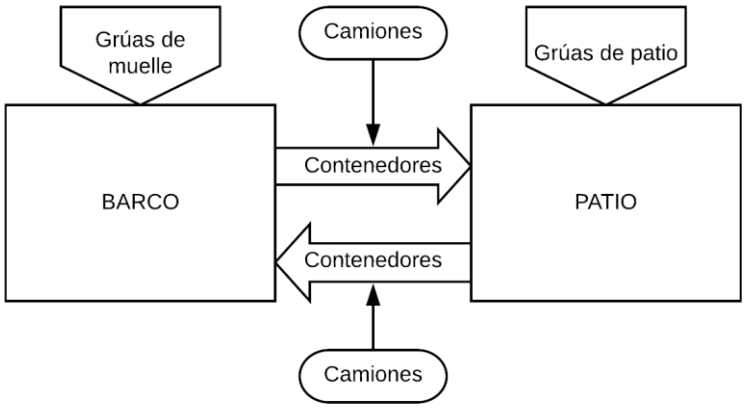


Ilustración 11: Esquema de la zona mar-patio

En lo que se refiere a los contenedores en sí mismos, se distinguen dos tipos de estos, vacíos y llenos, que serán trasladados al patio a su correspondiente bloque, por esta razón, se decide hacer la simulación contando con dos bloques, uno para los contenedores llenos y otro para los contenedores vacíos. No existe ningún tipo de prioridad a la hora de descargar los contenedores desde los barcos, en el caso de que sea un contenedor peligroso o que se necesite con urgencia, es la zona de entrega y recepción que se encarga de ordenarlos para que sean más accesibles.

En el Puerto de Santa cruz, se calcula que descargan unos 180 en una jornada laboral de 6 horas, incluyendo por supuesto, el traslado a la zona de patio correspondiente.

La gestión de las actividades de carga y descarga de contenedores del puerto de Santa Cruz, necesita de una coordinación y planificación previa, adecuada a los recursos que se dispongan. Estos recursos, tanto materiales como humanos, pueden cambiar a lo largo de los días, además, estos se encuentran en constante movimiento

si existe una petición de descarga de contenedor. Esta planificación se realiza previamente a través de un sistema informático, que separa los diferentes tipos de especialidades o puestos, asignándoles unos recursos a cada uno, pero no en conjunto.

3. FASES DEL ESTUDIO DE LA SIMULACIÓN

Basándonos en la estructura y el funcionamiento del puerto de Santa Cruz de Tenerife del apartado *2.2.* procederemos a seguir los pasos de la fase de estudio de la simulación para poder llegar a los resultados del proyecto de una forma más organizada

3.1. Objetivos el sistema

Este apartado tiene relación directa con el apartado 1.5. Objetivos del proyecto de esta memoria, como se nombra en ese mismo apartado surge la necesidad de tener o implementar un programa en el que se pueda obtener información sobre las actividades que se realizan en el puerto.

Como ya se ha nombrado, la idea es obtener una serie de experimentos que simulen el funcionamiento de la carga y descarga de contenedores en la zona mar-patio del puerto de Santa Cruz. Esto incluye añadir la zona de la grúa de barco y los dos patios de contenedores, nombrados anteriormente.

En este caso, el objetivo del sistema es obtener una ventana en la que podamos elegir el contenedor del que queremos obtener información tanto de uso de camiones como de grúas. Además, nos interesa tener una tabla en la que se vea reflejado el orden de descarga de los contenedores y en qué momento se está ejecutando cada acción. En cuanto a la utilización de estos recursos, se nos abrirá una ventana en la que podemos ver el porcentaje de utilización de cada uno de ellos y decidir si ese recurso sobra o si necesitamos más cantidad de los mismos.

En este sistema se van a definir los siguientes elementos:

- Camión
- Grúa barco
- Grúa patio 1
- Grúa patio 2
- Contenedores

3.2. Modelo Conceptual

Como se nombra en el apartado 2.2, y gracias a la información aportada por Víctor Morín, sabemos que este puerto consta de dos zonas dedicadas a los contenedores, una para los contenedores llenos, y otra para los contenedores vacíos, por esta razón, se decide contar con un mapa como el siguiente:

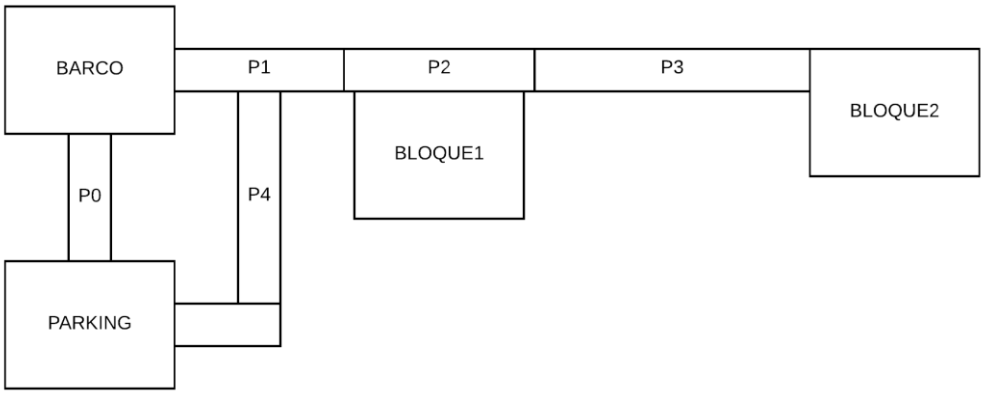


Ilustración 12: mapa de la simulación

Los camiones, saldrán del parking y recorrerán P0 hasta llegar al barco, donde recogerán el contenedor. A continuación, se decidirá si los camiones van al bloque1 o al bloque2 (zonas de patio). La probabilidad de ir a un destino u otro, es de 40% de posibilidades de ir al BLOQUE1, que serán los contenedores vacíos y un 60% de ir al BLOQUE2, que serán los contenedores que sí estén cargados.

En el caso de que haya una petición de camión desde el BARCO y que el camión esté liberado, podrá volver a la zona de barco sin necesidad de pasar por el parking. En el caso de que ya no existan más peticiones, este podrá volver por P4 a la zona asignada para ellos (PARKING). Aclarar que tanto P1, P2 como P3, son bidireccionales, permitiendo que los camiones puedan ir y volver de los bloques sin problema.

Una vez tenemos esto, la idea es a través del modelo informático, recrear este modelo conceptual de la manera más real posible, para que, a la hora de obtener las gráficas de tiempo de utilización de cada recurso, estas sean lo más próximas a la realidad y así obtener resultados lógicos.

3.3. Modelo informático

El fin principal de este proyecto, como se ha explicado anteriormente, es tener la posibilidad de realizar simulaciones sobre la actividad mar-patio del puerto de Santa Cruz de Tenerife, para así, tener la capacidad de optimizar recursos y tiempo, minimizando la probabilidad de cualquier tipo de interrupción y/o parada en las actividades diarias.

Basándonos en la distribución del puerto y con visión de posibles futuras mejoras, se pretende tener la posibilidad de trabajar con diferentes rutas, esto es posible gracias a la implementación de las clases *Router, MoveResourceFlow y PropabilitySelectionFlow* que provee PSIGHOS.

Se desea crear o permitir, la asignación de probabilidad a la hora de elegir un destino u otro para la descarga en patio de los contenedores. Podemos apreciar el modelo en la *Ilustración 12*. Esto hace que podamos ajustarnos mejor al modelo real del Puerto, que también consta de dos Bloques en la zona de patio.

Para facilitar la utilización del programa, se ha creado una interfaz gráfica de usuario, en la que éste podrá definir los parámetros del modelo y obtener los resultados de la simulación de una manera clara, pudiendo elegir de cuál contenedor obtendremos la información del tiempo y obteniendo también una gráfica con el porcentaje de utilización en tiempo de cada uno de los recursos, como hemos nombrado anteriormente, que no permitirá decidir si ese recurso podría no utilizarse o, si al contrario puede excederse de tiempo y ocasionar demoras en las actividades que se pretenden realizar.

3.3.1. Implementación del modelo

Actualmente, el proyecto se organiza en tres paquetes, "gui", "harbor" y "infoReceiver".

Ilustración 13: Muestra de los paquetes del proyecto

El paquete "gui" (Graphical User Interface o Interfaz Gráfica de Usuario), contendrá todas las clases utilizadas para mostrar información y pedir, en concreto para mostrar y pedir información mediante un entorno gráfico.

- InitWindow: ventana en la que se piden los datos para el experimento
- ConfirmationWindow: ventana de confirmación de los parámetros
- ResultsTable: Ventana que contiene las gráficas y la tabla del experimento
- Main: La clase *Main* del paquete gui, contiene la función main(), en este momento, la función main establece los parámetros de tiempo inicial y tiempo final, así como la iniciación de la ventana InitWindow y la instancia HarborExperiment, clase definida en el paquete "harbor" y que básicamente representa un experimento.

El paquete "harbor", contiene las clases referidas a la simulación, el experimento y el router. A grandes rasgos, *HarborExperiment* hereda de *Experiment* (clase de PSIGHOS) y contiene el método *getSimulation*() donde se crea una instancia de la simulación y se agregan los InfoReceivers de la simulación.

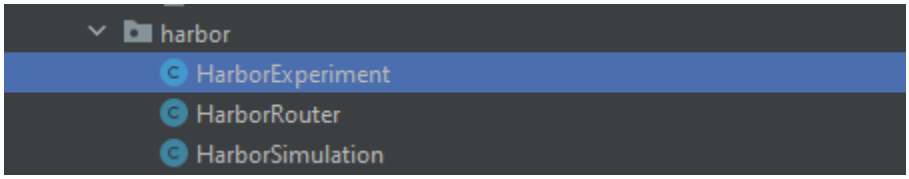


Ilustración 14: Muestra del contenido del paquete "harbor"

La clase *HarborRouter* hereda de *Router* y es básicamente la clase que define el "mapa", de la simulación, que podemos ver en la *Ilustración 16*, estableciendo las reglas sobre a dónde podemos movernos según donde se encuentre el recurso actual.

```
parking = new Node( description: "Parking", TimeFunctionFactory.getInstance( className: "ConstantVariate", DELAY_HOME), capacity: 50);
ship = new Node( description: "Barco", TimeFunctionFactory.getInstance( className: "ConstantVariate", DELAY_HOME), nCranes);
block1 = new Node( description: "Patio 1", TimeFunctionFactory.getInstance( className: "ConstantVariate", DELAY_HOME), nYardCranes[0]);
block2 = new Node( description: "Patio 2", TimeFunctionFactory.getInstance( className: "ConstantVariate", DELAY_HOME), nYardCranes[1]);
paths = new Path[NPATHS];
```

Ilustración 15: Muestra del código donde se definen los Nodos del mapa

```
  // Definición del mapa (definimos cómo están conectadas las cosas)
  parking.linkTo(paths[0]);

  paths[0].linkTo(ship);

  ship.linkTo(paths[1]);

  paths[1].linkTo(ship);
  paths[1].linkTo(paths[2]);
  paths[1].linkTo(paths[4]);

  paths[2].linkTo(block1);
  paths[2].linkTo(paths[1]);
  paths[2].linkTo(paths[3]);

  paths[3].linkTo(block2);
  paths[3].linkTo(paths[2]);

  paths[4].linkTo(parking);
}
```

Ilustración 16: Muestra del código donde se define las conexiones del mapa

Por último, la clase *HarborSimulation*, que hereda de *Simulation* y que contiene la simulación del modelo utilizando elementos, recursos, grupos de trabajo, *activityflow*. En cuanto a los elementos del modelo, son la parte más importante, ya que son los que interaccionan con el sistema.

```
public class HarborSimulation extends Simulation {

  // Resources descriptions
  public static final String CONTAINER = "Container.";
  public static final String TRUCK = "Delivery Truck.";
  public static final String SHIP_CRANE = "Ship crane.";
  public static final String YARD_CRANE = "Yard crane ";

  // Activities
  public static final String RELEASE_CONTAINER_AT_BLOCK = "Release container at Block ";

  // Movement flows
  public static final String TRUCK_TO_SHIP = "Move truck to ship.";
  public static final String TRUCK_TO_B = "Move truck to Block ";

  // Activities
  public static final String PICK_CONTAINER_FROM_SHIP = "Pick container from the ship.";
  public static final String REQ_CONTAINER = "Container ready to go.";
  public static final String RELEASE_TRUCK = "Release truck.";

  public static final Double B1_PERCENTAGE = 0.4; // Percentage
  public static final Double B2_PERCENTAGE = 0.6; // Percentage

  public static final Integer DELAY_SHIP_CRANE = 2; // Minutes
  public static final Integer DELAY_YARD_CRANE = 2; // Minutes
```

Ilustración 17: Muestra del código donde se describen los recursos, las actividades y el flujo.

Para definir los tipos de elementos que existen en la simulación, en nuestro caso los elementos serán los contenedores, se utiliza la clase *"Element Type"*

Los recursos son los mecanismos con los que los elementos puedan realizar las actividades y se representan con un objeto de la clase *"Resource"*. Todos estos recursos se deberán agrupar según el tipo del mismo, para ello, utilizaremos la clase "*ResourceType".* En este caso, los tipos de recurso son las grúas de barco, las grúas de patio1, las grúas de patio2 y los camiones, como se ve en la siguiente captura:

```
// Define elements
final ElementType etContainers = new ElementType( model: this, CONTAINER);

// Define resources
final ResourceType rtTruck      = new ResourceType( model: this, TRUCK);
final ResourceType rtShipCrane  = new ResourceType( model: this, SHIP_CRANE);
final ResourceType rtYardCrane1 = new ResourceType( model: this,  description: YARD_CRANE + "Block 1");
final ResourceType rtYardCrane2 = new ResourceType( model: this,  description: YARD_CRANE + "Block 2");
```

Ilustración 18: Muestra de código en la que se definen los recursos y los elementos.

Se define también una clase para crear grupos de recursos ya que cada actividad tiene un grupo de trabajo y cada uno con sus recursos correspondientes, para esto utilizamos la *herramienta "Workgroup".*

```
// Asignación de los WorkGroup
final WorkGroup wgTruck = new WorkGroup( model: this, rtTruck,  needed: 1);
final WorkGroup wgShipCrane = new WorkGroup( model: this, new ResourceType[] { rtShipCrane }, new int[] { 1 });
final WorkGroup wgYardCranes1 = new WorkGroup( model: this, new ResourceType[] { rtYardCrane1 }, new int[] { 1 });
final WorkGroup wgYardCranes2 = new WorkGroup( model: this, new ResourceType[] { rtYardCrane2 }, new int[] { 1 });

// Definición de Actividades y movimientos.
final RequestResourcesFlow reqFlow = new RequestResourcesFlow( model: this, REQ_CONTAINER);
reqFlow.newWorkGroupAdder(wgTruck).add(); // Hace falta camiones disponibles para requerir camiones.

ActivityFlow actTruckShip = new ActivityFlow( model: this, PICK_CONTAINER_FROM_SHIP);
ActivityFlow actTruckB1 = new ActivityFlow( model: this,  description: RELEASE_CONTAINER_AT_BLOCK + "1");
ActivityFlow actTruckB2 = new ActivityFlow( model: this,  description: RELEASE_CONTAINER_AT_BLOCK + "2");

actTruckShip.newWorkGroupAdder(wgShipCrane).withDelay(DELAY_SHIP_CRANE).add();
actTruckB1.newWorkGroupAdder(wgYardCranes1).withDelay(DELAY_YARD_CRANE).add();
actTruckB2.newWorkGroupAdder(wgYardCranes2).withDelay(DELAY_YARD_CRANE).add();
```

Ilustración 19: Muestra de código en el que se asignan los grupos de trabajo y el flujo de los mismos.

Las actividades requieren un tiempo para realizar la tarea asignada, para esto creamos las actividades utilizando la clase "*ActivityFlow*", a las que le añadiremos un *delay*, que será lo que indique el tiempo que tarda cada una.

Con esto solo nos queda definir el movimiento de estos recursos con "*MoveResourseFlow*" y definir el flow probabilístico que decidirá si el contenedor se traslada al patio o bloque 1 o si se traslada al patio o bloque 2. Esta probabilidad será de un 40 y 60% respectivamente

```
final MoveResourcesFlow moveToShip = new MoveResourcesFlow( model: this, TRUCK_TO_SHIP, router.getDestinationShip(), router, wgTruck);
final MoveResourcesFlow moveToB1 = new MoveResourcesFlow( model: this,  description: TRUCK_TO_B + "1", router.getDestinationB1(), router, wgTruck);
final MoveResourcesFlow moveToB2 = new MoveResourcesFlow( model: this,  description: TRUCK_TO_B + "2", router.getDestinationB2(), router, wgTruck);

final ReleaseResourcesFlow relFlow = new ReleaseResourcesFlow( model: this, RELEASE_TRUCK, wgTruck);

/* Se define un flow probabilístico.
 */
final ProbabilitySelectionFlow selectBlockFlow = new ProbabilitySelectionFlow( model: this);
// Por ejemplo, suponemos que un 40% irán al bloque 1 y un 60% al bloque 2
selectBlockFlow.link(moveToB1, B1_PERCENTAGE);
selectBlockFlow.link(moveToB2, B2_PERCENTAGE);
```

Ilustración 20: Muestra de código en la que se define el movimiento de los recursos y se asigna un flujo probabilístico

El paquete *"infoReceiver",* contendrá los receptores de información de la simulación. Estos serán los encargados de calcular todo lo necesario en base a la información que reciben y cuando finalice la simulación, llamar a la clase que muestre gráficos o cualquier otro tipo de salida al usuario.

3.4. Experimentos

Para realizar los experimentos, debemos de tener en cuenta los datos que tenemos sobre el funcionamiento del puerto. En este caso, la información que tenemos ha sido facilitada por uno de los trabajadores del puerto de contenedores de Santa Cruz, Víctor Morín, que se dedica al transporte de los contenedores nos ha dado los siguientes datos:

- Camiones por jornada laboral: Entre 12 y 17 camiones
- Grúas por barco: Entre 2 y 3
- Grúas por cada patio: Entre 2 y 4
- Número de contenedores por jornada laboral: Entre 150 y 180
- Jornada laboral: Entre 6 y 7 horas

A modo de guía y para que sirva de apoyo en el resultado de los experimentos, Víctor Morín comenta: *"Ahora mismo los conductores nos solemos ver cortos de tiempo ya que están entrado más contenedores que en años anteriores, pero seguimos con el mismo número de camiones."*

Al ejecutar el proyecto, lo primero que se ve es la ventana de pedida de parámetros

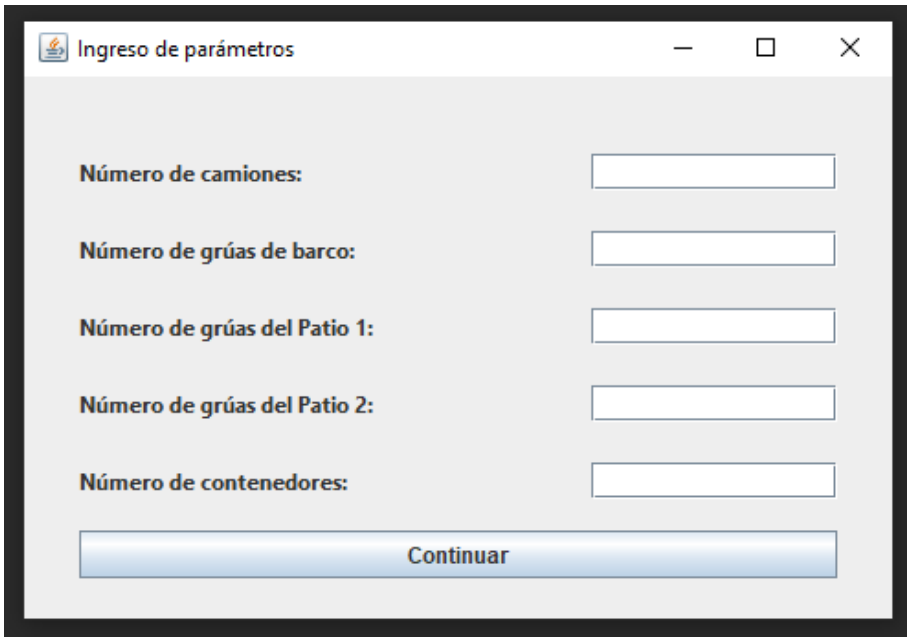


Ilustración 21: Ventana para el ingreso de parámetros

Una vez se hayan ingresado los datos en la interfaz, procederemos a comprobar que estos están bien. En el caso de que los parámetros no sean los correctos, podremos utilizar el botón de *Corregir* para volver a ingresarlos, si por el contrario los datos para la simulación son correctos, le daremos al botón *Continuar*.

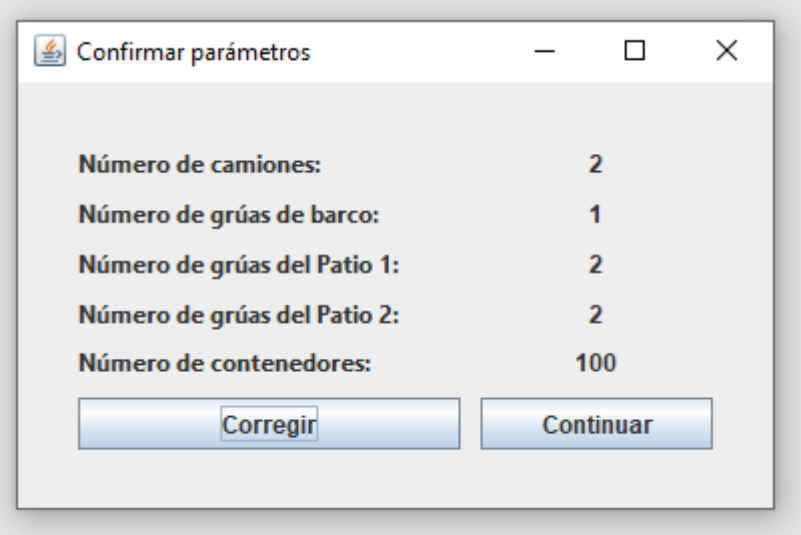
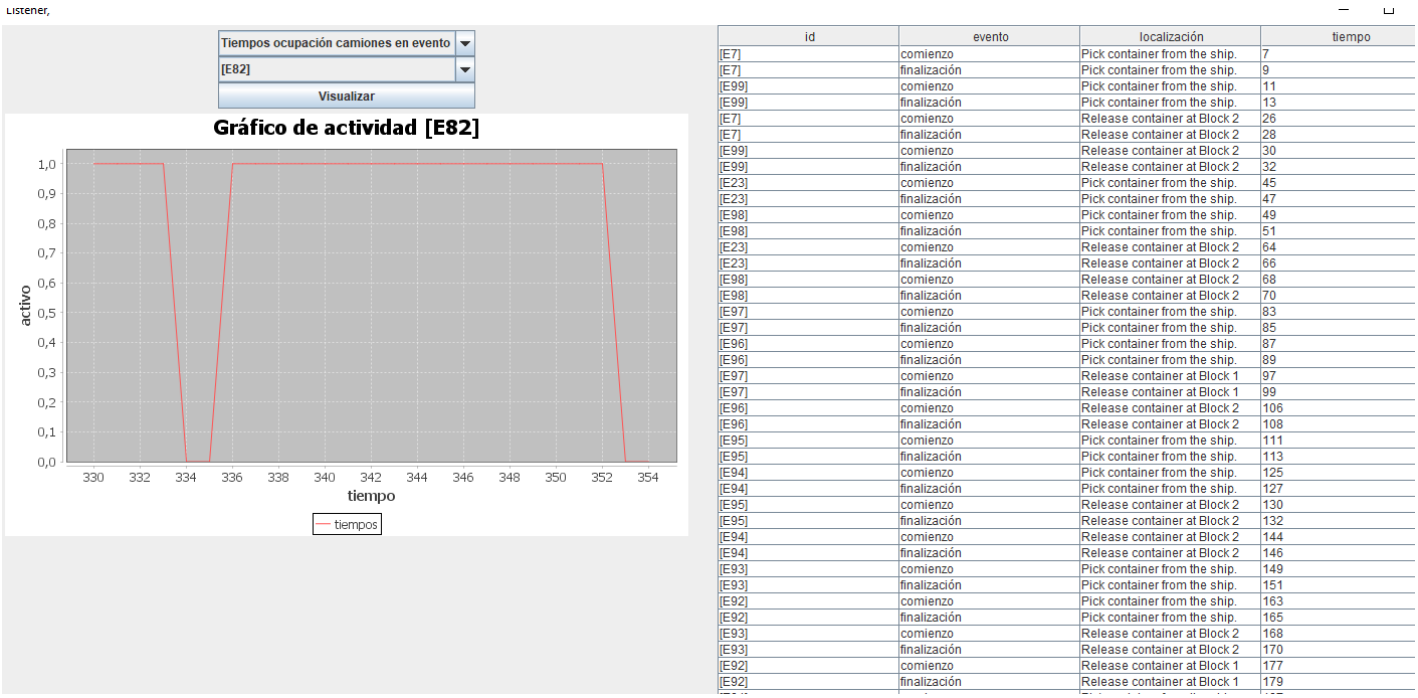


Ilustración 22: Ventana confirmación de parámetros

Una vez le demos a *Continuar*, se nos abrirán dos ventanas, como ya se ha nombrado en anteriores apartados. Una de ellas nos permite ver en qué momento se está utilizando tanto las grúas como los camiones filtrándolo por contenedor, como apoyo tendremos una tabla en la que tendremos una lista de los tiempos en los que está ocurriendo cada evento, como podría ser la recogida de un contenedor o el camión llegando a alguno de los patios.



id	evento	localización	tiempo
[E7]	comienzo	Pick container from the ship.	7
[E7]	finalización	Pick container from the ship.	9
[E99]	comienzo	Pick container from the ship.	11
[E99]	finalización	Pick container from the ship.	13
[E7]	comienzo	Release container at Block 2	26
[E7]	finalización	Release container at Block 2	28
[E99]	comienzo	Release container at Block 2	30
[E99]	finalización	Release container at Block 2	32
[E23]	comienzo	Pick container from the ship.	45
[E23]	finalización	Pick container from the ship.	47
[E98]	comienzo	Pick container from the ship.	49
[E98]	finalización	Pick container from the ship.	51
[E23]	comienzo	Release container at Block 2	64
[E23]	finalización	Release container at Block 2	66
[E98]	comienzo	Release container at Block 2	68
[E98]	finalización	Release container at Block 2	70
[E97]	comienzo	Pick container from the ship.	83
[E97]	finalización	Pick container from the ship.	85
[E96]	comienzo	Pick container from the ship.	87
[E96]	finalización	Pick container from the ship.	89
[E97]	comienzo	Release container at Block 1	97
[E97]	finalización	Release container at Block 1	99
[E96]	comienzo	Release container at Block 2	106
[E96]	finalización	Release container at Block 2	108
[E95]	comienzo	Pick container from the ship.	111
[E95]	finalización	Pick container from the ship.	113
[E94]	comienzo	Pick container from the ship.	125
[E94]	finalización	Pick container from the ship.	127
[E95]	comienzo	Release container at Block 2	130
[E95]	finalización	Release container at Block 2	132
[E94]	comienzo	Release container at Block 2	144
[E94]	finalización	Release container at Block 2	146
[E93]	comienzo	Pick container from the ship.	149
[E93]	finalización	Pick container from the ship.	151
[E92]	comienzo	Pick container from the ship.	163
[E92]	finalización	Pick container from the ship.	165
[E93]	comienzo	Release container at Block 2	168
[E93]	finalización	Release container at Block 2	170
[E92]	comienzo	Release container at Block 1	177
[E92]	finalización	Release container at Block 1	179

Ilustración 23: Ventana con los gráficos de actividad por recursos y tabla de apoyo.

Estas gráficas funcionan más que nada como un complemento, pero la más interesante es la del porcentaje de tiempo de utilización de cada recurso, que nos hará ver cuales recursos están utilizándose más de lo que deberían y cuales se utilizan poco o nada. Aquellos recursos que superen el 95% del tiempo utilizado se considera que podrían entrar en conflicto, impidiendo que las actividades a realizar se finalicen.

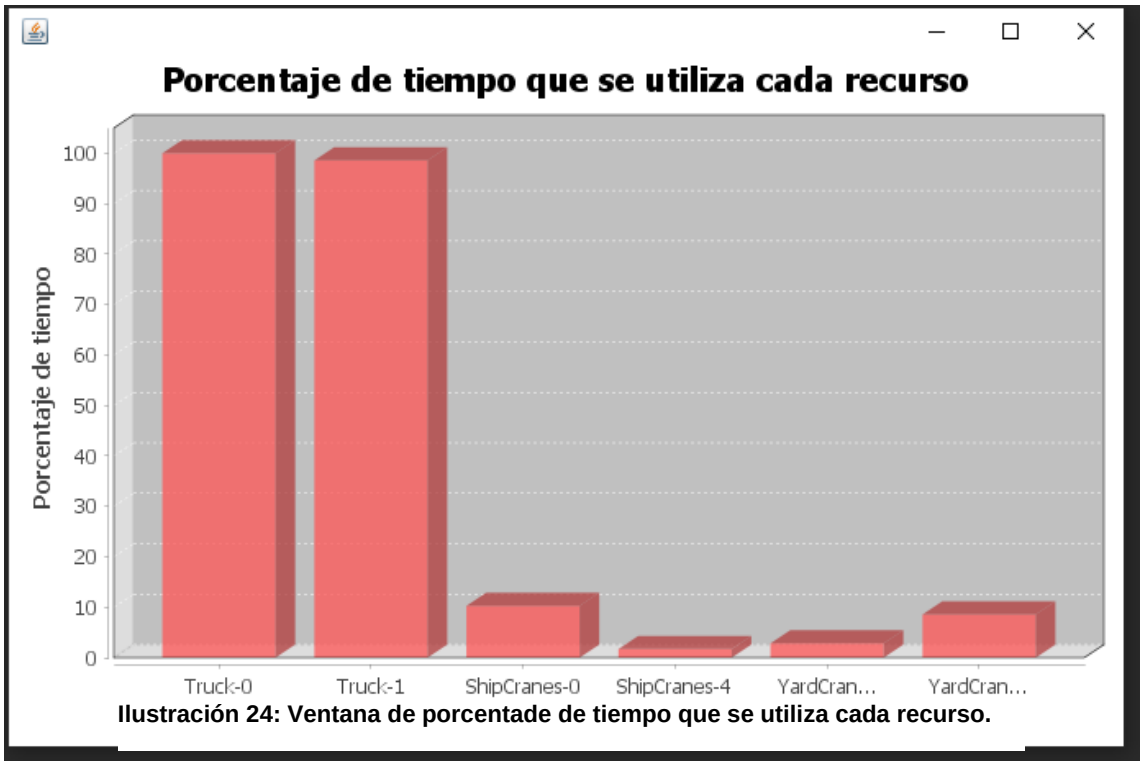


Ilustración 24: Ventana de porcentade de tiempo que se utiliza cada recurso.

Una vez se entiende el procedimiento y las interfaces gráficas, procedemos a realizar los experimentos e indicar lo obtenido en el apartado 3.5. *Resultados*.

3.5. Resultados

Para este apartado se realizarán dos experimentos que servirán como ejemplo para apoyar el correcto funcionamiento de la simulación, teniendo en cuenta los datos previos que tenemos del funcionamiento del puerto de contenedores de Santa Cruz de Tenerife, aportados en el apartado anterior 3.4. Experimentos.

3.5.1. Ejemplo 1

Para este primer ejemplo se aportarán parámetros “conflictivos” para comprobar el funcionamiento de la simulación:

- Número de camiones: 12
- Número de grúas barco: 2
- Número de grúas patio 1: 1
- Número de grúas patio 2: 2
- Número de contenedores: 170

A continuación, ejecutamos el programa con estos valores y obtenemos:

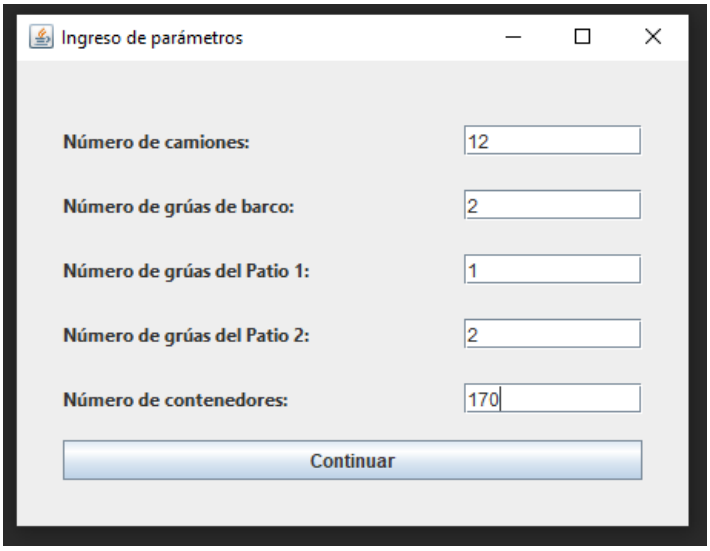


Ilustración 25: Ventana Ingreso de parámetros del ejemplo 1

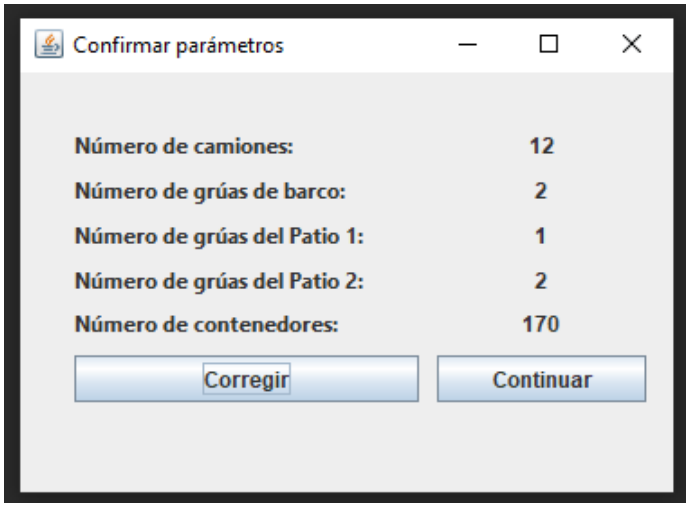
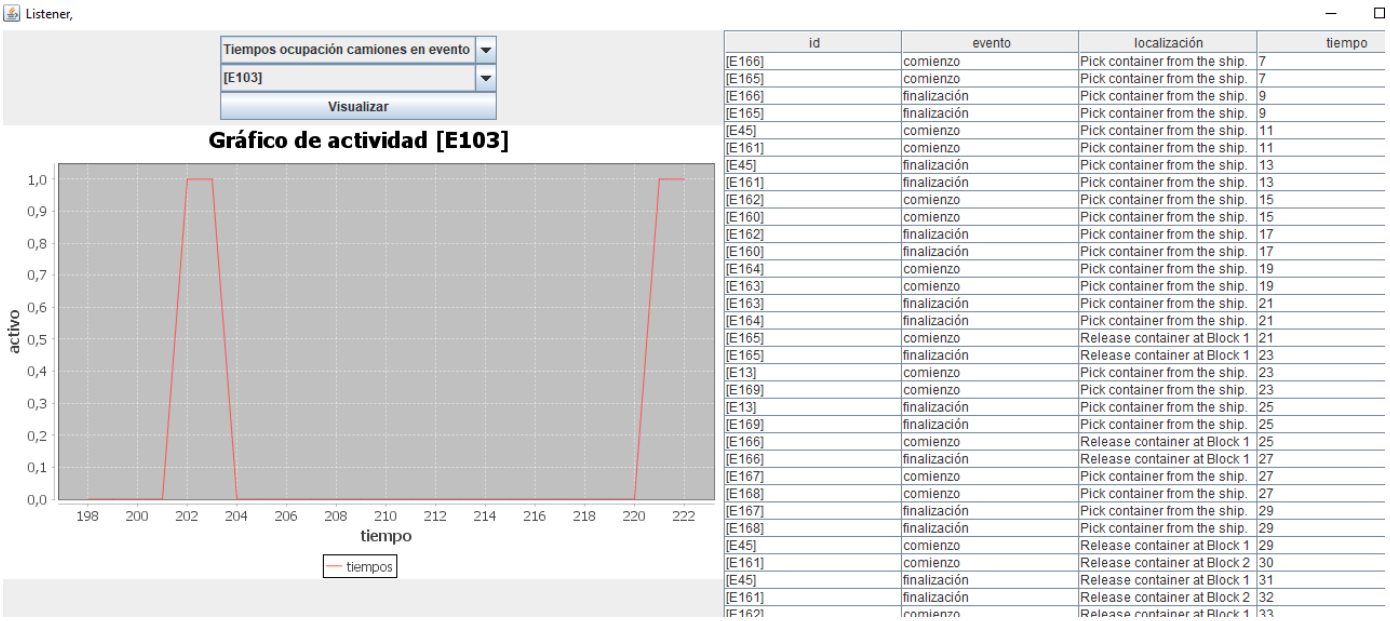


Ilustración 26: Ventana de confirmación de parámetros del ejemplo 1



id	evento	localización	tiempo
[E166]	comienzo	Pick container from the ship.	7
[E165]	comienzo	Pick container from the ship.	7
[E166]	finalización	Pick container from the ship.	9
[E165]	finalización	Pick container from the ship.	9
[E45]	comienzo	Pick container from the ship.	11
[E161]	comienzo	Pick container from the ship.	11
[E45]	finalización	Pick container from the ship.	13
[E161]	finalización	Pick container from the ship.	13
[E162]	comienzo	Pick container from the ship.	15
[E160]	comienzo	Pick container from the ship.	15
[E162]	finalización	Pick container from the ship.	17
[E160]	finalización	Pick container from the ship.	17
[E164]	comienzo	Pick container from the ship.	19
[E163]	comienzo	Pick container from the ship.	19
[E163]	finalización	Pick container from the ship.	21
[E164]	finalización	Pick container from the ship.	21
[E165]	comienzo	Release container at Block 1	21
[E165]	finalización	Release container at Block 1	23
[E13]	comienzo	Pick container from the ship.	23
[E169]	comienzo	Pick container from the ship.	23
[E13]	finalización	Pick container from the ship.	25
[E169]	finalización	Pick container from the ship.	25
[E166]	comienzo	Release container at Block 1	25
[E166]	finalización	Release container at Block 1	27
[E167]	comienzo	Pick container from the ship.	27
[E168]	comienzo	Pick container from the ship.	27
[E167]	finalización	Pick container from the ship.	29
[E168]	finalización	Pick container from the ship.	29
[E45]	comienzo	Release container at Block 1	29
[E161]	comienzo	Release container at Block 2	30
[E45]	finalización	Release container at Block 1	31
[E161]	finalización	Release container at Block 2	32
[E162]	comienzo	Release container at Block 1	33

Ilustración 27: Ventana del gráfico de la actividad y la tabla de tiempo de actividades del ejemplo 1.

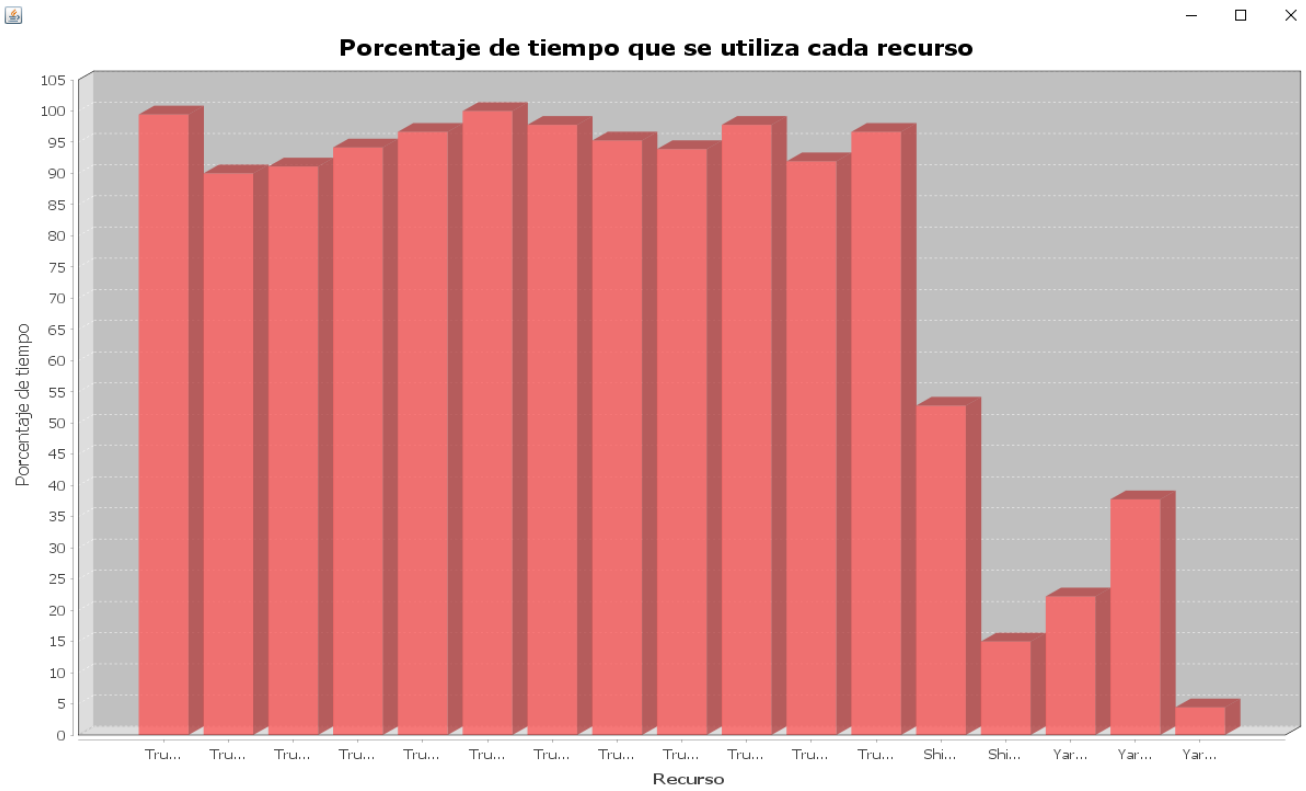


Ilustración 28: Ventana que contiene el tiempo de utilización de cada recurso del ejemplo1

Una vez tenemos las gráficas, procedemos a analizar los datos. En la *Ilustración 28* podemos observar que la cantidad de camiones es muy justa para poder completar la descarga de todos los contenedores en la jornada habitual, ya que la mayoría de estos recursos exceden el 90% del tiempo que se utilizan. Contrastándolo con los datos aportados por Víctor Morín tiene sentido, ya que le hemos asignado el número mínimo de camiones y un número elevado de contenedores.

3.5.2. Ejemplo 2

Para este segundo ejemplo, se aportarán parámetros lo más cerca de la realidad posible, para comprobar el funcionamiento de la simulación y obtener una idea de lo que es el día a día en el puerto. Para ello se ha escogido:

- Número de camiones: 16
- Número de grúas barco: 2
- Número de grúas patio 1: 2
- Número de grúas patio 2: 2
- Número de contenedores: 160

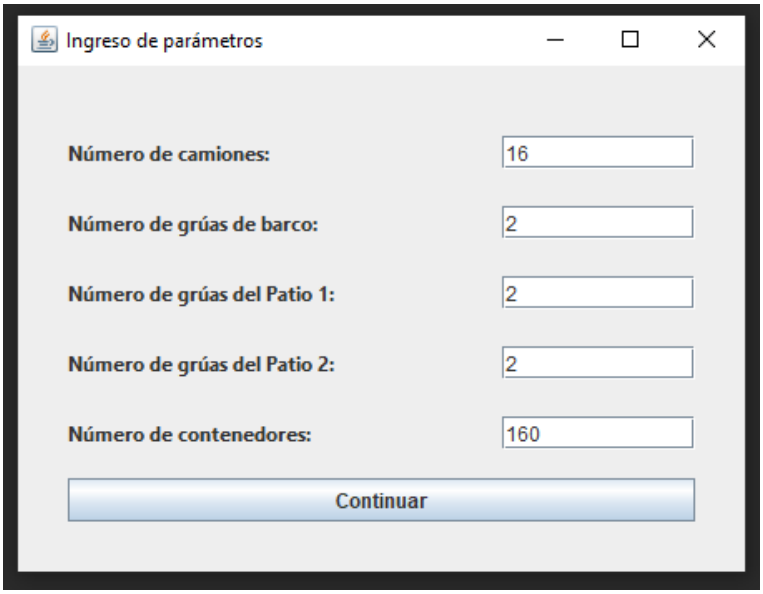


Ilustración 29: Ventana Ingreso de parámetros del ejemplo 2

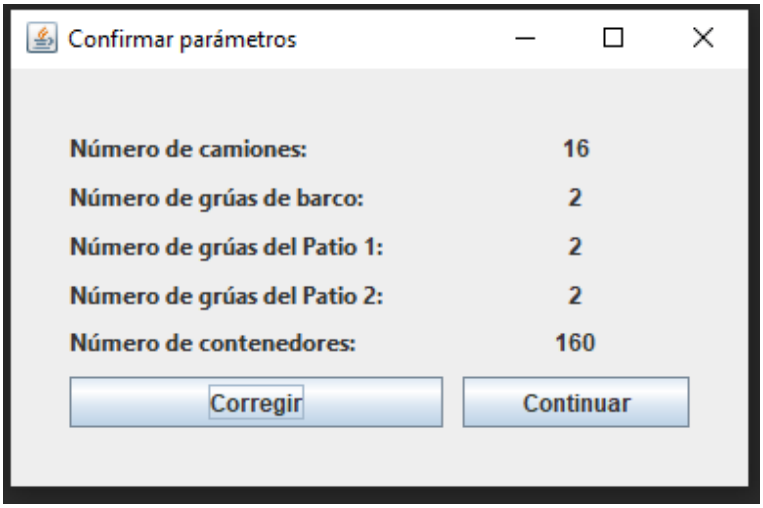
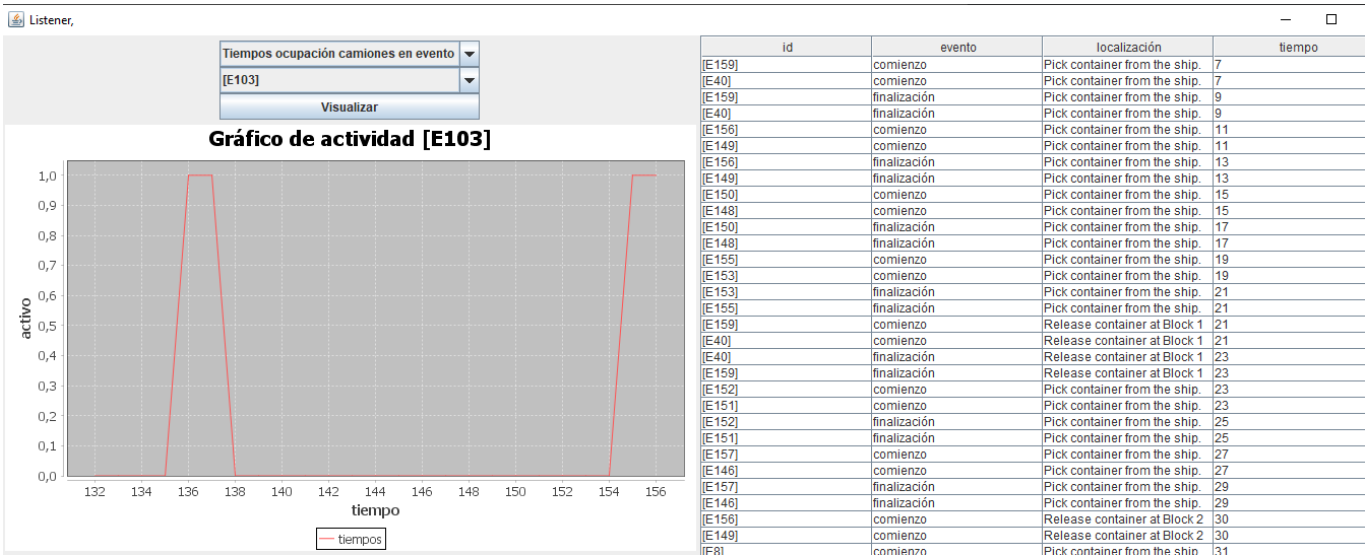


Ilustración 30: Ventana de confirmación de parámetros del ejemplo 2



id	evento	localización	tiempo
[E159]	comienzo	Pick container from the ship.	7
[E40]	comienzo	Pick container from the ship.	7
[E159]	finalización	Pick container from the ship.	9
[E40]	finalización	Pick container from the ship.	9
[E156]	comienzo	Pick container from the ship.	11
[E149]	comienzo	Pick container from the ship.	11
[E156]	finalización	Pick container from the ship.	13
[E149]	finalización	Pick container from the ship.	13
[E150]	comienzo	Pick container from the ship.	15
[E148]	comienzo	Pick container from the ship.	15
[E150]	finalización	Pick container from the ship.	17
[E148]	finalización	Pick container from the ship.	17
[E155]	comienzo	Pick container from the ship.	19
[E153]	comienzo	Pick container from the ship.	19
[E153]	finalización	Pick container from the ship.	21
[E155]	finalización	Pick container from the ship.	21
[E159]	comienzo	Release container at Block 1	21
[E40]	comienzo	Release container at Block 1	21
[E40]	finalización	Release container at Block 1	23
[E159]	finalización	Release container at Block 1	23
[E152]	comienzo	Pick container from the ship.	23
[E151]	comienzo	Pick container from the ship.	23
[E152]	finalización	Pick container from the ship.	25
[E151]	finalización	Pick container from the ship.	25
[E157]	comienzo	Pick container from the ship.	27
[E146]	comienzo	Pick container from the ship.	27
[E157]	finalización	Pick container from the ship.	29
[E146]	finalización	Pick container from the ship.	29
[E156]	comienzo	Release container at Block 2	30
[E149]	comienzo	Release container at Block 2	30
[E8]	comienzo	Pick container from the ship.	31

Ilustración 31: Ventana del gráfico de la actividad y la tabla de tiempo de actividades del ejemplo 2.

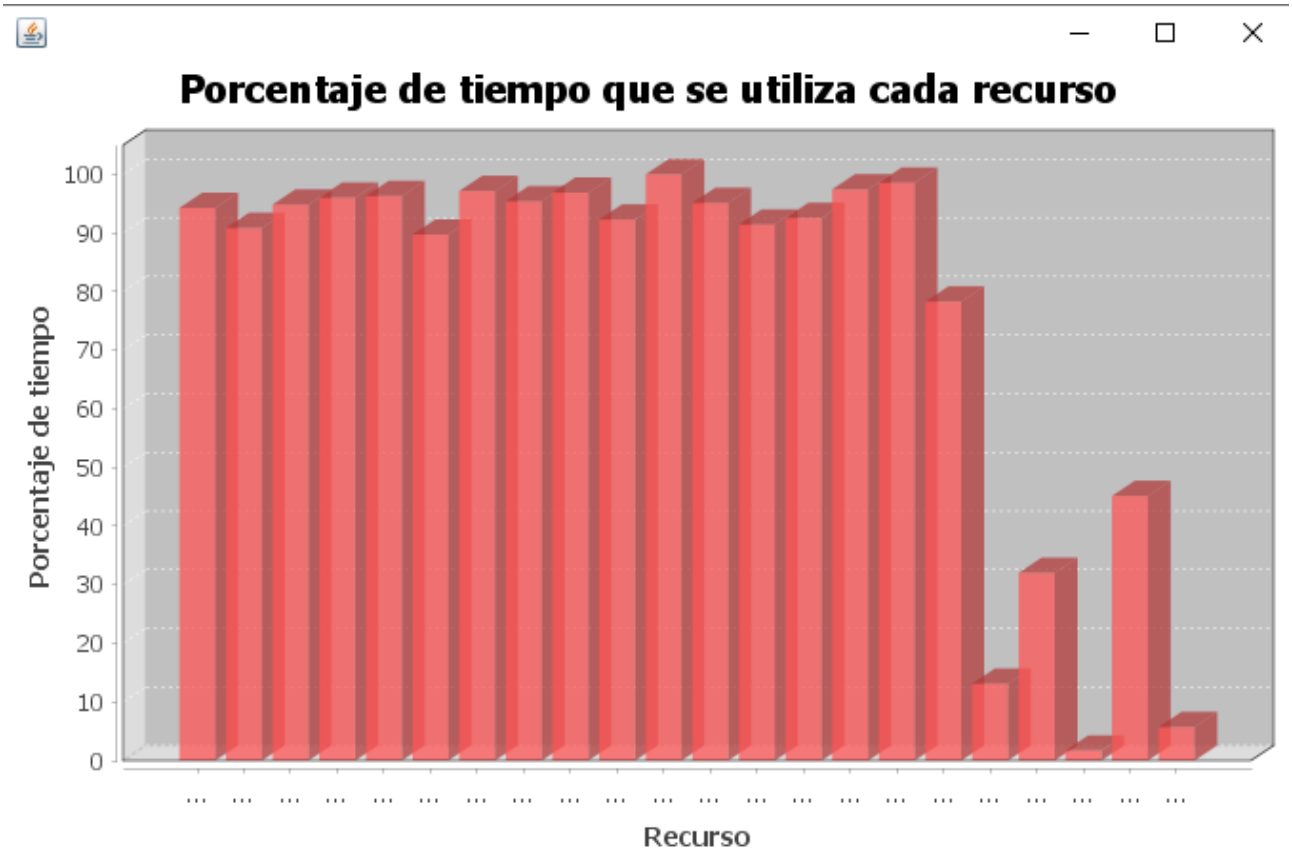


Ilustración 32: Ventana que contiene el tiempo de utilización de cada recurso del ejemplo 2

Una vez tenemos las gráficas del ejemplo 2, procedemos a analizar los datos. En la *Ilustración 32* podemos observar que la cantidad de camiones es menos justa para el número de contenedores que se le asigna, ya que la mayoría de estos recursos están muy cerca del 95% pero no son tantos como los del ejemplo 1. Contrastándolo esto con los datos aportados por Víctor Morín podemos observar que existe una necesidad de ampliar el número de camiones que dispone el puerto, así como tener en mente la instalación de una grúa adicional en el Patio 2, ya que es el patio que más contenedores recibe, pudiendo así prevenir conflictos de tiempo en el caso de que uno de las grúas que ya tiene en el patio se averíe.

4. CONCLUSIONES

Como conclusión a este proyecto podemos decir que se ha conseguido entender el funcionamiento de la zona mar-patio del puerto de Santa Cruz de Tenerife a través del estudio del mismo y de la información dada por Víctor Morín, trabajador del puerto de Santa Cruz. Se ha aprendido y utilizado un nuevo lenguaje de programación, en este caso Java, con la herramienta PSIGHOS, creada por el personal docente de la Universidad de La Laguna, que permite realizar la simulación de eventos discretos.

Además, se han seguido los pasos de la fase del estudio de la simulación, consiguiendo así un proyecto y una memoria estructurada. De esta forma también, hemos creado un modelo de la zona mar-patio del puerto de contenedores de Santa Cruz mucho más afín a la realidad.

Este proyecto resuelve la necesidad de controlar el tiempo que se emplea cada recurso en realizar una actividad, que en nuestro caso es desplazar el contenedor desde el barco a uno de los bloques del patio, además de esto, teniendo una visión de en qué momento se está realizando cada actividad. Se nos abre de igual forma, una gráfica con el porcentaje de utilización en tiempo de cada uno de los recursos que nos permitirá ver cuáles de ellos sobran o si en cambio, hacen falta más unidades del mismo.

En cuanto a mejorar el proyecto, queda abierta la posibilidad de unificar las actividades de las distintas zonas del puerto, como la zona de patio-tierra con el fin de aportar accesibilidad y servir de apoyo a los supervisores que llevan a cabo la organización y gestión de los recursos de las distintas zonas del puerto. Basándose en el diseño de la ventana de resultados de este proyecto, se podrían aplicar un filtro para mostrar los elementos que se pasen de tiempo previsto.

4.1. Conclusions

In conclusion, we can say that it has been possible to understand how the sea-yard area of the port of Santa Cruz de Tenerife works thanks to the study and information given by Víctor Morín, a worker from Santa Cruz's port. We have learned a new programming language, Java, in order to use PSIGHOS tool, created by the teachers of the University of La Laguna, which allows the simulation of discrete events.

In addition, all the steps of the simulation study phase have been understood, in this way, we obtain a project and a structured memory. We have also created a model of the sea-yard area of the container port that is much more in line with reality.

This project solves the need to control the time that each resource uses to carry out an activity, which in our case is to move the container from the ship to one block in the yard, having a vision of when each of the activities is being carried out. In the same way, a graph is show to us with the percentage of use in time of each of the resources. It is also open to add new graphs or have the possibility of filtering the elements that exceed the expected time.

This project gives the possibility of unifying the activities of the different areas of the port, such as the yard-land area in order to provide accessibility to supervisors, who carry out the organization and management of resources.

5. BIBLIOGRAFÍA.

[1]: https://definicion.de/puerto/

[2]: http://bibing.us.es/proyectos/abreproy/70181/fichero/Trabajo+Fin+de+M%C3%A1ster+Planificaci%C3%B3n+de+terminales+portuarias+de+contenedores%252F4.+Capitulo+2.pdf

[4]: http://www.comunidadandina.org/DS/Manual%20Contenedores.pdf

[3]: https://www.noatummaritime.com/tipo-de-contenedores-maritimos-estandar/

[5]: https://www.gruasyaparejos.com/grua-portico-para-contenedores/gruas-de-patio/

[6]: https://masqueingenieria.com/blog/que-es-una-terminal-portuaria/

[7]: http://www.iol.etsii.upm.es/arch/intro_simulacion.pdf

[8]: http://www.tctenerife.es/nosotros/proyecto/

6. ANEXOS.

```java
package gui;

import com.sun.org.apache.xml.internal.security.Init;
import harbor.HarborExperiment;
import infoReceiver.HarborLocationListener;

import javax.sound.sampled.Line;
import javax.swing.*;

public class Main {

    public int nTrucksM ;
    public int nCranesM;
    public int nYardCranesM1;
    public int nYardCranesM2;
    public int nContainersM;


    public Main (int nTrucksM, int nCranesM, int nYardCranesM1, int nYardCranesM2, int nContainersM){
        this.nTrucksM = nTrucksM;
        this.nCranesM = nCranesM;
        this.nYardCranesM1 = nYardCranesM1;
        this.nYardCranesM2 = nYardCranesM2;
        this.nContainersM = nContainersM;

        int nYardCranes[] = {nYardCranesM1, nYardCranesM2};

        // Start simulation on text mode
        int startTS = 0;
        int endTS = 480;



        //InitWindow initWindow = new InitWindow();
        HarborExperiment experiment = new HarborExperiment(endTS, startTS, nCranesM, nTrucksM, nYardCranes, nContainersM);
        experiment.start();



    }

    public static void main(String[] args) {

        JFrame w = new InitWindow();
        // JFrame w = new JFrame("Ingreso de parámetros");
        // w.setContentPane(new InitWindow().panel1);
        w.pack();
        w.setVisible(true);


```

```
51
52      }
53  }
54
55
56
```

```java
package gui;

import infoReceiver.HarborLocationListener;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class ResultsTable extends JFrame {

    //el listener captura todos los cambios en el simulador para poder manejarlos
    private HarborLocationListener observer;
    //para mostrar los resultados en gráficos
    public JFreeChart grafico;

    /*
     * Constructor. Le pasamos el Listener que contiene la información del experimento del cual queremos
     * mostrar el resultado.
     */
    public ResultsTable(HarborLocationListener observer) {
        //título de la ventana, info del listener mas el tiempo empleado
        super(observer.toString() + ", " );
        //layout rejilla con dos columnas (para la parte gráfica y la tabla)
        this.setLayout(new GridLayout(0, 2));
        this.observer = observer;
        //genero la tabla con el modelo de la clase interna de esta propia clase
        JTable tabla = new JTable(new TableModel(observer));
        //pone barras desplazamiento a la tabla, ya que es bastante grande
        JScrollPane s = new JScrollPane(tabla);

        //añade al frame el panel de los selectores del gráfico
        getContentPane().add(combos());

        getContentPane().add(s);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
47          pack();
48      }
49
50      /*
51       * Crea y devuelve los selectores del gráfico así como el propio gráfico.
52       */
53      private JPanel combos() {
54          JPanel panel = new JPanel();
55          JPanel botones = new JPanel(new GridLayout(0,1));
56          //los valores seleccionables del combo de abajo son los ids de los elementos
57          JComboBox<String> combo = new JComboBox<String>(observer.getIds());
58          //los del combo de arriba son los dos tipos de gráficos implementadoss
59          String [] tipos = {"Tiempos ocupación grúa en evento", /*"Porcentaje ocupación por evento",*/ "Tiempos ocupación camiones en evento",/*"Tiempos"*/};
60          JComboBox<String> tipoGrafico = new JComboBox(tipos);
61          JButton aceptar = new JButton("Visualizar");
62
63          aceptar.addActionListener(new ActionListener() {
64
65              /*
66               * Captura el evento del click en el boton y genera los gráficos.
67               */
68              @Override
69              public void actionPerformed(ActionEvent arg0) {
70                  //selecciona en función del tipo del combo superior
71                  switch ((String) tipoGrafico.getSelectedItem()) {
72
73
74                      case ("Tiempos ocupación grúa en evento"):
75                          //crea un gráfico de coordenadas x y donde cada punto será 0 si en ese instante el elemento estaba
76                          //inactivo y 1 si estaba usado.
77                          XYSeries datos = new XYSeries("tiempos");
78                          //por defecto el valor que se escribe es 0 (inactivo)
79                          double valor = 0;
80                          //empezamos el muestreo en la posición 3 unidades anterior al primer cambio
81                          long actual = observer.getTiempos((String) combo.getSelectedItem()).get(0) - 3;
82
83                          //para cada tiempo registrado sabemos que van en parejas de dos, el primero activa y el segundo
```

```
83 desactiva
84                         //mientras no haya cambios en el
   momento actual, se escribe el último valor recordado
85                         //cuando se llega a un valor que está
   presente, se pinta y se cambia el valor a pintar al opuesto (
   valor+1)%2
86                         for (Long tiempo : observer.getTiempos
   ((String) combo.getSelectedItem())) {
87
88                             while (actual < tiempo)
89                                 datos.add(actual++, valor);
90                             datos.add(actual++, valor);
91                             valor = (valor + 1) % 2;
92                         }
93                         //terminado de plotear los datos, se
   elimina todos los elementos del panel para actualizar el
   gráfico
94                         panel.removeAll();
95                         //se añaden de nuevo los botones
96                         panel.add(botones);
97                         tipoGrafico.setSelectedIndex(1);
98                         //se añade el gráfico de timpo XY con
   los valores calculados.
99                         panel.add(new ChartPanel(
100                                ChartFactory.createXYLineChart
   ("Gráfico de actividad " + (String) combo.getSelectedItem(),
101                                                 "tiempo", "activo",
   new XYSeriesCollection(datos), PlotOrientation.VERTICAL, true
   , true, false)));
102
103                        repintar();
104
105                        break;
106                    case ("Tiempos ocupación camiones en
   evento"):
107                        //crea un gráfico de coordenadas x y
   donde cada punto será 0 si en ese instante el elemento estaba
108                        //inactivo y 1 si estaba usado.
109                        XYSeries datoscamiones = new XYSeries(
   "tiempos");
110                        //por defecto el valor que se escribe
   es 0 (inactivo)
111                        double valorcamiones = 1;
112                        //empezamos el muestreo en la posición
    3 unidades anterior al primer cambio
113                        long actualcamiones = observer.
   getTiempos((String) combo.getSelectedItem()).get(0) - 3;
114
115                        //para cada tiempo registrado sabemos
   que van en parejas de dos, el primero activa y el segudno
   desactiva
116                        //mientras no haya cambio en el
```

```
116 momento actual, se escribe el último valor recordado
117                         //cuando se llega a un valor que está
    presente, se pinta y se cambia el valor a pintar al opuesto (
    valor+1)%2
118                         for (Long tiempo : observer.getTiempos
    ((String) combo.getSelectedItem())) {
119                             while (actualcamiones < tiempo)
120                                 datoscamiones.add(
    actualcamiones++, valorcamiones);
121                             datoscamiones.add(actualcamiones
    ++, valorcamiones);
122                             valorcamiones = (valorcamiones + 1
    ) % 2;
123                         }
124                         //terminado de plotear los datos, se
    elimina todos los elementos del panel para actualizar el
    gráfico
125                         panel.removeAll();
126                         //se añaden de nuevo los botones33
127                         panel.add(botones);
128                         tipoGrafico.setSelectedIndex(1);
129                         //se añade el gráfico de timpo XY con
    los valores calculados.
130                         panel.add(new ChartPanel(
131                                 ChartFactory.createXYLineChart
    ("Gráfico de actividad " + (String) combo.getSelectedItem(),
132                                         "tiempo", "activo",
    new XYSeriesCollection(datoscamiones), PlotOrientation.
    VERTICAL, true, true, false)));
133
134                         repintar();
135
136                         break;
137                     //grafo de tarta
138                     case "Porcentaje ocupación por evento":
139                         DefaultPieDataset datos2 = new
    DefaultPieDataset();
140                         long inicial = observer.getTiempos((
    String) combo.getSelectedItem()).get(0);
141                         double activo =0;
142                         long fin =0;
143                         long anterior = inicial;
144                         boolean activado = false;
145                         /*
146                          * El algoritmo consiste en contar
    cuanto tiempo está activo apra calcular el porcentaje de
    ocupación
147                          * Recordamos en anterior cual fue el
    último cambio de estado para saber el tamaño del intervalo.
148                          */
149                         for (Long tiempo : observer.getTiempos
    ((String) combo.getSelectedItem())) {
```

```
150                             fin = tiempo;
151                             if(!activado) {
152                                 activado = true;
153                                 anterior = tiempo;
154                             }else {
155                                 activado = false;
156                                 activo+= (tiempo-anterior);
157                             }
158 
159                         }
160                         double porcentaje = activo/(fin-
    inicial);
161                         //calculamos los porcentajes y los
    añadimos al gráfico
162                         datos2.setValue("Uso de grúas",
    porcentaje);
163                         datos2.setValue("Uso de camiones", 1-
    porcentaje);
164                         panel.removeAll();
165                         panel.add(botones);
166                         panel.add(new ChartPanel(
167                                 ChartFactory.createPieChart("
    Porcentaje de uso "+ (String) combo.getSelectedItem(),
168                                         datos2, true,true,
    false)));
169                         repintar();
170                         break;
171 
172                     //grafo tarta total
173                     case "Porcentaje ocupación total":
174                         long tiempo_total_gruas = 0;
175                         DefaultPieDataset datos3 = new
    DefaultPieDataset();
176 
177                         /*
178                          * El algoritmo consiste en contar
    cuanto tiempo está activo apra calcular el porcentaje de
    ocupación
179                          * Recordamos en anterior cual fue el
    último cambio de estado para saber el tamaño del intervalo.
180                          */
181                         for(int i = 0; i < observer.tamMapa
    (); i++) {
182                             long inicial2 = observer.
    getTiempos((String) combo.getItemAt(i)).get(0);
183                             double activo2 =0;
184                             long fin2 =0;
185                             long anterior2 = inicial2;
186                             boolean activado2 = false;
187                             for (Long tiempo : observer.
    getTiempos((String) combo.getItemAt(i)) ){
188                                 fin2 = tiempo;
```

```
189                                     if(!activado2) {
190                                         activado2 = true;
191                                         anterior2 = tiempo;
192                                     }else {
193                                         activado2 = false;
194                                         activo2+= (tiempo-
    anterior2);
195                                         tiempo_total_gruas +=
    activo2;
196                                     }
197                                 }
198                             }
199 
200                             String[] arrofstring = observer.
    toString().split(" ");
201                             double resultado = Math.abs(observer.
    getFin() - (tiempo_total_gruas / Integer.parseInt(arrofstring[
    3])));
202                             double porcentaje2 = resultado/(double
    ) observer.getFin();
203                             //calculamos los porcentajes y los
    añadimos al gráfico
204                             datos3.setValue("Uso de grúas total",
    porcentaje2);
205                             datos3.setValue("Uso de camiones total
    ", 1-porcentaje2);
206                             panel.removeAll();
207                             panel.add(botones);
208                             panel.add(new ChartPanel(
209                                     ChartFactory.createPieChart("
    Porcentaje de uso "+ (String) combo.getSelectedItem(),
210                                             datos3, true,true,
    false)));
211                             repintar();
212 
213 
214                 }
215 
216             }
217         });
218         /*
219          * Además de los gráficos con datos, en la
    construcción del frame se crea un gráfico vacío para guardar
    los espacios
220          * y mantener una estructura..
221          */
222         DefaultCategoryDataset datos = new
    DefaultCategoryDataset();
223         grafico = ChartFactory.createLineChart("Gráfico de
    actividad " + (String) combo.getSelectedItem(),
224                 "tiempo", "activo", datos, PlotOrientation.
    VERTICAL, true, true, false);
```

```
225         botones.add(tipoGrafico);
226         botones.add(combo);
227         botones.add(aceptar);
228         panel.add(botones);
229         panel.add(new ChartPanel(grafico));
230         return panel;
231     }
232 
233     private void repintar() {
234         setVisible(false);
235         setVisible(true);
236     }
237 }
238 
239 /*
240  * Modelo de datos para rellenar la tabla de resultados.
241  * Hereda de AbstractTableModel que es quien realiza toda la
    labor de presentación de datos, se sobreescriben
242  * los métodos necesarios para que dicho modelo encuentre los
    datos que necesita
243  */
244 class TableModel extends AbstractTableModel {
245 
246     private HarborLocationListener observer;
247     /*
248      * Constructor, el modelo obtiene toda su información
    directamente del listener. Modificando el
249      * listener se cambia el comportamiento e información de
    la tabla
250      */
251     TableModel(HarborLocationListener observer) {
252         super();
253         this.observer = observer;
254     }
255     //devuelve el número de columnas que debe dibujar, la
    información depende del observer
256     @Override
257     public int getColumnCount() {
258         return observer.columnas();
259     }
260 
261     //obtiene el nombre de la columna numerada según el
    parametro
262     @Override
263     public String getColumnName(int column) {
264         return observer.cabecera(column);
265     }
266 
267     //obtiene el número de filas que debe mostrar
268     @Override
269     public int getRowCount() {
270         return observer.filas();
```

```
271     }
272
273     //obtiene el valor para la celda x,y obtenida como
    parámetro
274     @Override
275     public Object getValueAt(int fila, int col) {
276         return observer.get(fila, col);
277     }
278
279
280
281 }
282
283
```

```java
package gui;

import infoReceiver.HarborLocationListener;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class ResultsTable extends JFrame {

    //el listener captura todos los cambios en el simulador para poder manejarlos
    private HarborLocationListener observer;
    //para mostrar los resultados en gráficos
    public JFreeChart grafico;

    /*
     * Constructor. Le pasamos el Listener que contiene la información del experimento del cual queremos
     * mostrar el resultado.
     */
    public ResultsTable(HarborLocationListener observer) {
        //título de la ventana, info del listener mas el tiempo empleado
        super(observer.toString() + ", " );
        //layout rejilla con dos columnas (para la parte gráfica y la tabla)
        this.setLayout(new GridLayout(0, 2));
        this.observer = observer;
        //genero la tabla con el modelo de la clase interna de esta propia clase
        JTable tabla = new JTable(new TableModel(observer));
        //pone barras desplazamiento a la tabla, ya que es bastante grande
        JScrollPane s = new JScrollPane(tabla);

        //añade al frame el panel de los selectores del gráfico
        getContentPane().add(combos());

        getContentPane().add(s);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
47          pack();
48      }
49
50      /*
51       * Crea y devuelve los selectores del gráfico así como el propio gráfico.
52       */
53      private JPanel combos() {
54          JPanel panel = new JPanel();
55          JPanel botones = new JPanel(new GridLayout(0,1));
56          //los valores seleccionables del combo de abajo son los ids de los elementos
57          JComboBox<String> combo = new JComboBox<String>(observer.getIds());
58          //los del combo de arriba son los dos tipos de gráficos implementadoss
59          String [] tipos = {"Tiempos ocupación grúa en evento", /*"Porcentaje ocupación por evento",*/ "Tiempos ocupación camiones en evento",/*"Tiempos"*/};
60          JComboBox<String> tipoGrafico = new JComboBox(tipos);
61          JButton aceptar = new JButton("Visualizar");
62
63          aceptar.addActionListener(new ActionListener() {
64
65              /*
66               * Captura el evento del click en el boton y genera los gráficos.
67               */
68              @Override
69              public void actionPerformed(ActionEvent arg0) {
70                  //selecciona en función del tipo del combo superior
71                  switch ((String) tipoGrafico.getSelectedItem()) {
72
73
74                      case ("Tiempos ocupación grúa en evento"):
75                          //crea un gráfico de coordenadas x y donde cada punto será 0 si en ese instante el elemento estaba
76                          //inactivo y 1 si estaba usado.
77                          XYSeries datos = new XYSeries("tiempos");
78                          //por defecto el valor que se escribe es 0 (inactivo)
79                          double valor = 0;
80                          //empezamos el muestreo en la posición 3 unidades anterior al primer cambio
81                          long actual = observer.getTiempos((String) combo.getSelectedItem()).get(0) - 3;
82
83                          //para cada tiempo registrado sabemos que van en parejas de dos, el primero activa y el segundo
```

```
83 desactiva
84                         //mientras no haya cambios en el
   momento actual, se escribe el último valor recordado
85                         //cuando se llega a un valor que está
   presente, se pinta y se cambia el valor a pintar al opuesto (
   valor+1)%2
86                         for (Long tiempo : observer.getTiempos
   ((String) combo.getSelectedItem())) {
87
88                             while (actual < tiempo)
89                                 datos.add(actual++, valor);
90                             datos.add(actual++, valor);
91                             valor = (valor + 1) % 2;
92                         }
93                         //terminado de plotear los datos, se
   elimina todos los elementos del panel para actualizar el
   gráfico
94                         panel.removeAll();
95                         //se añaden de nuevo los botones
96                         panel.add(botones);
97                         tipoGrafico.setSelectedIndex(1);
98                         //se añade el gráfico de timpo XY con
   los valores calculados.
99                         panel.add(new ChartPanel(
100                                ChartFactory.createXYLineChart
   ("Gráfico de actividad " + (String) combo.getSelectedItem(),
101                                                 "tiempo", "activo",
   new XYSeriesCollection(datos), PlotOrientation.VERTICAL, true
   , true, false)));
102
103                        repintar();
104
105                        break;
106                    case ("Tiempos ocupación camiones en
   evento"):
107                        //crea un gráfico de coordenadas x y
   donde cada punto será 0 si en ese instante el elemento estaba
108                        //inactivo y 1 si estaba usado.
109                        XYSeries datoscamiones = new XYSeries(
   "tiempos");
110                        //por defecto el valor que se escribe
   es 0 (inactivo)
111                        double valorcamiones = 1;
112                        //empezamos el muestreo en la posición
    3 unidades anterior al primer cambio
113                        long actualcamiones = observer.
   getTiempos((String) combo.getSelectedItem()).get(0) - 3;
114
115                        //para cada tiempo registrado sabemos
   que van en parejas de dos, el primero activa y el segudno
   desactiva
116                        //mientras no haya cambio en el
```

```
116 momento actual, se escribe el último valor recordado
117                         //cuando se llega a un valor que está
    presente, se pinta y se cambia el valor a pintar al opuesto (
    valor+1)%2
118                         for (Long tiempo : observer.getTiempos
    ((String) combo.getSelectedItem())) {
119                             while (actualcamiones < tiempo)
120                                 datoscamiones.add(
    actualcamiones++, valorcamiones);
121                             datoscamiones.add(actualcamiones
    ++, valorcamiones);
122                             valorcamiones = (valorcamiones + 1
    ) % 2;
123                         }
124                         //terminado de plotear los datos, se
    elimina todos los elementos del panel para actualizar el
    gráfico
125                         panel.removeAll();
126                         //se añaden de nuevo los botones33
127                         panel.add(botones);
128                         tipoGrafico.setSelectedIndex(1);
129                         //se añade el gráfico de timpo XY con
    los valores calculados.
130                         panel.add(new ChartPanel(
131                                 ChartFactory.createXYLineChart
    ("Gráfico de actividad " + (String) combo.getSelectedItem(),
132                                         "tiempo", "activo",
    new XYSeriesCollection(datoscamiones), PlotOrientation.
    VERTICAL, true, true, false)));
133
134                         repintar();
135
136                         break;
137                     //grafo de tarta
138                     case "Porcentaje ocupación por evento":
139                         DefaultPieDataset datos2 = new
    DefaultPieDataset();
140                         long inicial = observer.getTiempos((
    String) combo.getSelectedItem()).get(0);
141                         double activo =0;
142                         long fin =0;
143                         long anterior = inicial;
144                         boolean activado = false;
145                         /*
146                          * El algoritmo consiste en contar
    cuanto tiempo está activo apra calcular el porcentaje de
    ocupación
147                          * Recordamos en anterior cual fue el
    último cambio de estado para saber el tamaño del intervalo.
148                          */
149                         for (Long tiempo : observer.getTiempos
    ((String) combo.getSelectedItem())) {
```

```
150                             fin = tiempo;
151                             if(!activado) {
152                                 activado = true;
153                                 anterior = tiempo;
154                             }else {
155                                 activado = false;
156                                 activo+= (tiempo-anterior);
157                             }
158 
159                         }
160                         double porcentaje = activo/(fin-
    inicial);
161                         //calculamos los porcentajes y los
    añadimos al gráfico
162                         datos2.setValue("Uso de grúas",
    porcentaje);
163                         datos2.setValue("Uso de camiones", 1-
    porcentaje);
164                         panel.removeAll();
165                         panel.add(botones);
166                         panel.add(new ChartPanel(
167                                 ChartFactory.createPieChart("
    Porcentaje de uso "+ (String) combo.getSelectedItem(),
168                                         datos2, true,true,
    false)));
169                         repintar();
170                         break;
171 
172                     //grafo tarta total
173                     case "Porcentaje ocupación total":
174                         long tiempo_total_gruas = 0;
175                         DefaultPieDataset datos3 = new
    DefaultPieDataset();
176 
177                         /*
178                          * El algoritmo consiste en contar
    cuanto tiempo está activo apra calcular el porcentaje de
    ocupación
179                          * Recordamos en anterior cual fue el
    último cambio de estado para saber el tamaño del intervalo.
180                          */
181                         for(int i = 0; i < observer.tamMapa
    (); i++) {
182                             long inicial2 = observer.
    getTiempos((String) combo.getItemAt(i)).get(0);
183                             double activo2 =0;
184                             long fin2 =0;
185                             long anterior2 = inicial2;
186                             boolean activado2 = false;
187                             for (Long tiempo : observer.
    getTiempos((String) combo.getItemAt(i)) ){
188                                 fin2 = tiempo;
```

```
189                                     if(!activado2) {
190                                         activado2 = true;
191                                         anterior2 = tiempo;
192                                     }else {
193                                         activado2 = false;
194                                         activo2+= (tiempo-
    anterior2);
195                                         tiempo_total_gruas +=
    activo2;
196                                     }
197                                 }
198                             }
199 
200                             String[] arrofstring = observer.
    toString().split(" ");
201                             double resultado = Math.abs(observer.
    getFin() - (tiempo_total_gruas / Integer.parseInt(arrofstring[
    3])));
202                             double porcentaje2 = resultado/(double
    ) observer.getFin();
203                             //calculamos los porcentajes y los
    añadimos al gráfico
204                             datos3.setValue("Uso de grúas total",
    porcentaje2);
205                             datos3.setValue("Uso de camiones total
    ", 1-porcentaje2);
206                             panel.removeAll();
207                             panel.add(botones);
208                             panel.add(new ChartPanel(
209                                     ChartFactory.createPieChart("
    Porcentaje de uso "+ (String) combo.getSelectedItem(),
210                                             datos3, true,true,
    false)));
211                             repintar();
212 
213 
214                 }
215 
216             }
217         });
218         /*
219          * Además de los gráficos con datos, en la
    construcción del frame se crea un gráfico vacío para guardar
    los espacios
220          * y mantener una estructura..
221          */
222         DefaultCategoryDataset datos = new
    DefaultCategoryDataset();
223         grafico = ChartFactory.createLineChart("Gráfico de
    actividad " + (String) combo.getSelectedItem(),
224                 "tiempo", "activo", datos, PlotOrientation.
    VERTICAL, true, true, false);
```

```
225         botones.add(tipoGrafico);
226         botones.add(combo);
227         botones.add(aceptar);
228         panel.add(botones);
229         panel.add(new ChartPanel(grafico));
230         return panel;
231     }
232 
233     private void repintar() {
234         setVisible(false);
235         setVisible(true);
236     }
237 }
238 
239 /*
240  * Modelo de datos para rellenar la tabla de resultados.
241  * Hereda de AbstractTableModel que es quien realiza toda la
    labor de presentación de datos, se sobreescriben
242  * los métodos necesarios para que dicho modelo encuentre los
    datos que necesita
243  */
244 class TableModel extends AbstractTableModel {
245 
246     private HarborLocationListener observer;
247     /*
248      * Constructor, el modelo obtiene toda su información
    directamente del listener. Modificando el
249      * listener se cambia el comportamiento e información de
    la tabla
250      */
251     TableModel(HarborLocationListener observer) {
252         super();
253         this.observer = observer;
254     }
255     //devuelve el número de columnas que debe dibujar, la
    información depende del observer
256     @Override
257     public int getColumnCount() {
258         return observer.columnas();
259     }
260 
261     //obtiene el nombre de la columna numerada según el
    parametro
262     @Override
263     public String getColumnName(int column) {
264         return observer.cabecera(column);
265     }
266 
267     //obtiene el número de filas que debe mostrar
268     @Override
269     public int getRowCount() {
270         return observer.filas();
```

```
271     }
272
273     //obtiene el valor para la celda x,y obtenida como
    parámetro
274     @Override
275     public Object getValueAt(int fila, int col) {
276         return observer.get(fila, col);
277     }
278
279
280
281 }
282
283
```

```java
package gui;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ConfirmationWindow extends JFrame {
    public JPanel mainPanel;
    public JPanel panel2;

    public JButton continuarButton;
    public JButton corregirButton;
    public JLabel Textfield1;
    public JLabel Textfield2;
    public JLabel Textfield3;
    public JLabel Textfield4;
    public JLabel Textfield5;

    private InitWindow pantallaInicio;


    public ConfirmationWindow(InitWindow pantallaInicio, JTextField TextField1, JTextField TextField2, JTextField TextField3, JTextField TextField4, JTextField TextField5  ) {
        super("Confirmar parámetros");
        this.pantallaInicio = pantallaInicio;

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setContentPane(panel2);



        String txt1 = TextField1.getText();
        String txt2 = TextField2.getText();
        String txt3 = TextField3.getText();
        String txt4 = TextField4.getText();
        String txt5 = TextField5.getText();

        Textfield1.setText(txt1);
        Textfield2.setText(txt2);
        Textfield3.setText(txt3);
        Textfield4.setText(txt4);
        Textfield5.setText(txt5);



        corregirButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){

                pantallaInicio.setVisible(true);
                ConfirmationWindow.this.dispose();
            }
```

```
51
52          });
53
54          continuarButton.addActionListener(new ActionListener
    () {
55              public void actionPerformed(ActionEvent e){
56                  int var1 = Integer.parseInt(txt1);
57                  int var2 = Integer.parseInt(txt2);
58                  int var3 = Integer.parseInt(txt3);
59                  int var4 = Integer.parseInt(txt4);
60                  int var5 = Integer.parseInt(txt5);
61                  int var34[]= {var3, var4} ;
62                  Main parameters = new Main(var1, var2, var3 ,
    var4, var5);
63                  //TimeInfo info = new TimeInfo(var1,var2,var34
    ,var5);
64
65
66
67                  ConfirmationWindow.this.dispose();
68              }
69
70          });
71      }
72  }
73
```

```java
package gui;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;


public class InitWindow extends JFrame {
  public JPanel panel1;
  public JPanel mainPanel;

  // Labels
  private JLabel nTrucksLabel;
  private JLabel nCranesShipLabel;
  private JLabel nCranesYard1Label;
  private JLabel nContainersLabel;

  // Fields
  public JTextField nTrucksField;
  public JTextField nCranesShipField;
  public JTextField nYardCranes1Field;
  public JTextField nYardCranes2Field;
  public JTextField nContainersField;
  public JButton continueButton;


  public InitWindow() {
    super ("Ingreso de parámetros");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setContentPane(panel1);

    continueButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        System.out.println("Número de camiones: " +
nTrucksField.getText());
        System.out.println("Número de gruas de barco: " +
nCranesShipField.getText());
        System.out.println("Número de gruas de patio 1: " +
nYardCranes1Field.getText());
        System.out.println("Número de gruas de patio 2: " +
nYardCranes2Field.getText());
        System.out.println("Número de contenedores: " +
nContainersField.getText());


        JFrame wc = new ConfirmationWindow(InitWindow.this,
nTrucksField,nCranesShipField,nYardCranes1Field,
nYardCranes2Field,nContainersField);
        wc.pack();
        wc.setVisible(true);
        InitWindow.this.setVisible(false);
```

```
46
47          }
48      });
49
50    }
51
52
53
54  }
55
```

```java
package gui;
import java.awt.BorderLayout;
import java.util.TreeMap;

import javax.swing.JFrame;
import javax.swing.JPanel;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import es.ull.iis.simulation.model.Resource;

public class timePercentageGraph extends JFrame {
    public JPanel Graphic;
    public JPanel jPanel;


    public timePercentageGraph(TreeMap<Resource, Long> truckIdleTime, TreeMap<Resource, Long> CraneIdleTime, TreeMap<Resource, Long> YardCraneIdleTime, long tMax) {
        super ("Tiempo Utilizacion");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setContentPane(Graphic);
        DefaultCategoryDataset data = new DefaultCategoryDataset();

        for (Resource resource : truckIdleTime.keySet()){
            double d = (double) truckIdleTime.get(resource);
            data.setValue((d / tMax) * 100, "Porcentaje de tiempo utilizado", resource.getDescription());
            System.out.println(data);
        }
        for (Resource recurso : CraneIdleTime.keySet()){
            double d = (double) CraneIdleTime.get(recurso);
            data.setValue((d / tMax) * 100, "Porcentaje de tiempo utilizado", recurso.getDescription());
            System.out.println(data);
        }
        for (Resource recurso : YardCraneIdleTime.keySet()){
            double d = (double) YardCraneIdleTime.get(recurso);
            data.setValue((d / tMax) * 100, "Porcentaje de tiempo utilizado", recurso.getDescription());
        }

        JFreeChart chart = ChartFactory.createBarChart3D(
                "Porcentaje de tiempo que se utiliza cada recurso",
                "Recurso",
                "Porcentaje de tiempo",
```

```
46                  data,
47                  PlotOrientation.VERTICAL,
48                  false,
49                  true,
50                  false);
51
52          ChartPanel chartpanel = new ChartPanel(chart);
53
54
55          JPanel jPanel = new JPanel();
56          jPanel.setLayout(new BorderLayout());
57          jPanel.add(chartpanel, BorderLayout.CENTER);
58
59          JFrame Grafico = new JFrame();
60          Grafico.getContentPane().add(jPanel);
61          Grafico.pack();
62          Grafico.setVisible(true);
63          Grafico.setLocationRelativeTo(null);
64          Grafico.setDefaultCloseOperation(JFrame.
    DISPOSE_ON_CLOSE);
65
66      }
67  }
```

```java
package harbor;

import es.ull.iis.simulation.model.Experiment;
import es.ull.iis.simulation.model.Simulation;
import infoReceiver.HarborLocationListener;
import infoReceiver.TimeInfo;

public class HarborExperiment extends Experiment {
  private static final int NEXP = 1;
  private int nCranes;
  private int nTrucks;
  private int nYardCranes[];
  private int nContainers;
  private int endTS;
  private int startTS;

  public HarborExperiment(int endTS, int startTS, int nCranes, int nTrucks, int[] nYardCranes, int nContainers) {
    super("Experiment with locations", NEXP);
    this.endTS = endTS;
    this.startTS = startTS;
    this.nCranes = nCranes;
    this.nTrucks = nTrucks;
    this.nYardCranes = nYardCranes;
    this.nContainers = nContainers;
  }

  @Override
  public Simulation getSimulation(int ind) {
    final HarborSimulation sim =  new HarborSimulation(ind, endTS, startTS, nCranes, nTrucks, nYardCranes, nContainers);
    sim.addInfoReceiver(new HarborLocationListener(description));
    sim.addInfoReceiver(new TimeInfo(nCranes, nTrucks, nYardCranes, nContainers));
    return sim;
  }
}

```

```java
package harbor;

import es.ull.iis.function.TimeFunctionFactory;
import es.ull.iis.simulation.model.location.Location;
import es.ull.iis.simulation.model.location.Movable;
import es.ull.iis.simulation.model.location.Node;
import es.ull.iis.simulation.model.location.Path;
import es.ull.iis.simulation.model.location.Router;

class HarborRouter implements Router {
  final private Node parking;
  final private Node block1;
  final private Node block2;
  final private Node ship;
  final private Path[] paths;
  final private static int NPATHS = 5;
  private static final long DELAY_HOME = 2;
  private static final long DELAY_PATH = 5;

    /* + Longitud = + Delay
    +-------+-------+--------------+-------------------------
----+---------------+
    | barco |  p1   |     p2       |           p3
                |               |
    +-+---+-+-+---+-+---+-------------------------------------
----+               |
      |   |   |   |     |          |
    |     bloque2   |
      |p0 |   |   |     |  bloque1
  |                                +---------------+
      |   |   | p4|     +------------+
    +-+---+---+   |
    |parking|     |
    +-------------+
     */

  public HarborRouter(int nCranes, int[] nYardCranes) {
    parking = new Node("Parking", TimeFunctionFactory.
getInstance("ConstantVariate", DELAY_HOME), 50);
    ship = new Node("Barco", TimeFunctionFactory.getInstance("
ConstantVariate", DELAY_HOME), nCranes);
    block1 = new Node("Patio 1", TimeFunctionFactory.
getInstance("ConstantVariate", DELAY_HOME), nYardCranes[0]);
    block2 = new Node("Patio 2", TimeFunctionFactory.
getInstance("ConstantVariate", DELAY_HOME), nYardCranes[1]);
    paths = new Path[NPATHS];

    // Por el momento, todos los paths tienen el mismo delay.
    for (int i = 0; i < NPATHS; i++) {
      paths[i] = new Path("Path " + i, TimeFunctionFactory.
getInstance("ConstantVariate", DELAY_PATH), 10, 2);
    }
```

```
43
44      // Definición del mapa (definimos cómo están conectadas las
    cosas)
45      parking.linkTo(paths[0]);
46
47      paths[0].linkTo(ship);
48
49      ship.linkTo(paths[1]);
50
51      paths[1].linkTo(ship);
52      paths[1].linkTo(paths[2]);
53      paths[1].linkTo(paths[4]);
54
55      paths[2].linkTo(block1);
56      paths[2].linkTo(paths[1]);
57      paths[2].linkTo(paths[3]);
58
59      paths[3].linkTo(block2);
60      paths[3].linkTo(paths[2]);
61
62      paths[4].linkTo(parking);
63    }
64
65
66    /**
67     * @return the parking
68     */
69    public Node getHome() {
70      return parking;
71    }
72
73    /**
74     * @return the destination ship
75     */
76    public Node getDestinationShip() {
77      return ship;
78    }
79
80    /**
81     * @return the first yard/block.
82     */
83    public Node getDestinationB1() {
84      return block1;
85    }
86
87    /**
88     * @return the second yard/block.
89     */
90    public Node getDestinationB2() {
91      return block2;
92    }
93
```

```
 94   /**
 95    * @return the first yard.
 96    */
 97
 98   // Dada la localización actual y el punto final, definimos,
    cuál es el
 99   // siguiente movimiento a realizar.
100   @Override
101   public Location getNextLocationTo(Movable entity, Location
    finalLocation) {
102     Location currentLocation = entity.getLocation(); //
    Posición actual del camión.
103
104     if (ship.equals(finalLocation)) {
105       if (currentLocation.equals(block2))
106         return paths[3];
107       if (currentLocation.equals(paths[3]))
108         return paths[2];
109       if (currentLocation.equals(block1))
110         return paths[2];
111       if (currentLocation.equals(paths[2]))
112         return paths[1];
113       if (currentLocation.equals(paths[1]))
114         return ship;
115       if (currentLocation.equals(paths[0]))
116         return ship;
117       if (currentLocation.equals(parking))
118         return paths[0];
119
120     } else if (block1.equals(finalLocation)) {
121       if (currentLocation.equals(block2))
122         return paths[3];
123       if (currentLocation.equals(paths[3]))
124         return paths[2];
125       if (currentLocation.equals(paths[2]))
126         return block1;
127       if (currentLocation.equals(paths[1]))
128         return paths[2];
129       if (currentLocation.equals(ship))
130         return paths[1];
131       if (currentLocation.equals(paths[0]))
132         return ship;
133       if (currentLocation.equals(parking))
134         return paths[0];
135
136     } else if (block2.equals(finalLocation)) {
137       if (currentLocation.equals(paths[3]))
138         return block2;
139       if (currentLocation.equals(paths[2]))
140         return paths[3];
141       if (currentLocation.equals(block1))
142         return paths[2];
```

```
143         if (currentLocation.equals(paths[1]))
144           return paths[2];
145         if (currentLocation.equals(ship))
146           return paths[1];
147         if (currentLocation.equals(paths[0]))
148           return ship;
149         if (currentLocation.equals(parking))
150           return paths[0];
151
152       } else if (parking.equals(finalLocation)) {
153         if (currentLocation.equals(block2))
154           return paths[3];
155         if (currentLocation.equals(paths[3]))
156           return paths[2];
157         if (currentLocation.equals(block1))
158           return paths[2];
159         if (currentLocation.equals(paths[2]))
160           return paths[1];
161         if (currentLocation.equals(paths[1]))
162           return paths[4];
163         if (currentLocation.equals(paths[4]))
164           return parking;
165       }
166
167       return null;
168     }
169
170 }
```

```java
package harbor;

import es.ull.iis.simulation.model.ElementType;
import es.ull.iis.simulation.model.Resource;
import es.ull.iis.simulation.model.ResourceType;
import es.ull.iis.simulation.model.Simulation;
import es.ull.iis.simulation.model.SimulationPeriodicCycle;
import es.ull.iis.simulation.model.SimulationTimeFunction;
import es.ull.iis.simulation.model.TimeDrivenElementGenerator;
import es.ull.iis.simulation.model.TimeUnit;
import es.ull.iis.simulation.model.WorkGroup;
import es.ull.iis.simulation.model.flow.ActivityFlow;
import es.ull.iis.simulation.model.flow.
ProbabilitySelectionFlow;
import es.ull.iis.simulation.model.flow.ReleaseResourcesFlow;
import es.ull.iis.simulation.model.flow.RequestResourcesFlow;
import es.ull.iis.simulation.model.location.MoveResourcesFlow;

public class HarborSimulation extends Simulation {

  // Resources descriptions
  public static final String CONTAINER = "Container.";
  public static final String TRUCK = "Delivery Truck.";
  public static final String SHIP_CRANE = "Ship crane.";
  public static final String YARD_CRANE = "Yard crane ";

  // Activities
  public static final String RELEASE_CONTAINER_AT_BLOCK = "
Release container at Block ";

  // Movement flows
  public static final String TRUCK_TO_SHIP = "Move truck to
ship.";
  public static final String TRUCK_TO_B = "Move truck to Block
 ";

  // Activities
  public static final String PICK_CONTAINER_FROM_SHIP = "Pick
container from the ship.";
  public static final String REQ_CONTAINER = "Container ready
to go.";
  public static final String RELEASE_TRUCK = "Release truck.";

  public static final Double B1_PERCENTAGE = 0.4; // Percentage
  public static final Double B2_PERCENTAGE = 0.6; // Percentage

  public static final Integer DELAY_SHIP_CRANE = 2; // Minutes
  public static final Integer DELAY_YARD_CRANE = 2; // Minutes



  public HarborSimulation(int id, long endTs, long startTs,
```

```
47                              int nShipCranes, int nTrucks, int[]
   nYardCranes, int nContainers) {
48      super(id, "Simulating locations " + id, 0, endTs);
49
50      // Create the "map"
51      final HarborRouter router = new HarborRouter(nShipCranes,
   nYardCranes);
52
53      // Define elements
54      final ElementType etContainers = new ElementType(this,
   CONTAINER);
55
56      // Define resources
57      final ResourceType rtTruck      = new ResourceType(this,
   TRUCK);
58      final ResourceType rtShipCrane  = new ResourceType(this,
   SHIP_CRANE);
59      final ResourceType rtYardCrane1 = new ResourceType(this,
   YARD_CRANE + "Block 1");
60      final ResourceType rtYardCrane2 = new ResourceType(this,
   YARD_CRANE + "Block 2");
61
62      SimulationPeriodicCycle DailyCycle = new
   SimulationPeriodicCycle(TimeUnit.MINUTE, 0, new
   SimulationTimeFunction(TimeUnit.MINUTE, "ConstantVariate", 24*
   60), 0);
63
64      // Creo e inicializo los camiones en el location Home
65      Resource resTrucks[]      = new Resource[nTrucks];
66      Resource resShipCranes[]  = new Resource[nShipCranes];
67      Resource resYardCranes1[] = new Resource[nYardCranes[0]];
68      Resource resYardCranes2[] = new Resource[nYardCranes[1]];
69
70      // Inicialización de los recursos
71      for(int i = 0; i < nTrucks; i++) {
72        resTrucks[i] = new Resource(this,"Truck-" + i, 1, router.
   getHome());
73        resTrucks[i].newTimeTableOrCancelEntriesAdder(rtTruck).
   withDuration(DailyCycle, 6 * 60).addTimeTableEntry();
74      }
75
76      for(int i = 0; i < nShipCranes; i++) {
77        resShipCranes[i] = new Resource(this,"ShipCranes-" + i, 1
   , router.getHome());
78        resShipCranes[i].newTimeTableOrCancelEntriesAdder(
   rtShipCrane).withDuration(DailyCycle, 6 * 60).addTimeTableEntry
   ();
79      }
80
81      for(int i = 0; i < nYardCranes[0]; i++) {
82        resYardCranes1[i] = new Resource(this,"YardCrane1-" + i,
   1, router.getHome());
```

```
83        resYardCranes1[i].newTimeTableOrCancelEntriesAdder(
   rtYardCrane1).withDuration(DailyCycle, 6 * 60).
   addTimeTableEntry();
84      }
85
86      for(int i = 0; i < nYardCranes[1]; i++) {
87        resYardCranes2[i] = new Resource(this,"YardCrane2-" + i
   , 1, router.getHome());
88        resYardCranes2[i].newTimeTableOrCancelEntriesAdder(
   rtYardCrane2).withDuration(DailyCycle, 6 * 60).
   addTimeTableEntry();
89      }
90
91      // Asignación de los WorkGroup
92      final WorkGroup wgTruck = new WorkGroup(this, rtTruck, 1);
93      final WorkGroup wgShipCrane = new WorkGroup(this, new
   ResourceType[] { rtShipCrane }, new int[] { 1 });
94      final WorkGroup wgYardCranes1 = new WorkGroup(this, new
   ResourceType[] { rtYardCrane1 }, new int[] { 1 });
95      final WorkGroup wgYardCranes2 = new WorkGroup(this, new
   ResourceType[] { rtYardCrane2 }, new int[] { 1 });
96
97      // Definición de Actividades y movimientos.
98      final RequestResourcesFlow reqFlow = new
   RequestResourcesFlow(this, REQ_CONTAINER);
99      reqFlow.newWorkGroupAdder(wgTruck).add(); // Hace falta
   camiones disponibles para requerir camiones.
100
101     ActivityFlow actTruckShip = new ActivityFlow(this,
   PICK_CONTAINER_FROM_SHIP);
102     ActivityFlow actTruckB1 = new ActivityFlow(this,
   RELEASE_CONTAINER_AT_BLOCK + "1");
103     ActivityFlow actTruckB2 = new ActivityFlow(this,
   RELEASE_CONTAINER_AT_BLOCK + "2");
104
105     actTruckShip.newWorkGroupAdder(wgShipCrane).withDelay(
   DELAY_SHIP_CRANE).add();
106     actTruckB1.newWorkGroupAdder(wgYardCranes1).withDelay(
   DELAY_YARD_CRANE).add();
107     actTruckB2.newWorkGroupAdder(wgYardCranes2).withDelay(
   DELAY_YARD_CRANE).add();
108
109     //final MoveResourcesFlow moveToParking = new
   MoveResourcesFlow(this, TRUCK_TO_PARKING, router.getHome(),
   router, wgTruck);
110     final MoveResourcesFlow moveToShip = new MoveResourcesFlow
   (this, TRUCK_TO_SHIP, router.getDestinationShip(), router,
   wgTruck);
111     final MoveResourcesFlow moveToB1 = new MoveResourcesFlow(
   this, TRUCK_TO_B + "1", router.getDestinationB1(), router,
   wgTruck);
112     final MoveResourcesFlow moveToB2 = new MoveResourcesFlow(
```

```
112 this, TRUCK_TO_B + "2", router.getDestinationB2(), router,
    wgTruck);
113
114     final ReleaseResourcesFlow relFlow = new
    ReleaseResourcesFlow(this, RELEASE_TRUCK, wgTruck);
115
116     /* Se define un flow probabilístico.
117      */
118     final ProbabilitySelectionFlow selectBlockFlow = new
    ProbabilitySelectionFlow(this);
119     // Por ejemplo, suponemos que un 40% irán al bloque 1 y un
     60% al bloque 2
120     selectBlockFlow.link(moveToB1, B1_PERCENTAGE);
121     selectBlockFlow.link(moveToB2, B2_PERCENTAGE);
122
123     // Indicamos el flujo de movimientos que realiza un barco
    desde que
124     // se pide hasta que se libera.
125     reqFlow.link(moveToShip);
126     moveToShip.link(actTruckShip);
127     actTruckShip.link(selectBlockFlow);
128     moveToB1.link(actTruckB1);
129     moveToB2.link(actTruckB2);
130     actTruckB1.link(relFlow);
131     actTruckB2.link(relFlow);
132
133     new TimeDrivenElementGenerator(this, nContainers,
    etContainers, reqFlow, new SimulationPeriodicCycle(getTimeUnit
    (), 0L, new SimulationTimeFunction(getTimeUnit(), "
    ConstantVariate", getEndTs()), 365));
134   }
135
136 }
137
```

```java
package infoReceiver;


import es.ull.iis.simulation.info.*;
import es.ull.iis.simulation.inforeceiver.Listener;
import es.ull.iis.simulation.model.Resource;
import gui.ResultsTable;
import harbor.HarborSimulation;

import java.util.*;

public class HarborLocationListener extends Listener {
    //guarda un registro de todos los eventos de tipo incio/final
    private List<Register> log;
    //un mapa para cada elemento, guardamos una lista de los tiempos en los que ha cambiado de estado
    private Map<String, List<Long>> data;
    //para calcular el tiempo empleado
    private long end;
    private boolean finished = false;

    TreeMap<Integer, Long> ContainerIni_BC = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_BC = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CI = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CI = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CP = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CP = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CV = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CV = new TreeMap<Integer, Long>();
    long tMaximo = 0L;


    public HarborLocationListener(String description) {
        super("Listener");
        addEntrance(EntityLocationInfo.class);
        addEntrance(ElementActionInfo.class);
        addEntrance(ResourceUsageInfo.class);
        addEntrance(SimulationStartStopInfo.class);
        log = new LinkedList();
        data = new HashMap();
        end = -1;
    }

```

```
43      @Override
44      public void infoEmited(SimulationInfo info) {
45
46          System.out.println(info);
47
48          if (!finished)
49              //si es de tipo acción de elemento, casteamos el
    objeto y extraemos los datos básicos
50              if (info instanceof ElementActionInfo) {
51                  final ElementActionInfo eInfo = (
    ElementActionInfo) info;
52                  final int containerId = eInfo.getElement().
    getIdentifier();
53                  String name = eInfo.getElement().toString();
54                  switch (eInfo.getType()) {//en función del tipo
     de evento lo registramos de una u otra forma
55
56                      case START:
57
58                          //añadimos el registro al log
59                          log.add(new Register(eInfo.getElement
    ().toString(), "comienzo", eInfo.getActivity().getDescription
    (), eInfo.getTs(), eInfo.getResources(), eInfo.getTs()-360));
60                          //si es la primera aparición del
    elemento, creamos la lista donde guardaremos la secuencia de
    tiempos
61                          if(!data.containsKey(name)) {
62                              data.put(name, new LinkedList());
63                          }
64                          //añadimos el tiempo actual al elemento
     actual
65                          data.get(name).add(eInfo.getTs());
66
67                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)){
68                              ContainerIni_BC.put(containerId,
    eInfo.getTs());
69                          }
70                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_SHIP)){
71                              ContainerIni_CI.put(containerId,
    eInfo.getTs());
72                          }
73                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.RELEASE_CONTAINER_AT_BLOCK)){
74                              ContainerIni_CP.put(containerId,
    eInfo.getTs());
75                          }
76                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_B)){
77                              ContainerIni_CV.put(containerId,
    eInfo.getTs());
```

```
78                          }
79
80                          break;
81
82                      case END:
83
84                          //guardamos el log del evento actual
85                          log.add(new Register(eInfo.getElement
    ().toString(), "finalización", eInfo.getActivity().
    getDescription(), eInfo.getTs(), eInfo.getResources(),eInfo.
    getTs()-360));
86                          //actualizamos el fin por si es el
    último evento del experimento
87                          end = Math.max(end, eInfo.getTs());
88                          data.get(name).add(eInfo.getTs());
89
90                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)){
91                              ContainerFin_BC.put(containerId,
    eInfo.getTs());
92                          }
93                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_SHIP)){
94                              ContainerFin_CI.put(containerId,
    eInfo.getTs());
95                          }
96                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.RELEASE_CONTAINER_AT_BLOCK)){
97                              ContainerFin_CP.put(containerId,
    eInfo.getTs());
98                          }
99                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_B)){
100                             ContainerFin_CV.put(containerId,
    eInfo.getTs());
101                         }
102
103                         break;
104                 }
105                 //si es un elemento de tipo info, si es de
    finalización verificamos que su cota no es superior a la
    almacenada
106             } else if (info instanceof ElementInfo) {
107                 ElementInfo eInfo = (ElementInfo) info;
108                 switch (eInfo.getType()) {
109                     case FINISH:
110                         end = Math.max(end, eInfo.getTs());
111
112                         break;
113
114                 }
115
```

```
116             } else if (info instanceof SimulationStartStopInfo
    ) {
117 
118                 // Al final de la simulacion, crea la ventana
    de resultados.
119                 if (SimulationStartStopInfo.Type.END.equals(((
    SimulationStartStopInfo) info).getType())) {
120                     for ( Integer containerId :
    ContainerIni_BC.keySet()) {
121                         if (ContainerFin_BC.containsKey(
    containerId)) {
122                             System.out.println(containerId + "
    \t\t" + ContainerIni_BC.get(containerId) + "\t\t" +
    ContainerFin_BC.get(containerId) +
123                                     "\t\t" + ContainerIni_CI.
    get(containerId) + "\t\t" + ContainerFin_CI.get(containerId) +
124                                     "\t\t" + ContainerIni_CP.
    get(containerId) + "\t\t" + ContainerFin_CP.get(containerId));
125                         }
126                     }
127 
128                     finished = true;
129                     new ResultsTable(this).setVisible(true);
130 
131                 }
132                 else if (SimulationStartStopInfo.Type.START.
    equals(((SimulationStartStopInfo) info).getType())) {
133                     System.out.println("Container\t
    TiempoIni_BC\tTiempoFinBC\tTiempoIni_CI\tTiempoFinCI\t
    TiempoIni_CP\tTiempoFinCP\t");
134                 }
135             }
136     }
137 
138 
139 
140     //proporciona el parámetro y del objeto de simulación x
141     public String get(int x, int y) {
142         return log.get(x).get(y);
143     }
144 
145     //devuelve el nombre de la columna en la posición i
146     public String cabecera(int i) {
147         return Register.cabecera(i);
148     }
149 
150     public long getFin() {
151         return end;
152     }
153 
154     //devuelve cuantas columnas posee los datos
155     public int columnas() {
```

```
156         return Register.columnas();
157     }
158     // devuelve cuantas filas se han obtenido
159     public int filas() {
160         return log.size();
161     }
162 
163     //devuelve el tiempo empleado
164     public String getTiempo() {
165         return "Tiempo empleado: " + end;
166     }
167 
168     //devuelve el conjunto de todas las ids implicadas.
    Obtener una lista de un mapa no es trivial
169     //hay que convertirlo en array y ordenarlo
170     public String[] getIds() {
171         String [] a = new String[1];
172         a= data.keySet().toArray(a);
173         Arrays.sort(a);
174         return a;
175     }
176 
177     //obtiene los tiempos de cambio de estado del elemento
    indicado
178     public List<Long> getTiempos(String name){
179         return data.get(name);
180     }
181 
182     public int tamMapa(){
183         return data.size();
184     }
185 
186     public Map<String, List<Long>> getMapa(){
187         return data;
188     }
189 
190 }
191 
192 /*
193  * Clase interna que modeliza una fila de datos para la tabla.
194  */
195 class Register {
196     //si quisieramos guardar mas información, sería un campo
    extra pasado en el constructor
197     private String evento;
198     private String id;
199     private String tiempo;
200     private String location;
201     private String types;
202     private String tiempoEmpleado;
203 
204     public Register(String id, String evento, String location
```

```
204 , long tiempo, ArrayDeque<Resource> types, double
    tiempoEmpleado) {
205         super();
206         this.evento = evento;
207         this.id = "" + id;
208         this.tiempo = "" + tiempo;
209         this.location = "";
210         this.tiempoEmpleado = "" + tiempoEmpleado;
211         if (location != null)
212             this.location = location;
213         if (types != null) {
214             for(Resource recurso : types) {
215                 this.types += (recurso.getCurrentResourceType
    ().getDescription()) + ", ";
216             }
217         }
218     }
219 
220     //tantas columnas como atributos hemos recogido en el
    constructor
221     public static int columnas() {
222         return 4;
223     }
224 
225     //devuelve cada una de las cabeceras de la tabla en orden
226     public static String cabecera(int id) {
227         switch (id) {
228             case 0:
229                 return "id";
230             case 1:
231                 return "evento";
232             case 2:
233                 return "localización";
234             case 3:
235                 return "tiempo";
236             case 4:
237                 return "tipos";
238             case 5:
239                 return "tiempo empleado";
240             default:
241                 return "";
242         }
243     }
244 
245     //devuelve el valor almacenado para cada número de columna
246     public String get(int campo) {
247         switch (campo) {
248             case 0:
249                 return id;
250             case 1:
251                 return evento;
252             case 2:
```

```
253                 return location;
254             case 3:
255                 return tiempo;
256             case 4:
257                 return types;
258             case 5:
259                 return tiempoEmpleado;
260             default:
261                 return "";
262         }
263     }
264 
265 
266     //CLASE TIEMPOS FINALES
267 
268 
269 }
270 
```

```java
package infoReceiver;


import es.ull.iis.simulation.info.*;
import es.ull.iis.simulation.inforeceiver.Listener;
import es.ull.iis.simulation.model.Resource;
import gui.ResultsTable;
import harbor.HarborSimulation;

import java.util.*;

public class HarborLocationListener extends Listener {
    //guarda un registro de todos los eventos de tipo incio/final
    private List<Register> log;
    //un mapa para cada elemento, guardamos una lista de los tiempos en los que ha cambiado de estado
    private Map<String, List<Long>> data;
    //para calcular el tiempo empleado
    private long end;
    private boolean finished = false;

    TreeMap<Integer, Long> ContainerIni_BC = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_BC = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CI = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CI = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CP = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CP = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerIni_CV = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CV = new TreeMap<Integer, Long>();
    long tMaximo = 0L;


    public HarborLocationListener(String description) {
        super("Listener");
        addEntrance(EntityLocationInfo.class);
        addEntrance(ElementActionInfo.class);
        addEntrance(ResourceUsageInfo.class);
        addEntrance(SimulationStartStopInfo.class);
        log = new LinkedList();
        data = new HashMap();
        end = -1;
    }

```

```
43      @Override
44      public void infoEmited(SimulationInfo info) {
45
46          System.out.println(info);
47
48          if (!finished)
49              //si es de tipo acción de elemento, casteamos el
    objeto y extraemos los datos básicos
50              if (info instanceof ElementActionInfo) {
51                  final ElementActionInfo eInfo = (
    ElementActionInfo) info;
52                  final int containerId = eInfo.getElement().
    getIdentifier();
53                  String name = eInfo.getElement().toString();
54                  switch (eInfo.getType()) {//en función del tipo
     de evento lo registramos de una u otra forma
55
56                      case START:
57
58                          //añadimos el registro al log
59                          log.add(new Register(eInfo.getElement
    ().toString(), "comienzo", eInfo.getActivity().getDescription
    (), eInfo.getTs(), eInfo.getResources(), eInfo.getTs()-360));
60                          //si es la primera aparición del
    elemento, creamos la lista donde guardaremos la secuencia de
    tiempos
61                          if(!data.containsKey(name)) {
62                              data.put(name, new LinkedList());
63                          }
64                          //añadimos el tiempo actual al elemento
     actual
65                          data.get(name).add(eInfo.getTs());
66
67                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)){
68                              ContainerIni_BC.put(containerId,
    eInfo.getTs());
69                          }
70                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_SHIP)){
71                              ContainerIni_CI.put(containerId,
    eInfo.getTs());
72                          }
73                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.RELEASE_CONTAINER_AT_BLOCK)){
74                              ContainerIni_CP.put(containerId,
    eInfo.getTs());
75                          }
76                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_B)){
77                              ContainerIni_CV.put(containerId,
    eInfo.getTs());
```

```
78                          }
79
80                          break;
81
82                      case END:
83
84                          //guardamos el log del evento actual
85                          log.add(new Register(eInfo.getElement
    ().toString(), "finalización", eInfo.getActivity().
    getDescription(), eInfo.getTs(), eInfo.getResources(),eInfo.
    getTs()-360));
86                          //actualizamos el fin por si es el
    último evento del experimento
87                          end = Math.max(end, eInfo.getTs());
88                          data.get(name).add(eInfo.getTs());
89
90                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)){
91                              ContainerFin_BC.put(containerId,
    eInfo.getTs());
92                          }
93                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_SHIP)){
94                              ContainerFin_CI.put(containerId,
    eInfo.getTs());
95                          }
96                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.RELEASE_CONTAINER_AT_BLOCK)){
97                              ContainerFin_CP.put(containerId,
    eInfo.getTs());
98                          }
99                          if (eInfo.getActivity().getDescription
    ().contains(HarborSimulation.TRUCK_TO_B)){
100                             ContainerFin_CV.put(containerId,
    eInfo.getTs());
101                         }
102
103                         break;
104                 }
105                 //si es un elemento de tipo info, si es de
    finalización verificamos que su cota no es superior a la
    almacenada
106             } else if (info instanceof ElementInfo) {
107                 ElementInfo eInfo = (ElementInfo) info;
108                 switch (eInfo.getType()) {
109                     case FINISH:
110                         end = Math.max(end, eInfo.getTs());
111
112                         break;
113
114                 }
115
```

```
116             } else if (info instanceof SimulationStartStopInfo
    ) {
117 
118                 // Al final de la simulacion, crea la ventana
    de resultados.
119                 if (SimulationStartStopInfo.Type.END.equals(((
    SimulationStartStopInfo) info).getType())) {
120                     for ( Integer containerId :
    ContainerIni_BC.keySet()) {
121                         if (ContainerFin_BC.containsKey(
    containerId)) {
122                             System.out.println(containerId + "
    \t\t" + ContainerIni_BC.get(containerId) + "\t\t" +
    ContainerFin_BC.get(containerId) +
123                                     "\t\t" + ContainerIni_CI.
    get(containerId) + "\t\t" + ContainerFin_CI.get(containerId) +
124                                     "\t\t" + ContainerIni_CP.
    get(containerId) + "\t\t" + ContainerFin_CP.get(containerId));
125                         }
126                     }
127 
128                     finished = true;
129                     new ResultsTable(this).setVisible(true);
130 
131                 }
132                 else if (SimulationStartStopInfo.Type.START.
    equals(((SimulationStartStopInfo) info).getType())) {
133                     System.out.println("Container\t
    TiempoIni_BC\tTiempoFinBC\tTiempoIni_CI\tTiempoFinCI\t
    TiempoIni_CP\tTiempoFinCP\t");
134                 }
135             }
136     }
137 
138 
139 
140     //proporciona el parámetro y del objeto de simulación x
141     public String get(int x, int y) {
142         return log.get(x).get(y);
143     }
144 
145     //devuelve el nombre de la columna en la posición i
146     public String cabecera(int i) {
147         return Register.cabecera(i);
148     }
149 
150     public long getFin() {
151         return end;
152     }
153 
154     //devuelve cuantas columnas posee los datos
155     public int columnas() {
```

```
156         return Register.columnas();
157     }
158     // devuelve cuantas filas se han obtenido
159     public int filas() {
160         return log.size();
161     }
162 
163     //devuelve el tiempo empleado
164     public String getTiempo() {
165         return "Tiempo empleado: " + end;
166     }
167 
168     //devuelve el conjunto de todas las ids implicadas.
    Obtener una lista de un mapa no es trivial
169     //hay que convertirlo en array y ordenarlo
170     public String[] getIds() {
171         String [] a = new String[1];
172         a= data.keySet().toArray(a);
173         Arrays.sort(a);
174         return a;
175     }
176 
177     //obtiene los tiempos de cambio de estado del elemento
    indicado
178     public List<Long> getTiempos(String name){
179         return data.get(name);
180     }
181 
182     public int tamMapa(){
183         return data.size();
184     }
185 
186     public Map<String, List<Long>> getMapa(){
187         return data;
188     }
189 
190 }
191 
192 /*
193  * Clase interna que modeliza una fila de datos para la tabla.
194  */
195 class Register {
196     //si quisieramos guardar mas información, sería un campo
    extra pasado en el constructor
197     private String evento;
198     private String id;
199     private String tiempo;
200     private String location;
201     private String types;
202     private String tiempoEmpleado;
203 
204     public Register(String id, String evento, String location
```

```
204 , long tiempo, ArrayDeque<Resource> types, double
    tiempoEmpleado) {
205         super();
206         this.evento = evento;
207         this.id = "" + id;
208         this.tiempo = "" + tiempo;
209         this.location = "";
210         this.tiempoEmpleado = "" + tiempoEmpleado;
211         if (location != null)
212             this.location = location;
213         if (types != null) {
214             for(Resource recurso : types) {
215                 this.types += (recurso.getCurrentResourceType
    ().getDescription()) + ", ";
216             }
217         }
218     }
219 
220     //tantas columnas como atributos hemos recogido en el
    constructor
221     public static int columnas() {
222         return 4;
223     }
224 
225     //devuelve cada una de las cabeceras de la tabla en orden
226     public static String cabecera(int id) {
227         switch (id) {
228             case 0:
229                 return "id";
230             case 1:
231                 return "evento";
232             case 2:
233                 return "localización";
234             case 3:
235                 return "tiempo";
236             case 4:
237                 return "tipos";
238             case 5:
239                 return "tiempo empleado";
240             default:
241                 return "";
242         }
243     }
244 
245     //devuelve el valor almacenado para cada número de columna
246     public String get(int campo) {
247         switch (campo) {
248             case 0:
249                 return id;
250             case 1:
251                 return evento;
252             case 2:
```

```
253                 return location;
254             case 3:
255                 return tiempo;
256             case 4:
257                 return types;
258             case 5:
259                 return tiempoEmpleado;
260             default:
261                 return "";
262         }
263     }
264 
265 
266     //CLASE TIEMPOS FINALES
267 
268 
269 }
270 
```

```java
package infoReceiver;

import es.ull.iis.simulation.info.*;
import es.ull.iis.simulation.inforeceiver.Listener;
import es.ull.iis.simulation.model.Element;
import es.ull.iis.simulation.model.Resource;
import gui.timePercentageGraph;
import harbor.HarborSimulation;

import java.util.TreeMap;

public class TimeInfo extends Listener {
    TreeMap<Element, Long> initialTimes = new TreeMap<Element, Long>();
    TreeMap<Element, Long> endTimes = new TreeMap<Element, Long>();
    TreeMap<Integer, Long> ContainerIni_BC = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CI = new TreeMap<Integer, Long>();
    TreeMap<Integer, Long> ContainerFin_CP = new TreeMap<Integer, Long>();

    TreeMap<Integer, Long> time = new TreeMap<Integer, Long>(); // Tiempos por contenedor
    TreeMap<Resource, Long> lastTruck = new TreeMap<Resource, Long>();
    TreeMap<Resource, Long> truckAccumulated = new TreeMap<Resource, Long>(); // tiempoOciosoCamiones
    TreeMap<Resource, Long> lastCrane = new TreeMap<Resource, Long>();
    TreeMap<Resource, Long> craneAccumulated = new TreeMap<Resource, Long>(); // tiempoOciosoGruasBarco
    TreeMap<Resource, Long> lastYardCrane = new TreeMap<Resource, Long>();
    TreeMap<Resource, Long> yardCraneAccumulated = new TreeMap<Resource, Long>(); // tiempoOciosoGruasPatio

    Long resta = 0L;
    long tMaximo = 0L;
    private boolean finished = false;

    private int nCranes;
    private int nTrucks;
    private int [] nYardCranes;
    private int nContainers;

    public TimeInfo(int nCranes, int nTrucks, int [] nYardCranes, int nContainers) {
        // Listener que dice cuando se acaba de colocar todos los containers
        super("Tiempo que se tarda en realizar el ciclo
```

```
38 completo y tengo el tiemmpo que tarda cada contenedor");
39         addEntrance(ElementInfo.class);
40         addEntrance(ElementActionInfo.class);
41         addEntrance(SimulationStartStopInfo.class);
42         addEntrance(ResourceUsageInfo.class);
43         this.nCranes = nCranes;
44         this.nTrucks = nTrucks;
45         this.nYardCranes = nYardCranes;
46         this.nContainers = nContainers;
47     }
48 
49     @Override
50     public void infoEmited(SimulationInfo info) {
51         // ara conocer el tiempo que cada contenedor se pasa
   del tiempo
52         if (info instanceof ElementActionInfo) {
53             final ElementActionInfo eInfo = (ElementActionInfo
   ) info;
54             final int containerId = eInfo.getElement().
   getIdentifier();
55             switch (eInfo.getType()) {
56                 case REQ:
57                     break;
58                 case END:
59                     if (eInfo.getActivity().getDescription().
   contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)) {
60                         ContainerFin_CI.put(containerId, eInfo.
   getTs());
61                     }
62                     if (eInfo.getActivity().getDescription().
   contains(HarborSimulation.TRUCK_TO_B)) {
63                         ContainerFin_CP.put(containerId, eInfo.
   getTs());
64 
65                         resta = ContainerFin_CP.get(containerId
   ) - ContainerIni_BC.get(containerId);
66                         time.put(containerId, resta);
67                     }
68                     break;
69                 case START:
70                     if (eInfo.getActivity().getDescription().
   contains(HarborSimulation.PICK_CONTAINER_FROM_SHIP)) {
71                         ContainerIni_BC.put(containerId, eInfo.
   getTs());
72                     }
73                     break;
74                 case INTACT:
75                     break;
76                 case RESACT:
77                     break;
78                 default:
79                     break;
```

```
80              }
81          }
82          // listener para conocer el tiempoo de los recursos
83          if (info instanceof ResourceUsageInfo) {
84              final ResourceUsageInfo eInfo = (ResourceUsageInfo
    ) info;
85              final Resource recurso = eInfo.getResource();
86              switch (eInfo.getType()) {
87                  case CAUGHT:
88                      if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.TRUCK)) {
89                          lastTruck.put(recurso, eInfo.getTs());
90                      }
91                      if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.SHIP_CRANE)) {
92                          lastCrane.put(recurso, eInfo.getTs());
93                      }
94                      if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.YARD_CRANE)) {
95                          lastYardCrane.put(recurso, eInfo.getTs
    ());
96                      }
97                      break;
98                  case RELEASED:
99                      if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.TRUCK)) {
100                         long acumulado = 0;
101                         if (truckAccumulated.containsKey(
    recurso)) {
102                             acumulado = truckAccumulated.get(
    recurso);
103                         }
104                         truckAccumulated.put(recurso,
    acumulado + eInfo.getTs() - lastTruck.get(recurso));
105                     }
106                     if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.SHIP_CRANE)) {
107                         long acumulado = 0;
108                         if (craneAccumulated.containsKey(
    recurso)) {
109                             acumulado = craneAccumulated.get(
    recurso);
110                         }
111                         craneAccumulated.put(recurso,
    acumulado + eInfo.getTs() - lastCrane.get(recurso));
112                     }
113                     if (eInfo.getResourceType().getDescription
    ().contains(HarborSimulation.YARD_CRANE)) {
114                         long acumulado = 0;
115                         if (yardCraneAccumulated.containsKey(
    recurso)) {
116                             acumulado = yardCraneAccumulated.
```

```
116 get(recurso);
117                         }
118                         yardCraneAccumulated.put(recurso,
    acumulado + eInfo.getTs() - lastYardCrane.get(recurso));
119                     }
120                     break;
121             }
122         }
123         // Listener que calcula el tiempo que ha durado la
    simulacion en terminar
124         else if (info instanceof ElementInfo) {
125             ElementInfo eInfo = (ElementInfo) info;
126             switch (eInfo.getType()) {
127                 case FINISH:
128                     endTimes.put(eInfo.getElement(), eInfo.
    getTs() );
129                     if (tMaximo < eInfo.getTs()) {
130                         tMaximo = eInfo.getTs() ;
131                     }
132                     break;
133                 case START:
134                     initialTimes.put(eInfo.getElement(), eInfo
    .getTs() );
135                     break;
136                 default:
137                     break;
138             }
139 
140         } else if (info instanceof SimulationStartStopInfo) {
141 
142             // Cuando acaba se crea la ventana de resultados.
143             if (SimulationStartStopInfo.Type.END.equals(((
    SimulationStartStopInfo) info).getType())) {
144 
145                 finished = true;
146 
147                timePercentageGraph graf = new
    timePercentageGraph(truckAccumulated, craneAccumulated,
    yardCraneAccumulated, tMaximo);
148 
149 
150             }
151         }
152 
153     }
154 
155 }
156 
```