



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

1

2

Trabajo de Fin de Grado

3

4

Simulación de las Operaciones de Gestión 5 de Contenedores en Terminales Marítimas

6

7

*Simulation of Container Management Operations in
Maritime Terminals*

8

Óscar Darias Plasencia

9

10

La Laguna, 30 de junio de 2018

11 Doña **Belén Melián Batista**, con N.I.F. 44.311.040-E Profesora Titular de Universidad
12 adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La
13 Laguna y Don **Airam Expósito-Márquez**, con N.I.F. 54.056.048-E personal investigador
14 adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de
15 La Laguna,

16 **C E R T I F I C A N**

17 Que la presente memoria titulada:

18 "*Simulación de las Operaciones de Gestión de Contenedores en Terminales Marítimas*"

19 ha sido realizada bajo su dirección por Don **Óscar Darias Plasencia**, con N.I.F. 54.108.774-
20 D.

21 Y para que así conste, en cumplimiento de la legislación vigente y a los efectos
22 oportunos firman la presente en La Laguna a 30 de junio de 2018

Agradecimientos

23

24

25 Antes que nada, a mis tutores, que me han orientado a la perfección y han
26 estado ahí para ayudarme con los principales obstáculos que he
27 encontrado. Agradecimiento especial a Christopher Expósito Izquierdo,
28 por permitir el acceso a su software de simulación para desarrollar este
29 trabajo. Gracias también a muchos de mis profesores a lo largo de la
30 carrera. Ha sido un viaje increíble donde me he enamorado más y más de
31 esta disciplina, y mis ganas de seguir aprendiendo no han hecho sino
32 aumentar. Y por supuesto, mil gracias a mis principales compañeros de
33 clase, con quienes he podido afrontar todos los problemas con algo de
34 humor, y que me han ayudado a desconectar tantísimas veces estos
35 últimos cuatro años.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

Resumen

40 *La optimización de las operaciones de gestión de contenedores en terminales marítimas es
41 sin duda muy importante para la economía de cientos de ciudades de todo el mundo. Con
42 un enorme incremento en la importancia de la mercancía contenerizada en los últimos 25
43 años, estas operaciones deben estar muy bien programadas para asegurar la eficiencia
44 suficiente.*

45 *El objetivo de este trabajo de fin de grado es diseñar y desarrollar un software Java
46 para la simulación de estas operaciones de gestión de contenedores en cualquier terminal
47 marítima. Estas simulaciones pretenden reflejar la realidad lo máximo posible, y tienen
48 en cuenta los principales parámetros que afectan a la eficiencia de estos sistemas. Cada
49 simulación ejecutada exporta sus datos a ficheros externos, con el objetivo de permitir su
50 posterior análisis para extraer útiles conclusiones acerca de qué se tiene que mejorar o
51 modificar con respecto a estas operaciones.*

Palabras clave: simulación, librería, terminal, contenedor, optimización, recursos

Abstract

54 Optimization in the container management operations of a maritime terminal is certainly
55 very important for the economy of hundreds of cities around the world. With a huge
56 increase in the importance of containerized goods in the past 25 years, these operations
57 must be extremely well programmed to ensure enough efficiency.

58 The objective of this project is to design and develop a Java software for the simulation
59 of these container management operations in any maritime terminal. These simulations
60 intend to be as close to reality as possible, and they consider the main parameters
61 that affect the efficiency of these systems. Each simulation executed exports its data to
62 external files, so that it can be analyzed to extract useful conclusions about what should
63 be improved or changed within the management of the terminal.

6 **Keywords:** *simulation, library, terminal, container, optimization, resources*

⁶⁵ Índice general

66 1. Introducción	1
67 1.1. Contexto	1
68 1.2. Objetivos	2
69 1.3. Motivación	3
70 1.4. Estructura de la memoria	3
71 2. Simulación y optimización	5
72 2.1. Conceptos	5
73 2.2. Simulación y Optimización	6
74 2.3. Estado del arte	6
75 2.4. Estado estacionario	7
76 3. Problema	9
77 3.1. Terminal marítima de contenedores	9
78 3.2. Propuesta de modelo de simulación	10
79 3.3. Paradigma de simulación	11
80 3.4. Parámetros del modelo	12
81 3.5. Objetivos de la simulación	13
82 4. Propuesta de solución	15
83 4.1. Conceptos básicos	15
84 4.2. Librería de simulación	16
85 4.3. Vega-Lite	19
86 4.4. Diagramas de simulación	20
87 5. Codificación del modelo	26
88 5.1. Consideraciones generales	26
89 5.1.1. Representación del espacio	26
90 5.1.2. Gestión del tiempo	28
91 5.2. Agentes	28
92 5.2.1. La jerarquía de clases <i>Container</i>	28
93 5.2.2. La jerarquía de clases <i>TerminalResource</i>	29
94 5.2.3. La clase <i>ContainerBlock</i>	29
95 5.3. Principales políticas	29
96 5.3.1. Asignación de contenedores a bloques	29
97 5.3.2. Retrasos	30
98 5.4. Generación de contenedores	31
99 5.4.1. Enlace marítimo	32
100 5.4.2. Frecuencia de generación	32

101	5.5. Periodos de indisponibilidad de recursos	33
102	5.5.1. Indisponibilidad planificada	33
103	5.5.2. Indisponibilidad no planificada	33
104	5.6. Prioridad de contenedores	34
105	6. Experimentación	35
106	6.1. Consideraciones previas	35
107	6.2. Diseño de la terminal	35
108	6.3. Políticas de asignación	36
109	6.4. Recursos	39
110	6.5. Análisis de probabilidad de fallo	40
111	6.6. Análisis de períodos de indisponibilidad	42
112	6.7. Análisis de prioridad de contenedores	45
113	7. Conclusiones y líneas futuras	47
114	7.1. Librería desarrollada	47
115	7.2. Posibles usos	48
116	7.3. Líneas futuras	49
117	8. Summary and Conclusions	50
118	8.1. Resulting Library	50
119	8.2. Potential Applications	51
120	8.3. Future Possibilities	52
121	9. Presupuesto	53
122	A. Uso de la librería	55
123	A.1. Simulación asistida	55
124	A.2. Simulación manual	57
125	B. Códigos destacados	58
126	B.1. Inicialización de contenedores (Container.java)	58
127	B.2. Control de atraques (Berth.java)	59
128	B.3. Disponibilidad de recursos	61
129	B.4. Políticas de asignación	63
130	Bibliografía	65

¹³¹ Índice de figuras

¹³² 3.1. Representación de una terminal marítima de contenedores	9
¹³³ 4.1. Visual de un nodo Delay	18
¹³⁴ 4.2. Interfaz de creación de diagramas de <i>AnyLogic</i>	20
¹³⁵ 4.3. Representación de los bloques utilizados	20
¹³⁶ 4.4. Diagrama de almacenamiento	21
¹³⁷ 4.5. Diagrama de recogida de contenedores	23
¹³⁸ 4.6. Diagrama de reubicación de contenedores	24
¹³⁹ 5.1. Representación interna del espacio de la terminal	27
¹⁴⁰ 6.1. Uso de las grúas con la política del bloque más cercano	38
¹⁴¹ 6.2. Uso de las grúas con la política aleatoria	38
¹⁴² 6.3. Colas con la política aleatoria	38
¹⁴³ 6.4. Colas con la política de la menor cola	38
¹⁴⁴ 6.5. Colas con la política del bloque más cercano	38
¹⁴⁵ 6.6. Colas de llegada con una disponibilidad media para 25 vehículos y 10 bloques	44
¹⁴⁶ 6.7. Tiempos de salida con una disponibilidad media para 25 vehículos y 10 bloques	44
¹⁴⁷ 6.8. Colas de llegada con una disponibilidad baja para 25 vehículos y 10 bloques	44
¹⁴⁸ 6.9. Tiempos de salida con una disponibilidad baja para 25 vehículos y 10 bloques	44
¹⁴⁹ 6.10. Tiempo de procesado según el nivel de prioridad	46

¹⁵⁰ Índice de tablas

¹⁵¹	1.1. Evolución y distribución del transporte marítimo mundial de mercancía general. Fuente: Drewry Shipping Consultans Ltd. [8]	2
¹⁵²		
¹⁵³	6.1. Resultados de la experimentación con políticas de asignación	37
¹⁵⁴	6.2. Resultados de la experimentación con el número de recursos	40
¹⁵⁵	6.3. Resultados de la experimentación con la probabilidad de fallo	41
¹⁵⁶	6.4. Resultados de la experimentación con la disponibilidad de recursos	43
¹⁵⁷	6.5. Resultados de la experimentación con prioridad de contenedores	45
¹⁵⁸	9.1. Horas de trabajo por módulo	53
¹⁵⁹	9.2. Presupuesto total	54

¹⁶⁰ Capítulo 1

¹⁶¹ Introducción

¹⁶² 1.1. Contexto

¹⁶³ Los puertos y terminales marítimas son puntos de considerable importancia para las
¹⁶⁴ economías de todo el mundo. Se trata de uno de los activos logísticos más relevantes,
¹⁶⁵ dada su participación destacada en el intercambio de bienes a todos los niveles. A través
¹⁶⁶ de ellos se realiza la entrada y salida de miles de productos de todo tipo, que en muchos
¹⁶⁷ casos pueden llegar a ser de vital necesidad. En lugares como la isla de Tenerife, el Puerto
¹⁶⁸ de Santa Cruz mueve el 60 % de los materiales que se usan en la producción de bienes
¹⁶⁹ dentro de la isla¹.

¹⁷⁰ Estadísticas de la Organización Mundial del Comercio indican que la cantidad de
¹⁷¹ mercancías comercializadas en el mundo que se mueven por vías marítimas supera el 80 %
¹⁷² de la cifra total² [8]. Los principales factores que hacen importantes los desplazamientos
¹⁷³ por mar son los siguientes:

- ¹⁷⁴ ■ La alta cobertura geográfica que ofrecen [8].
- ¹⁷⁵ ■ El incremento del tamaño de los buques debido a la búsqueda de economías de
¹⁷⁶ escala. Esto permite el transporte simultáneo de un mayor número de mercancías
¹⁷⁷ [17].
- ¹⁷⁸ ■ El alto nivel de eficiencia con el que todo esto se logra.

¹⁷⁹ Esta información eleva enormemente la importancia de los puertos, que actúan como
¹⁸⁰ nodos para el intercambio de mercancías entre mar y tierra [17]. La conectividad se
¹⁸¹ realiza principalmente a través de tres zonas en las que se dividen los puertos: la zona
¹⁸² marítima o línea de atraque, la zona terrestre para maniobras y la zona de enlace con los
¹⁸³ medios de transporte terrestres [18]. Especialmente interesante es esta última, donde
¹⁸⁴ se encuentran las superficies e instalaciones que permiten el acceso, estacionamiento y
¹⁸⁵ retirada de las mercancías operadas, tanto para importación (lo que se asociaría con la
¹⁸⁶ salida por tierra y la llegada por mar) como para exportación (salida por mar y llegada
¹⁸⁷ por tierra).

¹⁸⁸ De entre estas mercancías, aquella que se moviliza en contenedores ha cobrado
¹⁸⁹ muchísima importancia en los últimos años. El uso de contenedores, junto con el de

¹<http://www.laopinion.es/tenerife/2016/04/24/puerto-mueve-60-materiales-produccion/670726.html>

²<http://www.ciltec.com.mx/es/infraestructura-logistica/puertos-maritimos>

Tabla 1.1: Evolución y distribución del transporte marítimo mundial de mercancía general.
Fuente: Drewry Shipping Consultans Ltd. [8]

Mercancía general	Distribución %						
	1980	1985	1990	1995	2000	2004	2005
Mercancía no contenerizada	78,2	68,5	63,6	53,1	42,5	27,8	21,8
Mercancía contenerizada	21,8	31,5	36,4	46,9	57,5	72,7	79,2

190 equipamiento especial para su manipulación, hace que los procesos de carga y descarga
 191 de un buque sean mucho más rápidos. Como resultado, esta evolución ha llegado incluso
 192 a modificar el aspecto físico de los puertos, que se adaptan para la movilización de
 193 contenedores [17]. En la tabla 1.1 se visualiza el auge de la mercancía contenerizada
 194 entre 1980 y 2005.

195 **1.2. Objetivos**

196 Cuando se tiene un flujo constante de material entrante y saliente del que depende
 197 una actividad económica, la gestión de los tiempos y la optimización de las tareas a
 198 realizar por parte de la maquinaria y personal pertenecientes al puerto se traducen en
 199 un beneficio económico directo, tanto para el propio puerto como para los terceros que
 200 hacen uso del mismo. Una técnica muy común para lograr esto se basa en realizar una
 201 simulación del puerto.

202 En concreto, en este trabajo es de especial interés la zona de enlace de los puertos. A
 203 grandes rasgos, una simulación debería contemplar, al menos, los factores más influyentes
 204 en el desarrollo de las actividades del puerto, con el objetivo de representarlos, medir
 205 su impacto en diferentes escenarios, e ir modificando sus parámetros para dar con
 206 la combinación más adecuada. Se parte de la idea de que un determinado número
 207 de contenedores entran, se almacenan y salen del puerto cada cierto tiempo. Estos
 208 contenedores son gestionados en el interior del puerto por maquinaria y personal propios,
 209 concretamente vehículos automáticos guiados y grúas, y ocuparían un determinado
 210 espacio de almacenamiento durante un tiempo. La descripción completa del problema se
 211 aborda en el capítulo 3.

212 Existen numerosas herramientas de simulación comerciales que permitirían diseñar
 213 un modelo completo del puerto. Algunas, como AnyLogic³, la cual es una de las más
 214 conocidas a nivel mundial, permiten incluso la recreación virtual en 3D de la totalidad
 215 del puerto. Estas soluciones, sin embargo, tienen un alto coste y podrían no ofrecer
 216 demasiadas ventajas con respecto a la solución que se propone en este documento.

217 En base a lo anteriormente descrito, se considera que el objetivo de este Trabajo de Fin
 218 de Grado es elaborar un software para la simulación del funcionamiento de una terminal
 219 marítima de contenedores genérica, tomando como referencia la terminal de Santa Cruz
 220 de Tenerife, para estudiar sus características y encontrar una forma eficiente de gestionar
 221 sus operaciones de almacenamiento y distribución de contenedores. En lugar de utilizar
 222 herramientas de simulación comerciales, se hará uso de una librería escrita en el lenguaje
 223 de programación Java y desarrollada por Christopher Expósito Izquierdo. Esta librería
 224 permite recrear entornos muy similares a los que genera AnyLogic, teniendo sus mismos

³<https://www.anylogic.com>

225 elementos y la misma fundamentación teórica de trasfondo, pero hasta su versión actual,
 226 sin interfaz gráfica. Esto quiere decir, a grandes rasgos, que se genera un proyecto de
 227 desarrollo para directamente programar la simulación, también usando el lenguaje Java.

228 Por tanto, el objetivo es que, al finalizar el trabajo, se haya creado un modelo de
 229 simulación software, basado en la librería citada previamente, que permita realizar
 230 simulaciones con diferentes parámetros de entrada. En la sección 7.1, se especifica cómo
 231 hacer uso de la misma, y en la sección 4.1 se explican brevemente todos estos conceptos
 232 como librería o Java, para posibles lectores no experimentados en programación.

233 **1.3. Motivación**

234 A día de hoy, el uso de aplicaciones software en optimización se ha extendido enor-
 235 memente. Ciertos programas se ocupan de la totalidad de las tareas de optimización de
 236 muchas empresas. Esto, sin embargo, no se aplica a las operaciones del puerto (al menos
 237 en Santa Cruz de Tenerife), donde la ayuda por parte de herramientas de software solo
 238 es parcial, siendo todavía muy importante la elaboración de esquemas en papel. Esta
 239 situación comprende parte de la motivación a la hora de seleccionar este tema para este
 240 trabajo de fin de grado: la posibilidad futura de su aplicación a situaciones reales.

241 Además, teniendo en cuenta la naturaleza de estas operaciones, tiene sentido aprove-
 242 char la optimización matemática y la simulación para mejorar los procesos. La mayoría de
 243 eventos son predecibles y se puede ir estableciendo una forma determinada de distribuir
 244 los contenedores en función de los resultados que se van obteniendo. Esto último, para
 245 un ser humano, resulta prácticamente imposible de realizar eficientemente.

246 Así pues, se espera tener la capacidad de realizar ciertas recomendaciones a partir de
 247 los resultados que se observen en la simulación. Por ejemplo, una determinada política
 248 de distribución de los contenedores podría dar mejores resultados que las demás de
 249 forma general. Tal vez distintas políticas sean convenientes según el tipo de trabajo que
 250 predomine en un determinado intervalo de tiempo. En el apartado 3.4 se describirán
 251 todos los parámetros del modelo.

252 **1.4. Estructura de la memoria**

253 Este documento sigue una estructura casi cronológica, exponiendo las diferentes fases
 254 del trabajo en diferentes apartados, en el mismo orden aproximado en el que se realizaron.
 255 En el capítulo 2, se explican los conceptos más importantes respecto a la optimización
 256 y la simulación, con un análisis del estado actual de dicha disciplina, y planteando las
 257 diferentes alternativas que más se utilizan actualmente.

258 En el capítulo 3 se describe el escenario a simular. Primero, en el apartado 3.1 se
 259 describe el funcionamiento básico de una terminal marítima de contenedores, con todos
 260 los elementos a tener en cuenta a la hora de plantear la simulación. Hecho esto, se
 261 propone un modelo de simulación en el apartado 3.2, con la defensa del paradigma de
 262 simulación seleccionado en el apartado 3.3. Conociendo el paradigma que se va a utilizar,
 263 se parametriza el modelo en el 3.4. Finaliza el capítulo con el apartado 3.5, donde se
 264 aclaran los objetivos que se persiguen con el diseño descrito.

265 El capítulo 4 describe la propuesta final para la solución del problema descrito en el
 266 apartado anterior. Se comienza describiendo los conceptos fundamentales a conocer (4.1),
 267 como pueden ser Java o Maven. Tras esta introducción, se entra en materia en el apartado

268 4.2, describiendo el funcionamiento de la librería de simulación utilizada para construir el
269 modelo. Otra herramienta muy importante, utilizada conjuntamente con la propia librería,
270 se describe en el apartado 4.3. Finalmente, en el 4.4, se incluyen y explican los diagramas
271 y estructura que se ha seguido a la hora de programar la simulación.

272 En el capítulo 5, se entra en más detalle sobre cómo se programa el modelo. Se
273 hace uso de términos más avanzados y probablemente desconocidos para lectores no
274 experimentados con algún lenguaje de programación orientado a objetos. Aunque no
275 se entra en detalles sobre cómo es el código en sí, se explica la forma en la que se ha
276 programado la mayoría de los parámetros y procedimientos de la simulación.

277 El capítulo 6 describe la experimentación realizada con el modelo construido. Esta
278 se lleva a cabo para demostrar la utilidad para la que está pensado el modelo. En
279 sucesivas secciones, se va experimentando con múltiples parámetros y con las diferentes
280 funcionalidades implementadas.

281 El capítulo 9 describe el presupuesto de este proyecto.

282 Finalmente, se cierra este documento con un capítulo final de conclusiones (capítulo
283 7). En la sección 7.1 se analiza el resultado final de la librería en función de lo que
284 había planeado en un principio. También se hace mención a los posibles usos que se le
285 pueden dar a la herramienta (sección 7.2), y se finaliza haciendo un repaso de posibles
286 implementaciones futuras que podrían realizarse para mejorar el proyecto (sección 7.3).

²⁸⁷ **Capítulo 2**

²⁸⁸ **Simulación y optimización**

²⁸⁹ **2.1. Conceptos**

²⁹⁰ En matemáticas, ciencias de la computación y otras ciencias, la optimización es la
²⁹¹ selección del mejor elemento de un conjunto, según un determinado criterio. Específicamente,
²⁹² un problema de optimización consiste en minimizar o maximizar el valor de
²⁹³ una determinada función objetivo. Se trata de un área muy amplia de las matemáticas
²⁹⁴ aplicadas.

²⁹⁵ Ahora bien, un problema real, aplicable a las necesidades de una empresa, requiere
²⁹⁶ satisfacer simultáneamente múltiples criterios de desempeño u objetivos. Estos objetivos
²⁹⁷ suelen ser independientes entre sí; pueden llegar a ser contradictorios. Para tales casos,
²⁹⁸ se debe comprobar si es factible la combinación de los objetivos de manera adecuada,
²⁹⁹ tratando de considerarlos como un único objetivo a optimizar [22]. La solución del
³⁰⁰ problema se obtendría entonces encontrando el máximo o mínimo de una función objetivo
³⁰¹ que agrupa todos los objetivos que se desean optimizar. Esto es lo que se conoce como
³⁰² optimización mono-objetivo.

³⁰³ Sin embargo, combinar todos los objetivos es una tarea sin duda complicada. En
³⁰⁴ sistemas complejos, podrían perseguir estados totalmente opuestos, lo que imposibilita
³⁰⁵ encontrar una manera óptima de realizar la combinación. Un ejemplo relacionado con
³⁰⁶ este proyecto puede ilustrarse a través de los vehículos que desplazan los contenedores
³⁰⁷ en la terminal. El aumento del número de estos vehículos podría disminuir los tiempos de
³⁰⁸ espera de las mercancías que contienen los contenedores, pero por otro lado generaría
³⁰⁹ un mayor gasto en combustible, necesidad de un mayor número de reparaciones, etc. En
³¹⁰ estos casos, se dice que el problema es un problema de Optimización Multiobjetivo [22].

³¹¹ Cuando se habla de optimización multiobjetivo, son muy comunes las referencias a
³¹² un término económico llamado Óptimo de Pareto, también conocido como Eficiencia de
³¹³ Pareto u Optimalidad de Pareto. Se trata de un criterio de eficiencia que afirma que una
³¹⁴ asignación de bienes es óptima (en el sentido de Pareto) cuando no hay posibilidad de
³¹⁵ redistribución para que una persona mejore sin empeorar la situación de las demás [14].
³¹⁶ Esto se aplica en ingeniería a la optimización de sistemas complejos, sustituyendo las
³¹⁷ referencias a personas por referencias a los objetivos que hay que optimizar. Así, una
³¹⁸ solución es pareto-óptima cuando no existe ninguna otra solución que la domine [2], es
³¹⁹ decir, mejore alguno de los objetivos sin desmejorar otro simultáneamente [6].

³²⁰ Así pues, dados unos valores de entrada para los objetivos, se busca un cambio hacia
³²¹ una nueva asignación que mejore al menos a uno de los objetivos sin perjudicar a los
³²² demás. Esto se conoce como mejora de Pareto. Estas mejoras se van aplicando mientras

323 sea posible; en el momento en que dejen de serlo, la asignación será pareto-óptima. Por
 324 supuesto, hay que tener en cuenta que el hecho de que un cambio genere beneficio para
 325 un objetivo sin perjudicar a otro, no implica que ese cambio guíe al problema a través
 326 de un proceso natural de optimización hasta alcanzar el punto óptimo. Además, en todo
 327 problema de optimización multiobjetivo real, los objetivos pueden no tener el mismo
 328 nivel de importancia dentro del sistema, proporcionando unos un mayor beneficio al ser
 329 mejorados.

330 **2.2. Simulación y Optimización**

331 En sistemas con un alto número de entidades con intereses propios que deben tratar
 332 de optimizar sus operaciones, se puede recurrir a la optimización basada en simulaciones.
 333 Ésta aprovecha simulaciones por ordenador para obtener más información sobre el
 334 comportamiento de un sistema que haya sido previamente modelado. El rendimiento
 335 del sistema se mejora modificando el estado inicial de cada parámetro del modelo en
 336 cada ejecución, ejecutando múltiples veces, con cada iteración estando más cerca de la
 337 solución óptima.

338 Así pues, un modelo de simulación describe de forma muy precisa el comportamiento
 339 a lo largo del tiempo de toda o partes de un sistema, en función de sus parámetros y
 340 estrategias o políticas de funcionamiento seleccionadas [16]. Aunque desde el punto
 341 de vista metodológico, las diferencias entre simulación y optimización son bastante
 342 evidentes, queda mucho menos claro en la práctica real, donde la simulación puede ser
 343 el único método capaz de proporcionar información precisa y suficiente para mejorar el
 344 rendimiento del sistema. Especialmente cuando el impacto de los efectos aleatorios es
 345 significativo, lo cual es el caso en este estudio, como queda patente en el capítulo 6 de
 346 este documento.

347 La integración de modelos de optimización con los de simulación para optimizar
 348 procesos complejos es un campo de investigación abierto, ya que no se ha resuelto de
 349 forma efectiva a día de hoy [16]. Los modelos de simulación, generalmente, se desarrollan
 350 para la ayuda en aspectos muy interesantes para este caso de estudio, como son la toma
 351 de decisiones y la evaluación de diseños alternativos [16].

352 **2.3. Estado del arte**

353 "La simulación es el proceso de diseñar un modelo de un sistema real y llevar a término
 354 experiencias con él, con la finalidad de comprender el comportamiento del sistema o
 355 evaluar nuevas estrategias para su funcionamiento"[20]. Realizar un estudio de simulación
 356 implica una serie de pasos:

- 357 ■ Definir el sistema. Estudiar el contexto del problema para identificar los objetivos
 358 de la simulación.
- 359 ■ Formular el modelo. Se definen las variables y parámetros y el funcionamiento que
 360 se esperan de la simulación.
- 361 ■ Reunir datos. En este aspecto, este trabajo de fin de grado se encuentra algo más
 362 limitado, ya que no se dispone de datos exactos acerca de ciertos aspectos del
 363 puerto que se tomará como referencia (Santa Cruz de Tenerife), como pueden ser

364 las especificaciones de las grúas o los vehículos. Sin embargo, el objetivo es diseñar
 365 el modelo de tal manera que pueda modificarse y adaptarse a un buen número de
 366 puertos.

- 367 ■ Implementar el modelo. Para ello, se hace uso de una librería de simulación, que se
 368 describe en la sección 4.2 de este documento.
- 369 ■ Verificación y validación. Se comprueba que el modelo actúa como se había estable-
 370 cido.
- 371 ■ Experimentar con los resultados, modificando parámetros para tratar de mejorar
 372 los resultados.
- 373 ■ Interpretar y documentar los resultados (capítulo 6).

374 Dependiendo del tipo de simulación que se quiera llevar a cabo, hay ciertos paradigmas
 375 o modelos de simulación que son más indicados que otros. Cuando se hace referencia al
 376 estado del arte de esta disciplina, destacan tres paradigmas:

- 377 ■ *Dinámica de sistemas*. Se aplica a sistemas dinámicos que se caracterizan por inter-
 378 dependencia, interacción mutua, retroalimentación de la información y causalidad
 379 circular [5]. Se basa en la identificación de bucles de realimentación entre ele-
 380 mentos, estructurando la dinámica de comportamiento de los sistemas a través de
 381 modelos matemáticos. Jay Forrester, considerado el creador de la disciplina, utilizó
 382 este paradigma para demostrar que el actual crecimiento de la población mundial
 383 es insostenible por más de 100 años [5].
- 384 ■ *Simulación basada en eventos discretos*. Implica, como su propio nombre indica, la
 385 simulación de sistemas que evolucionan a lo largo del tiempo por medio de eventos
 386 discretos, en contraposición a aquellos cuyos eventos simulados son continuos [12].
 387 Estos eventos son almacenados por el simulador en una lista, ordenados en el tiempo
 388 con un cierto intervalo entre ellos. Al sucederse, modifican el estado del sistema y
 389 añaden otros eventos nuevos a la lista [9]. Podría ser una opción válida a la hora de
 390 simular el presente caso de estudio, aunque ciertos eventos como el desplazamiento
 391 de los vehículos sería más conveniente modelarlos como eventos continuos.
- 392 ■ *Simulación basada en agentes*. Este tipo de simulación se centra en los componentes
 393 individuales del sistema a simular, en contraste con la dinámica de sistemas, algo
 394 más abstracta, y la simulación de eventos discretos, más centrada en los procesos.
 395 Las entidades activas del sistema, como pueden ser personas, vehículos, equipa-
 396 miento, o cualquier componente relevante dentro del mismo, se identifican y simulan
 397 individualmente, definiendo su comportamiento para reflejar el que tienen en la
 398 realidad y estableciendo conexiones entre ellos [7] [3] [4]. Se les otorga un estado
 399 interno y unas reglas que modifican dicho estado en cada unidad de tiempo, depen-
 400 diendo de los efectos que el resto de agentes producen sobre ellos y en el conjunto
 401 del sistema.

402 **2.4. Estado estacionario**

403 Una simulación se desarrolla a lo largo de un intervalo de tiempo determinado, que
 404 puede ser un día, una semana, un mes, etc. A lo largo de este *horizonte temporal*, los

405 atributos de las entidades de la simulación (agentes para el caso de la simulación basada
406 en agentes) van adoptando diversos valores. Estos valores determinan el *estado* del
407 sistema [21].

408 El estado puede ser estático o estacionario, constante en el tiempo; o puede ser
409 dinámico o transitorio, si evoluciona con el tiempo. Por lo general, un sistema inicia su
410 funcionamiento en estado dinámico, y según evoluciona alcanza cierta estabilidad y pasa
411 a un estado estacionario [10]. Este puede ser estable o inestable, dependiendo de si el
412 efecto de las perturbaciones lo devuelve a un estado dinámico [21].

413 Así pues, lo ideal es tomar como referencia los datos de los períodos de simulación en
414 los que la misma se encuentra en estado estacionario o cerca del mismo. Básicamente,
415 se trata de sacar conclusiones de una situación estable, realista. La simulación de una
416 semana de una terminal marítima debería tener en cuenta que la terminal ya estaba en
417 funcionamiento antes de la simulación, y que seguirá en funcionamiento después de la
418 misma. Debería haber contenedores en tránsito o almacenados desde antes del comienzo
419 de esa semana, y contenedores que permanecerán después de la misma. Igualmente,
420 debería haber vehículos y grúas activas y, posiblemente, en tránsito. Para lograr esto,
421 una buena práctica es simular un tiempo anterior y un tiempo posterior, buscando una
422 aproximación al estado estacionario, del cual se puedan sacar conclusiones válidas.

423 Capítulo 3

424 Problema

425 3.1. Terminal marítima de contenedores

426 La figura 3.1 muestra una terminal marítima de contenedores al completo. Como
427 ya fue descrito brevemente en la sección 1.1, se trata de una infraestructura dividida
428 principalmente en tres áreas funcionales o subsistemas [18]:

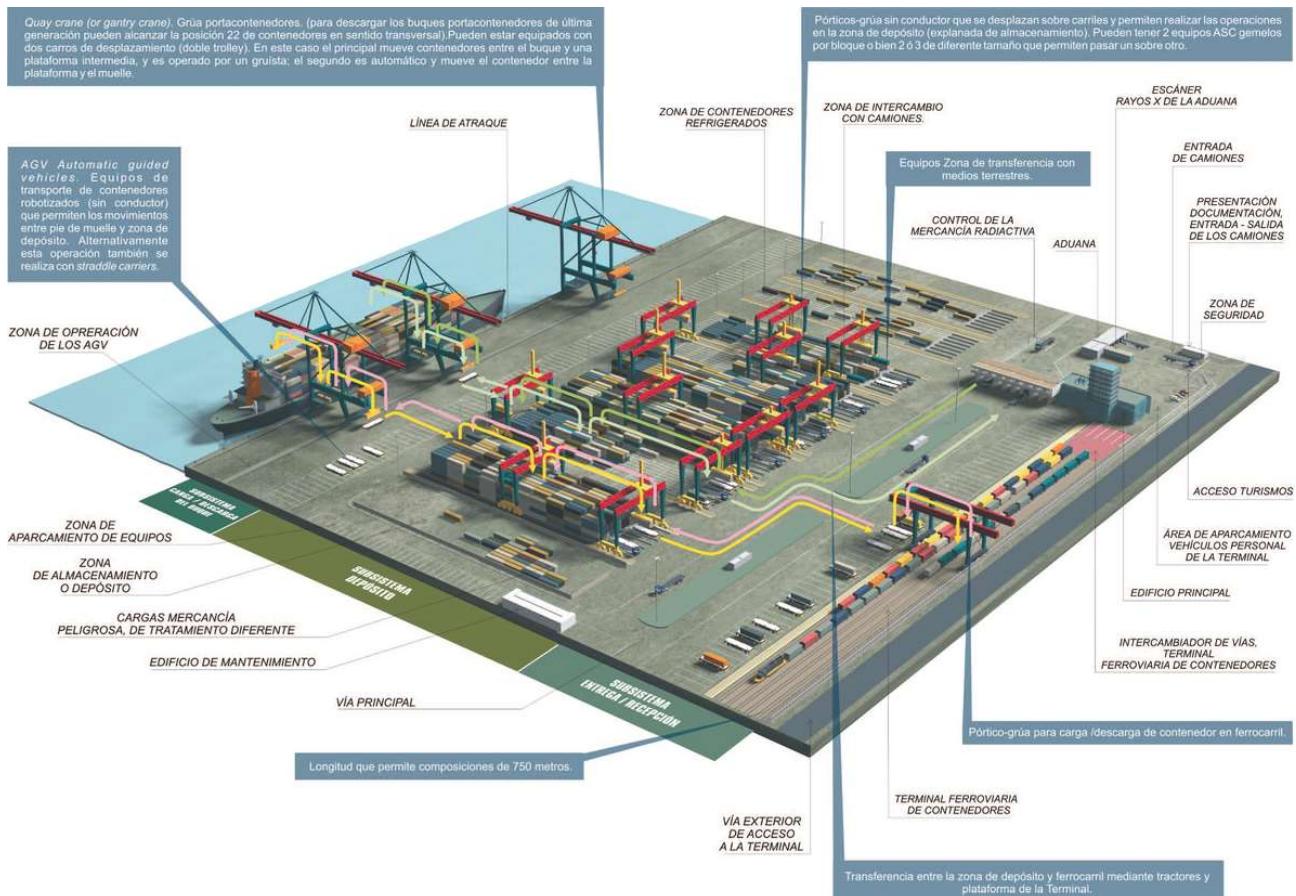


Figura 3.1: Representación de una terminal marítima de contenedores

- Zona marítima o subsistema de carga y descarga de buques. Sección que permite la entrada de embarcaciones al puerto. Tiene diversos parámetros “físicos” que no son del interés de este problema, como puede ser la profundidad de sus aguas. Sí es de interés, en cambio, el número de atraques o la capacidad de transporte de los barcos que llegan (número de contenedores que podrían transportar).
- Zona terrestre o subsistema de recepción y entrega. Interfaz que conecta la terminal con los medios de transportes terrestres (camiones y/o trenes), que traen contenedores para su almacenaje y posterior exportación, o viceversa, recogen contenedores de importación.
- Zona de enlace o subsistema de almacenamiento. También llamada patio, es la superficie plana donde se almacenan los contenedores apilados. Contiene las instalaciones, personal y vehículos que permiten el acceso, circulación, estacionamiento y operación de los modos de transporte terrestre de carga, así como las destinadas al almacenamiento de transferencia de las mercancías operadas. También comprende otras infraestructuras como los circuitos de reconocimiento aduanero, que no son del interés de este problema.

Como se puede ver, el rendimiento de las zonas marítimas y terrestre está directamente relacionado con la productividad de la zona de enlace. Una rápida y eficiente circulación de contenedores en la zona de enlace supone un menor tiempo de espera para los barcos en la zona marítima y para los medios de transporte terrestre en la zona terrestre.

A continuación, se lleva a cabo la propuesta de un modelo de simulación que represente el esquema anterior, de tal forma que se puedan analizar sus principales parámetros y rendimiento.

3.2. Propuesta de modelo de simulación

Sea una terminal marítima de contenedores caracterizada por lo siguiente:

- El patio o zona de enlace se divide en m bloques de contenedores. Estos se disponen de forma perpendicular a la línea de atraque, tal y como se puede apreciar en la imagen 3.1. Esa variable m será un parámetro que podrá modificarse en cada instancia del problema.
- Cada bloque tendrá S pilas, T alturas y B bahías. Una bahía es una fila de pilas de contenedores dentro de un bloque y paralela a la línea de atraque de los barcos. Cada pila tiene una cierta altura. Esto da lugar a que en cada bloque puedan ubicarse hasta $S \times T \times B$ contenedores al mismo tiempo. Al igual que en el caso anterior, estas tres variables podrán tener el valor que se desee en cada instancia del problema.
- Cada uno de los m bloques está servido por una grúa que puede moverse a lo largo del bloque correspondiente. Por tanto, se tienen m grúas.
- Los tiempos asociados a las operaciones de alzado, desplazamiento y suelta de los contenedores por parte de las grúas, vienen definidos según una distribución de probabilidad triangular. Será proporcional a la distancia entre las posiciones entre las cuales vaya a desplazarse la grúa.

470 Es importante el concepto de **trabajo** en la simulación propuesta. En particular se
 471 trata de una petición de movimiento de un contenedor desde una posición inicial hasta
 472 una posición final dentro de la terminal. Existen tres tipos de trabajos:

- 473 ■ Almacenamiento de un contenedor. Un contenedor descargado de un barco o que
 474 llega por tierra en un camión externo debe ser almacenado en uno de los bloques
 475 del patio. La posición de almacenamiento del contenedor es conocida.
- 476 ■ Retirada de un contenedor. Un contenedor es solicitado para ser cargado en barco o
 477 retirado de la terminal mediante un camión externo. La posición en que se encuentra
 478 el contenedor es conocida.
- 479 ■ Reubicación de un contenedor. Un contenedor es solicitado para ser reubicado en
 480 una posición diferente a la que se encuentra actualmente. Las posiciones actuales y
 481 futuras del contenedor son conocidas.

482 Es importante tener en cuenta las diferencias entre el flujo de trabajo que proviene
 483 de peticiones por mar y el flujo de trabajo que proviene de peticiones por tierra. Las
 484 peticiones por tierra normalmente vienen dadas por camiones que son capaces de extraer
 485 y solicitar la introducción de contenedores de forma muy limitada (de uno en uno),
 486 mientras que las peticiones por mar vienen dadas por barcos capaces de transportar
 487 cientos de contenedores. El número de peticiones por tierra será mucho más numerosa,
 488 pero cada petición por mar involucrará a un número de contenedores mucho mayor.

489 El desplazamiento de contenedores dentro de la terminal corre a cargo de un conjunto
 490 n de vehículos de reparto interno. Se trata de vehículos AGV (*Automated Guided Vehicle*),
 491 con capacidad para transportar un único contenedor al mismo tiempo. Se considerará
 492 que todos estos vehículos tienen el mismo rendimiento (velocidad, aceleración, etc.).
 493 El tiempo que tarda un vehículo en desplazarse entre dos posiciones dependerá de la
 494 distancia entre ellas y vendrá caracterizada por una variable aleatoria resultado de una
 495 distribución de probabilidad, de forma similar a los tiempos de desplazamiento de las
 496 grúas.

497 3.3. Paradigma de simulación

498 Para la simulación, se ha seleccionado el paradigma basado en agentes. Su premisa
 499 de funcionamiento se asemeja bastante al comportamiento previamente descrito de los
 500 elementos de un puerto: existen una serie de "individuos", diseñados como agentes en la
 501 simulación, que tienen características propias y que dependen del resto de individuos
 502 para llevar a cabo sus funciones. Cada uno de ellos tiene unos intereses propios y sus
 503 interacciones definen la calidad de los resultados que se obtienen.

504 Tradicionalmente, los modelos de simulación han tratado a elementos como compañías,
 505 productos, recursos o clientes como grupos uniformes, entidades pasivas que actúan como
 506 una sola [15]. En la Dinámica de Sistemas, por ejemplo, se tomarían afirmaciones como
 507 "Disponemos de 20 vehículos que pueden transportar exitosamente unos 10 contenedores
 508 por minuto". En simulación por eventos discretos, todas las operaciones se verían como un
 509 conjunto de procesos: "El barco A llega al punto de atraque 2, solicita el almacenamiento
 510 de un 20 % de sus contenedores y la retirada de una cantidad similar de contenedores.
 511 El tiempo de espera es aproximadamente...". Aunque se trata de aproximaciones per-
 512 fectamente válidas, ignoran la composición única y las complejas interacciones entre

513 las entidades individuales [11]. Por ejemplo, la disponibilidad de una grúa en concreto
 514 podría estar condicionada por sus propias necesidades de reparación o comprobaciones
 515 de seguridad, que no tienen nada que ver con esas mismas necesidades para otras grúas.

516 Todos y cada uno de los elementos descritos en el apartado anterior pueden ser
 517 modelados como agentes: los AGV, las grúas, los propios contenedores, etc. Estos últimos
 518 requieren de los dos anteriores para poder desplazarse. Al mismo tiempo, los AGV ven
 519 condicionadas sus operaciones por el comportamiento de las grúas, por las cuales tienen
 520 que esperar para delegarles los contenedores que desplazan. De la misma manera, si
 521 un vehículo o una grúa deja de estar disponible durante un tiempo determinado, todo el
 522 sistema se vería afectado.

523 Estas interacciones van mucho más allá. El comportamiento que tengan los vehículos a
 524 la hora de asignar los contenedores a los bloques tiene un efecto directo en la cantidad de
 525 trabajo que tienen que desempeñar las grúas. Por lo tanto, no solo se condicionan entre
 526 ellos por defecto, sino que los cambios en las políticas de uno de ellos también condiciona
 527 enormemente al resto.

528 Gestionar estos elementos como agentes individuales permite asignarles un compor-
 529 tamiento por defecto, y observarlos uno a uno y como conjunto, así como simular sus
 530 operaciones simultáneas para recrear y predecir las acciones de fenómenos más comple-
 531 jos. El nivel de detalle que puede llegar a lograrse puede ser desde el más elemental al
 532 más elevado.

533 **3.4. Parámetros del modelo**

534 En base a la descripción previa y observando los diagramas, debemos tener en cuenta
 535 los siguientes parámetros:

536 ■ m. Número de bloques y grúas de patio.

537 ■ Dimensiones de los bloques:

538 • S. Número de pilas de ancho.

539 • T. Número de alturas.

540 • B. Número de bahías.

541 ■ n. Número de vehículos internos.

542 ■ Períodos de disponibilidad de las grúas. Cabe la posibilidad de que cierta grúa deje
 543 de estar disponible durante un tiempo determinado. Esto puede darse por necesidad
 544 de reparación, comprobaciones de seguridad, etc.

545 ■ Períodos de disponibilidad de los vehículos internos. Lo mismo sucede con los AGV,
 546 que podrían dejar de estar disponibles durante un cierto periodo por reparaciones,
 547 comprobaciones de seguridad, repostar para combustible, etc.

548 ■ Probabilidad de fallo de cada una de las grúas. La maquinaria puede fallar, y eso
 549 es un factor que la simulación debe contemplar. La probabilidad de que una grúa
 550 falle vendría dada, como en casos anteriores, por un valor aleatorio dado por una
 551 distribución de probabilidad.

- Probabilidad de fallo de cada uno de los vehículos internos. Es un parámetro similar al anterior pero aplicado a vehículos.
- Nivel de prioridad de los contenedores. No todos los contenedores tienen por qué tener la misma relevancia. Es posible que se sepa que alguno de ellos tiene que ser procesado más rápidamente, ya sea porque su mercancía es necesitada urgentemente en otro lugar o porque se conoce que en poco tiempo será recogido, entre otras razones. Los contenedores con mayor prioridad deberían tener preferencia a la hora de avanzar por el sistema.
- Política de asignación de contenedores entrantes a bloques. Los contenedores pueden ir ubicados en bloques de acuerdo a las siguientes políticas:
 - Aleatorio. Se determina el bloque de destino generando números de forma pseudo-aleatoria. Todos los números tienen idéntica probabilidad de ser generados.
 - Bloque más vacío. A fin de no saturar ninguno de los bloques de contenedores, se podría aplicar una política que asigne cada nuevo contenedor al bloque más vacío en ese instante.
 - Bloque cuya grúa tenga menos carga de trabajo. Esta opción evitaría la sobrecarga de trabajo en ciertas grúas mientras otras estarían mucho tiempo sin ser utilizadas.
 - Bloque en el cual haya un menor número de contenedores esperando por la grúa. Esto supondría que el contenedor iría directamente a aquel bloque en el cual el tiempo de espera para ser almacenado sería, a priori, menor.

3.5. Objetivos de la simulación

Con toda la información descrita hasta el momento, el objetivo de la simulación, en términos más concretos, es crear un modelo que permita realizar la gestión de contenedores en la zona de enlace de una terminal marítima. Esta gestión se basa en la aparición de flujos de trabajos que podrían requerir el almacenamiento, retirada o reubicación de contenedores.

En este contexto, se pueden destacar los siguientes objetivos concretos:

- Determinar el número de operaciones por fracción de tiempo (hora, día, semana, etc.) realizado por el conjunto de grúas en función del flujo de contenedores requeridos, así como el tiempo de uso por fracción de tiempo.
- Número de operaciones por fracción de tiempo realizado por el conjunto de vehículos en función del flujo de contenedores requeridos, así como el tiempo de uso por fracción de tiempo.
- Analizar el impacto de las políticas de asignación de contenedores a bloques. Se trata de averiguar la política que mejor funciona en la mayoría de casos de entre las ya descritas en el apartado anterior: aleatorio, bloque más vacío, etc.

- 590 ■ Analizar el impacto en el rendimiento que tienen las variaciones en el número de
 591 vehículos disponibles. Un mayor número de vehículos, en general, supondría un
 592 menor tiempo de espera para los contenedores. Sin embargo, a partir de un cierto
 593 número de vehículos, se podría dar el caso en el que las mejoras en el rendimiento
 594 sean mínimas o incluso inexistentes. En esos casos, es mejor reducir el número de
 595 vehículos para aumentar el ahorro en recursos. En las distintas ejecuciones de la
 596 simulación, se tratará de buscar el número de vehículos más adecuado.
- 597 ■ Relacionado con el punto anterior, se puede modificar el número de accesos por
 598 tierra y mar a la terminal para comprobar el efecto que tiene esto en el rendimiento.
 599 Un mayor número de puntos de atraque para barcos facilitaría el acceso de más
 600 contenedores al mismo tiempo, pero aumentaría el flujo de trabajo y eso podría
 601 demandar un mayor número de vehículos, haciendo que los tiempos de espera
 602 puedan aumentar en algunos casos. Lo mismo ocurre con los accesos por tierra:
 603 múltiples accesos suponen múltiples colas, que a priori serían independientes pero
 604 que, al depender de los mismos recursos, podrían o no suponer una mejora.
- 605 ■ Analizar en qué grado afectan al rendimiento los períodos en los que un recurso
 606 queda fuera de disponibilidad debido a eventos conocidos, es decir, que puedan pre-
 607 decirse de antemano. Estos eventos son los ya mencionados períodos de reparación
 608 de grúas y vehículos, tiempos que los vehículos necesitan para repostar combustible,
 609 etc.
- 610 ■ Medir el impacto en el rendimiento que tienen los períodos en los que un recurso
 611 queda fuera de disponibilidad debido a eventos no conocidos, imposibles de predecir,
 612 como fallos mecánicos. Al contrario que en el caso anterior, para este no existe una
 613 forma clara de modificar las políticas del modelo con el fin de reducir el impacto de
 614 estos eventos, pues no siguen ningún patrón. Sin embargo, es interesante valorar
 615 hasta qué punto hay que tener en cuenta la posibilidad de que ocurran.
- 616 ■ Analizar el impacto en el rendimiento que tiene la asignación de niveles de prioridad
 617 a los diferentes contenedores. Conceder una mayor prioridad a un contenedor que
 618 lleva menos tiempo en el sistema retrasa los contenedores de menor prioridad que
 619 han estado más tiempo en espera o en tránsito. Esto podría afectar al rendimiento.

620 Todas estas mediciones se llevan a cabo en el entorno descrito previamente, en el que
 621 se tienen en cuenta los posibles fallos en el funcionamiento de los recursos, los períodos
 622 de disponibilidad de los mismos, las limitaciones en la velocidad de los vehículos y las
 623 grúas, etc. A grandes rasgos se trata de tomar los parámetros descritos en el apartado
 624 anterior y modificar sus valores de entrada, con el fin de encontrar el balance que otorgue
 625 un mejor rendimiento a las operaciones de la terminal.

626 Capítulo 4

627 Propuesta de solución

628 4.1. Conceptos básicos

629 En esta sección, se explica con más detalle el modelo propuesto para la simulación,
630 así como su implementación, previo paso al examen de los resultados obtenidos. Por
631 supuesto, multitud de elementos forman parte de la implementación, elementos que son
632 explicados con detalle.

633 A continuación, se presentan una serie de conceptos relevantes para la implementación
634 del modelo, con los cuales una persona experimentada en programación debería estar
635 familiarizada. Sin embargo, para personas con poca o ninguna experiencia en la materia,
636 podría ser necesaria una explicación. Ahora bien, estos conceptos no serán referenciados
637 muy a menudo en el resto de este documento, debido a que son términos que directamente
638 se dan por sentado. En esencia, es por eso por lo que son tan importantes.

- 639 ■ Java. Se trata de un lenguaje de programación orientado a objetos ampliamente
640 utilizado para todo tipo de sistemas operativos desde hace ya muchos años. Actual-
641 mente, su foco principal está en el desarrollo para la plataforma móvil Android,
642 pero en sistemas de escritorio sigue siendo muy importante y, sobre todo, muy
643 apoyado por la comunidad. Pocos lenguajes pueden presumir de contar con una
644 documentación tan abundante como Java. Por todo esto, sumado a que es el lenguaje
645 de programación seleccionado para la implementación de la librería de simulación
646 (como se explica en la sección 4.2), ha sido utilizado para codificar el modelo de
647 simulación.
- 648 ■ Librería. En programación, una librería es un conjunto de implementaciones funcio-
649 nales que se codifican en un determinado lenguaje de programación para ofrecer
650 una interfaz bien definida que permita realizar con ella operaciones, con el fin de
651 implementar una funcionalidad superior. Por ejemplo, una librería de operaciones
652 matemáticas complejas permitiría a un programador utilizar funciones para realizar
653 la raíz cuadrada de un número, obtener su valor absoluto, etc, sin necesidad de
654 que el programador realice ese trabajo, puesto que ya está hecho. Una librería de
655 simulación, por ende, debería facilitar funciones que permitan el diseño de una
656 simulación, sin que el programador deba preocuparse por su funcionamiento más
657 elemental.
- 658 ■ Maven (Apache Maven)¹. Se trata de una herramienta de software para la gestión de

¹<https://maven.apache.org/>

659 proyectos Java. Gestiona la construcción y documentación de un proyecto desde un
 660 fichero central con toda la información necesaria. Este es un fichero “Project Object
 661 Model” (POM) con formato XML [1]. La librería de simulación ha sido construida
 662 utilizando Maven, lo cual facilita mucho su importación como librería en un nuevo
 663 proyecto. Puesto que la idea es transformar la propia simulación en una librería que
 664 pueda importarse y ejecutarse con diversos parámetros, también se utiliza Maven.

665 Estos conceptos son importantes porque el resultado final del trabajo es una nueva
 666 librería Java que podrá utilizarse para ejecutar simulaciones de una terminal determinada,
 667 a partir de una serie de parámetros de entrada.

668 4.2. Librería de simulación

669 Definidos los conceptos fundamentales, a continuación se describe la librería de simu-
 670 lación utilizada. En los primeros apartados ya se menciona la herramienta para diseño de
 671 simulaciones *Anylogic*, la cual representa uno de los mejores software de simulación para
 672 empresas que hay actualmente en el mercado. Es una herramienta extremadamente po-
 673 tente y avanzada, un software profesional reputado entre las mejores empresas (*Anylogic*
 674 presume de tener como clientes a grandes empresas como Google), lo cual implica que
 675 también es bastante caro. No está diseñado para el uso personal, siendo excesivo incluso
 676 para empresas pequeñas.

677 En su lugar, se hace uso de una librería de simulación escrita en Java y gestionada
 678 utilizando Maven, diseñada por Christopher Expósito Izquierdo. Esta librería se aloja en
 679 GitHub en un repositorio privado y no se comercializa, pero ha sido cedida por su autor
 680 para su uso en este trabajo de fin de grado. Su finalidad, a grandes rasgos, es similar a la
 681 que persigue *Anylogic*: proporcionar un entorno para el diseño de simulaciones utilizando
 682 el paradigma de simulación basada en agentes.

683 La librería que se va a utilizar, la cual es referenciada a partir de aquí como simplemente
 684 la librería de simulación, se utiliza para codificar directamente el modelo de simulación,
 685 utilizando el lenguaje de programación Java en este caso (aunque podrían utilizarse
 686 otros lenguajes compatibles con Java *bytecode*, como Kotlin). Al contrario que *Anylogic*,
 687 no se ofrece ningún tipo de interfaz de usuario para elaborar la simulación, sino que
 688 permite acceder directamente a sus clases y métodos para aprovecharlas para programar
 689 las necesidades que se requiera. Evidentemente, esta forma permite un control más
 690 exhaustivo de todos los aspectos de la simulación. Nótese que *Anylogic* también permite
 691 construir la simulación de esta manera pero, de nuevo, con un coste excesivamente alto.

692 El funcionamiento de la librería se basa en muchos de los elementos en los que se
 693 fundamentan otras herramientas de simulación para construir modelos de simulación. Se
 694 utilizan diversos tipos de agentes con una serie de particularidades y métodos predefini-
 695 dos, llamados bloques, que se conectan entre sí para formar un diagrama. A través de este
 696 diagrama “circularán” los agentes que se diseñan específicamente para la simulación que
 697 se esté programando. Los bloques se personalizan para hacer que redirijan y manipulen a
 698 estos agentes de la forma adecuada.

699 Es importante no confundir el término *bloque* referido a agente complejo de la si-
 700 mulación con el concepto de bloque de contenedores (lugar de almacenamiento de los
 701 contenedores en la terminal). Para evitar esta confusión, en muchos casos se hace refe-
 702 rencia a los primeros como “nodos”, pues hacen las veces de nodos en los diagramas de
 703 simulación. En el apartado 4.4 se explica todo lo relativo a estos diagramas.

704 A continuación, se especifican aquellos nodos que son utilizados en el modelo:

- 705 ■ **Source** (fuente). Estos nodos son los que se encargan de generar los agentes que
706 van a formar parte de la simulación. Los generan de un determinado tipo, cada
707 cierto tiempo, con un cierto estado inicial. En el caso de esta simulación, los nodos
708 source generan agentes de tipo Contenedor, que son los que circulan por el sistema
709 y son el objetivo primordial de las operaciones de la terminal.
- 710 ■ **Delay** (retraso). Se trata de nodos que retrasan al agente un tiempo determinado
711 antes de permitirle continuar. Por ejemplo, el tiempo que pasa entre que un vehículo
712 recoge un contenedor y lo suelta en su lugar de almacenamiento se representa
713 mediante un bloque *Delay*.
- 714 ■ **SelectOutput** y **SelectOutputN** (selección de salida). Estos nodos permiten bifurcar
715 el flujo del diagrama para que el agente pueda seguir dos o más caminos diferentes
716 dependiendo de ciertas condiciones, que pueden ser propias o del sistema. Se
717 selecciona la salida a través de la cual continúa el agente. Un nodo *SelectOutput*
718 tiene dos posibles salidas, mientras que un *SelectOutputN* tiene N posibles salidas
719 (personalizado).
- 720 ■ **Resource** (recurso). Un recurso es otro agente que puede ser incautado, tomado
721 u ocupado por otro, durante un tiempo determinado, para luego ser liberado. Una
722 vez liberado, puede ser ocupado por otro agente. Tienen un estado interno que los
723 distingue entre recursos libres (no han sido ocupados), reservados (no han sido
724 ocupados pero ya se conoce a los próximos agentes que los ocuparán) u ocupados.
725 Un ejemplo de recurso son los vehículos o las grúas dentro de la terminal.
- 726 ■ **Resource Pool** (*pool* o conjunto de recursos). Como su propio nombre indica, se trata
727 de un conjunto de agentes *resource* que se asocian a otro nodo o nodos, estando
728 disponibles para ser ocupados (o liberados) por los agentes que pasen por dichos
729 bloques.
- 730 ■ **Queue** (cola). Cola de agentes. Permite retenerlos en un cierto orden antes de
731 permitirles avanzar al siguiente nodo. Normalmente precede al siguiente tipo.
- 732 ■ **Seize**. Cuando un agente llega a un nodo de este tipo, se le intenta asignar un
733 determinado recurso de un *Resource Pool* asociado. Mientras no haya recursos
734 disponibles, retiene a los agentes. Esta retención funciona de forma similar a una
735 cola, pero no permite modificar su orden.
- 736 ■ **Release** (liberación). Cuando un agente llega a un nodo de este tipo, se libera el
737 recurso que tenía ocupado después de haber pasado por un bloque *Seize*. Al igual
738 que estos, tienen asignado un *Resource pool* (el mismo que se le asignaba al *Seize*
739 relacionado), al cual pertenece el recurso o recursos que se van a liberar.
- 740 ■ **Service** (servicio). Se trata de un nodo especial que engloba un bloque *Seize*, un
741 bloque *Delay* y un bloque *Release*, y los conecta entre sí. La idea tras este diseño es
742 que, en numerosas ocasiones, un agente va a hacer uso de uno o varios recursos,
743 durante un tiempo determinado, para luego liberarlos. Así, en el *Seize* se ocupan los
744 recursos, el *Delay* retrasa el avance a través del diagrama representando el tiempo
745 de uso del recurso, y el *Release* finalmente lo libera.

- 746 ■ *Sink* (pozo). Es el nodo final, al cual llegan agentes para ser destruidos. Un agente
 747 debería ser redirigido aquí en el momento en que abandona la simulación.

748 Todos los nodos mencionados poseen puertos de entrada y puertos de salida, que
 749 permiten conectarlos entre sí para que los agentes pasen de uno a otro bajo ciertas
 750 condiciones. En estas conexiones es donde se encuentra el mayor potencial de la librería,
 751 y donde puede controlarse lo que le ocurre a los contenedores.

752 La librería de simulación pone a disposición del programador una serie de métodos
 753 que permiten controlar lo que le ocurre a los agentes en diferentes instantes a su paso
 754 por cada uno de los nodos. Estos métodos se guardan como objetos (de tipo Function,
 755 Consumer, BiFunction, etc.), y el simulador los ejecuta sobre los contenedores en el
 756 momento adecuado. Por ejemplo, los siguientes son comunes a todos tipos de nodo:

- 757 ■ *onAtEnter*: se aplica al agente cuando está a punto de entrar al nodo.
- 758 ■ *onEnter*: se aplica cuando ha entrado.
- 759 ■ *onAtExit*: se aplica cuando el agente está a punto de ser enviado por el puerto de
 760 salida.
- 761 ■ *onExit*: se aplica cuando se confirma que el agente va a ser enviado por el puerto de
 762 salida.

763 También existen casos específicos para ciertos tipos de nodo. Por ejemplo, en un objeto
 764 *Seize* se pueden establecer métodos que permiten controlar lo que le ocurre a los agentes
 765 en el momento previo a que ocupan un recurso o en el momento posterior a cuando
 766 ocupan un recurso. Por otra parte, los objetos *Delay* tienen un atributo *delayTime*, que
 767 al aplicarse a un agente devuelve un valor numérico que será el tiempo que el agente
 768 permanecerá en el *Delay* antes de ser enviado por el puerto de salida y continuar su ciclo
 769 de vida. La figura 4.1 muestra una representación, a modo de ejemplo, de un nodo de
 770 este tipo:

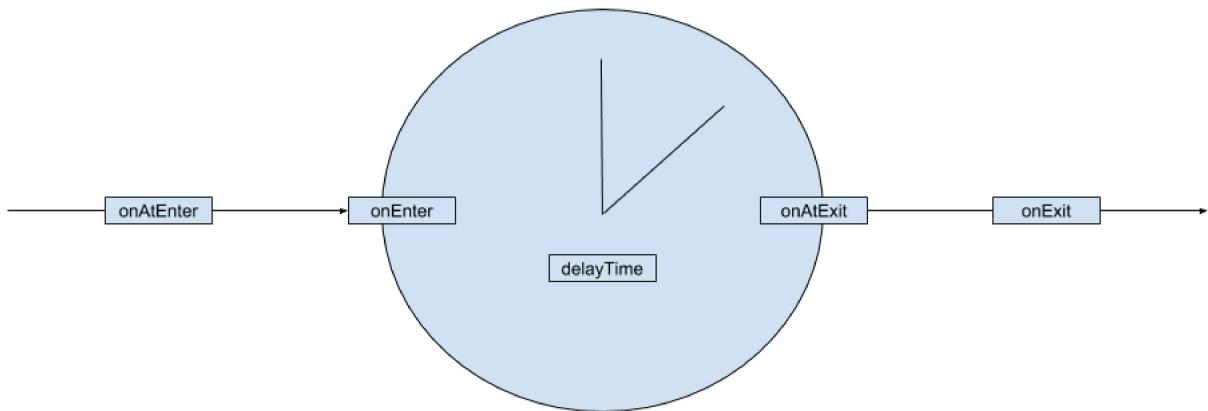


Figura 4.1: Visual de un nodo Delay

771 Por último, cabe destacar la importancia de los eventos: la clase *Event* y sus clases hijas.
 772 Prácticamente todo lo que ocurre internamente en la librería ocurre por medio de eventos,
 773 de forma totalmente transparente para el usuario que hace uso de la librería. Dicho
 774 usuario también puede programar eventos de forma manual para ejecutar segmentos de
 775 código en instantes concretos del tiempo de simulación.

776 4.3. Vega-Lite

777 Una vez se obtenga una cantidad satisfactoria de datos a partir de la simulación, lo
 778 interesante sería encontrar una manera de visualizar toda esa información de forma
 779 sencilla. El problema es que la elaboración de una interfaz que muestre dichos datos de
 780 forma gráfica podría llevar demasiado tiempo. Ahí es donde entra en juego la herramienta
 781 Vega-Lite.

782 Vega-Lite² es una gramática de alto nivel para el desarrollo de gráficos interactivos.
 783 Utilizando el extendido formato de ficheros JSON³, permite crear rápidamente visualiza-
 784 ciones de datos para análisis y presentación [19].

785 El proceso es tan sencillo como exportar los datos de los agentes de la simulación
 786 a formato JSON. Estos “objetos” JSON, que van a actuar como campos de datos, son
 787 utilizados para crear representaciones gráficas en base a la codificación especificada.
 788 Esta codificación se describe también en formato JSON. Vega-Lite también soporta otras
 789 operaciones con datos como ordenación, filtración, agregación, etc. [19]. Y además de
 790 todo esto, Vega-Lite permite generar gráficas de múltiples tipos, pudiendo personalizar la
 791 apariencia de cada una.

792 El siguiente fragmento de código JSON sirve para definir un gráfico de barras que
 793 relaciona cada letra de la A a la F con un valor numérico.

```
{
  "data": {
    "values": [
      {"a": "A", "b": 28}, {"a": "B", "b": 55},
      {"a": "C", "b": 43}, {"a": "D", "b": 28},
      {"a": "E", "b": 28}, {"a": "F", "b": 27}
    ],
    "mark": "bar",
    "encoding": {
      "x": {"field": "a", "type": "ordinal"},
      "y": {"field": "b", "type": "quantitative"}
    }
}
```

- 794 ■ El objeto `mark` especifica el tipo de gráfica.
 795 ■ El objeto `encoding` especifica la forma en que los datos (contenidos en el objeto
 796 `data`) son representados. Se puede observar cómo define las propiedades de cada
 797 eje de la gráfica por separado.

798 La idea es utilizar esta herramienta para representar y comparar los datos obteni-
 799 dos después de múltiples ejecuciones de la simulación. La facilidad para exportar la
 800 información a JSON, la sencillez de uso de la herramienta y las posibilidades que ofrece
 801 convierten a Vega-Lite en una opción ideal para el análisis y representación de los datos
 802 obtenidos. De hecho, todas las gráficas que se observan en el capítulo 6 se han elaborado
 803 haciendo uso de esta herramienta.

²<https://vega.github.io/vega-lite/>

³<https://json.org/json-es.html>

804 4.4. Diagramas de simulación

805 Representación

806 Una forma de visualizar el modelo de simulación que se va a construir utilizando la
 807 librería, es por medio de diagramas, de forma similar a como se muestra gráficamente en
 808 AnyLogic. En este programa, esencialmente, los bloques se colocan sobre un “plano” y se
 809 van conectando entre sí para construir el modelo, como puede verse en la imagen 4.2.

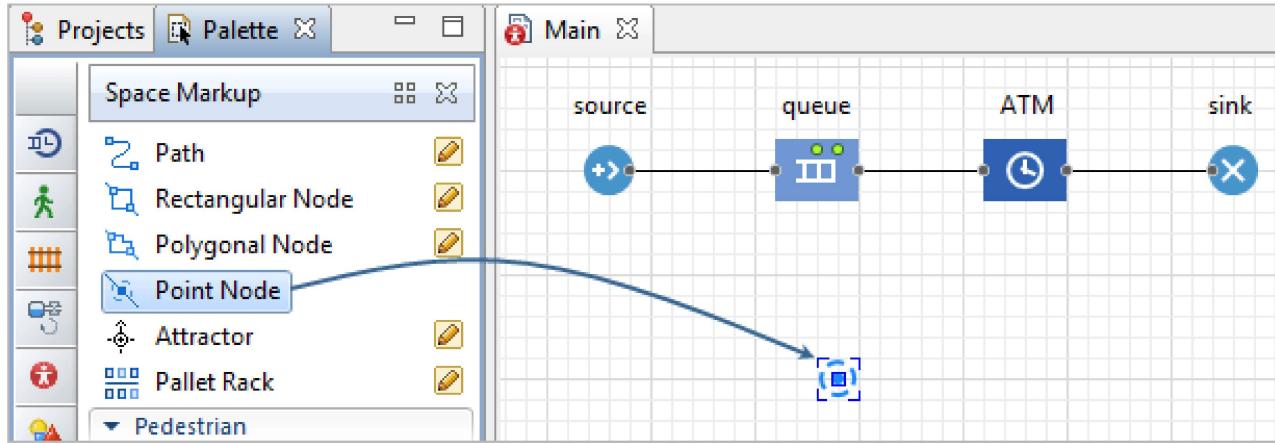


Figura 4.2: Interfaz de creación de diagramas de *AnyLogic*.

810 Para facilitar la comprensión del modelo a terceros, su explicación en este informe y,
 811 principalmente, tener alguna referencia a la hora de programarlo, se ha elaborado un
 812 diagrama de este tipo para la simulación de la terminal. Se han utilizado exactamente
 813 los mismos símbolos que utiliza *AnyLogic* para los distintos bloques, los cuales coinciden
 814 con los que se han explicado en el apartado 4.2. Están representados en la figura 4.3.
 815 Nótese que los nodos *Seize* y *Queue* se integran en un mismo elemento, ya que siempre
 816 se utilizan conjuntamente: para que a un contenedor se le asigne un recurso, primero
 817 debe hacer cola.

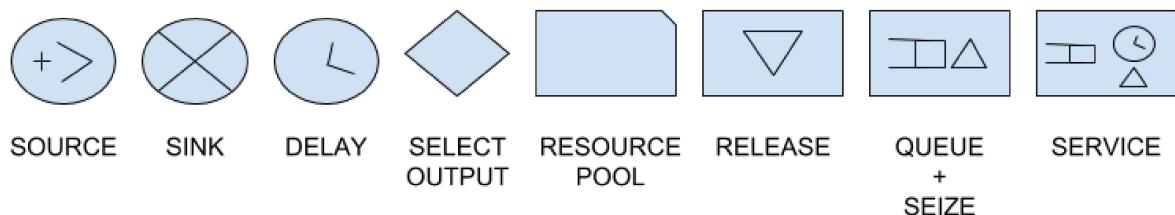


Figura 4.3: Representación de los bloques utilizados

818 Ahora bien, como se especifica en el apartado 3.2, es muy importante el concepto de
 819 trabajo en la simulación que nos ocupa, pues pueden realizarse múltiples actividades con
 820 los contenedores. En otras palabras, los contenedores son creados en distintos bloques y
 821 fluyen a través del modelo de una forma o de otra dependiendo del trabajo que se realice
 822 con ellos. Debido a esto, existen tres procedimientos bien diferenciados dentro de la
 823 simulación, que comparten únicamente recursos y bloque *Sink*.

824 Estos tres procedimientos se separan, para su más sencillo entendimiento, en tres
 825 diagramas diferentes. Así pues, aunque se refierece a los mismos pool de recursos, lo
 826 único que comparten es el nodo final que destruye los agentes, además de una fusión
 827 entre los trabajos de almacenamiento y recolocación para un caso determinado, que se
 828 explica más adelante en esta sección.

829 El pool de recursos de vehículos es compartido al 100 % por todos los nodos que hacen
 830 uso de este tipo de recursos. Sin embargo, en el caso de las grúas, no existe un único pool
 831 de recursos, sino que hay uno por cada bloque de contenedores. A continuación se verá
 832 que la asignación a distintos bloques viene dada por distintos caminos a los que redirige
 833 un *select output n*, siendo *n* igual al número de bloques.

834 Es importante saber que la programación del modelo no se ciñe estrechamente a lo
 835 que puede observarse en los diagramas. Por ejemplo, no existirán únicamente tres nodos
 836 *Source*, sino que habrá algunos más, ya que no solo depende del trabajo que se esté
 837 realizando sino también de qué ruta siguen los contenedores (si se marchan o llegan por
 838 tierra o por mar). Esto se explica con más detalle en el capítulo 5 de este documento.

839 **Diagrama de almacenamiento**

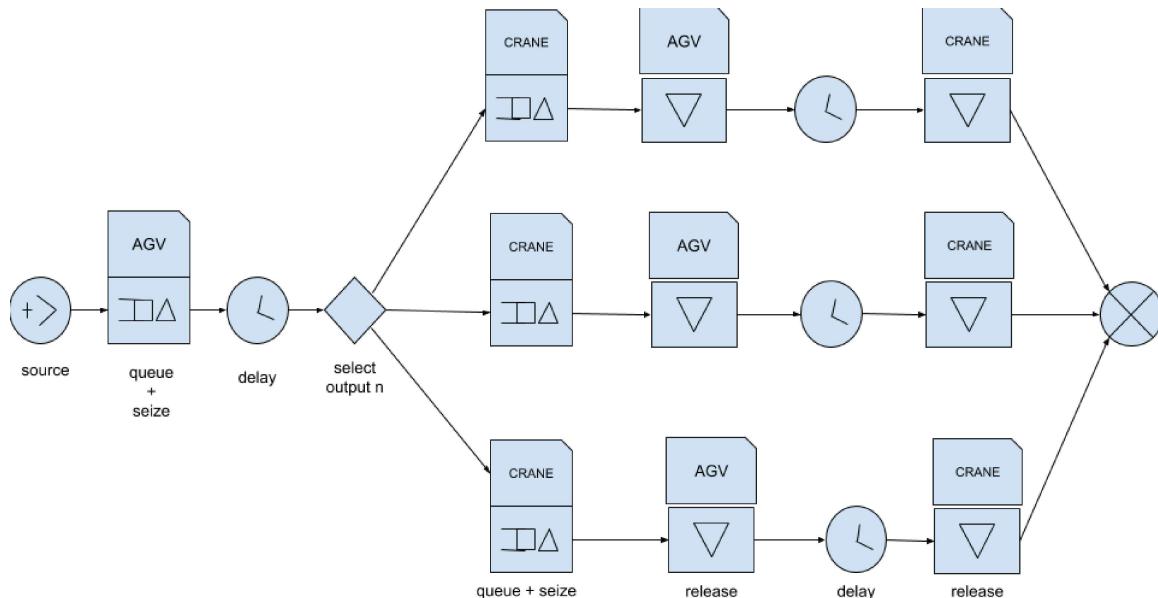


Figura 4.4: Diagrama de almacenamiento

840 El nodo fuente del diagrama de almacenamiento genera contenedores que llegan a
 841 la terminal y desean ser almacenados en uno de los bloques de contenedores. Como se
 842 ha mencionado, a la hora de programar no se trata de un único nodo, pero para una
 843 visualización más sencilla, se unifican todos en uno.

844 El camino que siguen los contenedores es el siguiente, representado por la figura 4.4:

- 845 1. Los contenedores solicitan que se les asigne un vehículo (pool de recursos de AGV),
 846 que les recoja y les lleve hacia el bloque de contenedores en el que van a ser
 847 almacenados. Esto se lleva a cabo en el nodo *seize*. Este nodo incluye una cola, en la
 848 que se retienen todos aquellos contenedores que han sido generados pero aún no se
 849 les ha asignado un vehículo. Un elemento solo puede salir de la cola cuando el *Seize*
 850 tiene recursos disponibles.

- 851 2. Una vez se les asigna un recurso vehículo, los contenedores son enviados al nodo
 852 *Delay*, que representa el tiempo necesario para que el vehículo los transporte al
 853 bloque de contenedores en el que van a ser almacenados. La llegada al *Delay*
 854 también supone el momento en el que se decide en qué bloque se almacena el
 855 contenedor (se aplica la política de asignación).
- 856 3. Cuando este tiempo finaliza, se llega a un bloque *select output n*. Como se puede ver,
 857 este redirige los contenedores hacia tres caminos posibles, los cuales son idénticos
 858 entre sí. En el diagrama aparecen tres caminos, pero en realidad serán tantos
 859 como bloques de contenedores haya en la terminal, ya que representan los caminos
 860 que siguen los contenedores cuando llegan al bloque que se les ha asignado. Así
 861 pues, este *select output* redirige a los contenedores hacia una salida en concreto
 862 dependiendo de qué bloque de destino se les haya asignado. Todos los *select output*
 863 *n* de todos los diagramas cumplen esta función.
- 864 4. Posteriormente, el contenedor llega a un bloque *Queue + Seize*. Una vez ha llegado
 865 al bloque de contenedores en el que va a ser almacenado, tiene que solicitar una
 866 grúa que le recoja y le coloque en la posición adecuada dentro del bloque. Nótese
 867 que el recurso vehículo no puede ser liberado hasta que un recurso grúa no le es
 868 asignado. En otras palabras, el vehículo tiene que esperar a que una grúa recoja el
 869 contenedor que está transportando.
- 870 5. Una vez se le ha asignado una grúa a los contenedores, el recurso vehículo puede
 871 ser liberado. Ese es el propósito del siguiente nodo *release*.
- 872 6. Al igual que ocurría después de asignar un recurso vehículo, es necesario repre-
 873 sentar, mediante un nodo *delay*, el tiempo necesario para que la grúa mueva el
 874 contenedor desde donde se libera el vehículo hasta su posición asignada dentro del
 875 bloque.
- 876 7. Una vez ese tiempo pasa, el contenedor ya se encuentra almacenado dentro de un
 877 bloque de contenedores. Se libera el recurso grúa en el *release* y el agente puede
 878 ser eliminado, puesto que el trabajo ha terminado.
- 879 8. Llega al *sink* y es destruido.

880 **Diagrama de retirada**

881 El trabajo de retirada, como podría esperarse, es básicamente un trabajo de almacena-
 882 miento en sentido contrario. Sin embargo, el diagrama difiere algo más, ya que al fluir en
 883 sentido contrario, cambian los momentos en los que se liberan u ocupan ciertos recursos
 884 (figura 4.5).

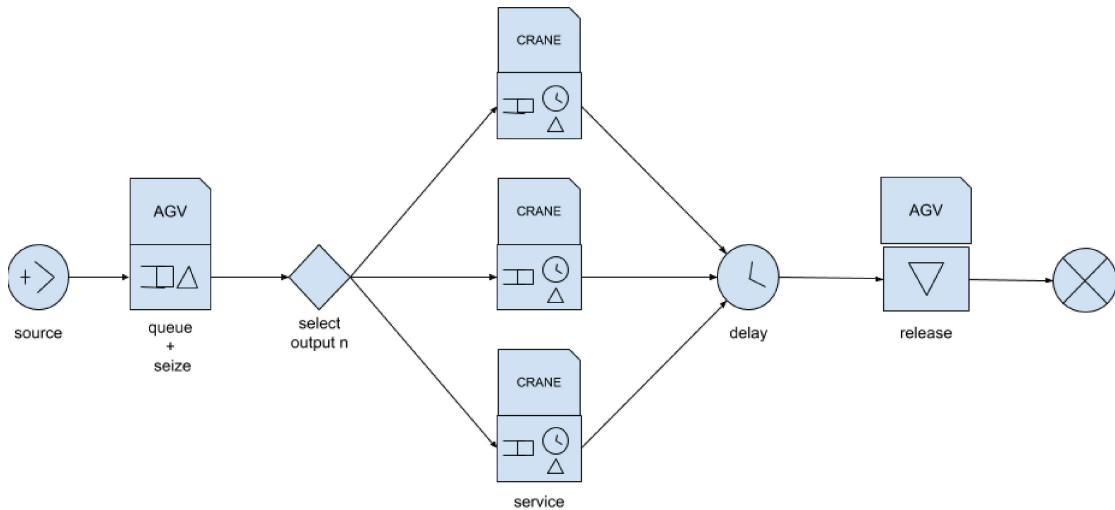


Figura 4.5: Diagrama de recogida de contenedores

- 885 1. Se generan contenedores de llegada en el nodo *source*.
- 886 2. Los contenedores llegan a un *seize* (previa cola) para ocupar un recurso vehículo. Aunque los contenedores de retirada son transportados por una grúa antes que por un vehículo, es necesario reservar el vehículo antes de reservar la grúa, ya que si no hay un vehículo asignado, la grúa no tendría dónde depositar el contenedor.
- 890 3. Una vez se les ha asignado un vehículo, llegan a un *select output n*. Al igual que en el diagrama de almacenamiento, el contenedor es redirigido a la salida que corresponde al bloque de contenedores en el que se encuentra almacenado.
- 893 4. Se aprovecha un servicio completo de grúa. Se le asigna una grúa, se retrasa el tiempo necesario para que la grúa lo agarre y deposite en su vehículo asignado, y se libera el recurso grúa.
- 896 5. Despues del servicio, todos los caminos convergen de nuevo en un *delay* que representa el tiempo que necesita el vehículo para sacar el contenedor de la terminal.
- 898 6. Una vez está fuera de la terminal, en el *release* se libera el recurso vehículo que se había ocupado al principio.
- 900 7. El trabajo ha terminado y el contenedor llega al *sink* para ser destruido.

901 **Diagrama de recolocación**

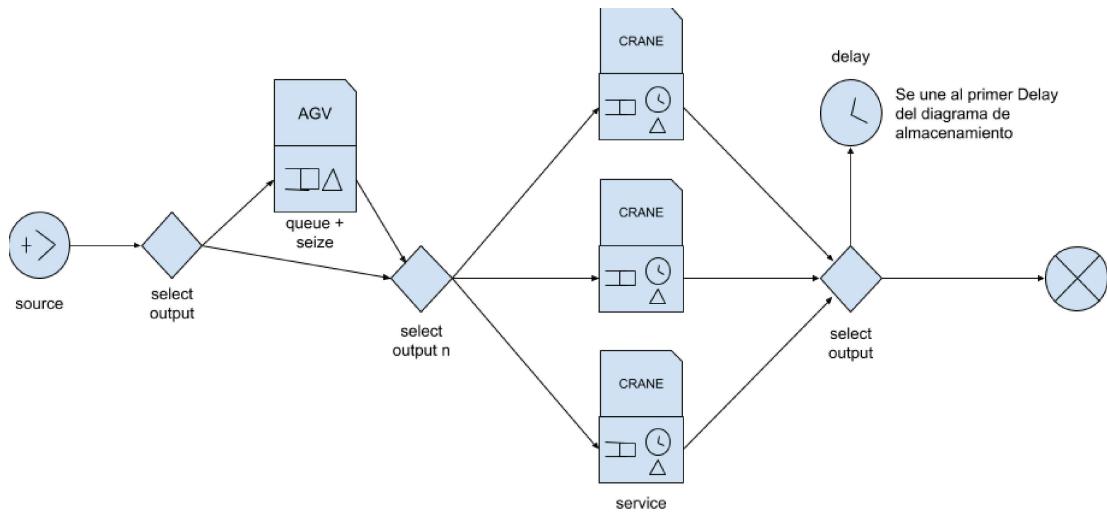


Figura 4.6: Diagrama de reubicación de contenedores

902 El diagrama de recolocación (figura 4.6) tiene varios caminos que pueden seguir los
 903 contenedores, dependiendo de si estos van a ser movidos a un bloque diferente o tan solo
 904 a una posición diferente dentro de su mismo bloque. También tiene la particularidad de
 905 que los contenedores podrían redirigirse al diagrama de almacenamiento.

- 906 1. El *source* genera contenedores de recolocación, que tienen un punto de origen y un
 907 punto de destino dentro del patio de contenedores.
- 908 2. Los nodos llegan a un *select output* donde se comprueba si el contenedor necesita
 909 o no un vehículo. Si va a ser movido a una posición diferente dentro del mismo
 910 contenedor, no necesita vehículo, pero si va a colocarse en un bloque diferente,
 911 entonces lo necesita.
 - 912 a) En caso afirmativo, el contenedor es redirigido a un *seize* similar al primer
 913 *seize* del diagrama de retirada: se necesita un vehículo para que la grúa tenga
 914 dónde depositar el contenedor al sacarlo del bloque.
 - 915 b) En caso negativo, llega al punto 3.
- 916 3. De nuevo un *select output n* que redirige los contenedores por un camino determina-
 917 do dependiendo del bloque en el que se encuentren almacenados en ese momento.
- 918 4. A continuación, el contenedor necesita que una grúa lo agarre, mueva y deposite, ya
 919 sea en su posición final o en un vehículo para que se lo lleve a un bloque diferente.
 920 Por ello, el contenedor se encuentra un *service* asociado al pool de recursos de
 921 grúas.
- 922 5. Los caminos confluyen en un *select output*. Este cumple una función similar al del
 923 punto 2: dependiendo de si el contenedor necesita vehículo o no, redirige por un
 924 camino u otro. Si no necesita vehículo, entonces ya ha sido depositado en su posición
 925 final y es redirigido al *sink*.

- 926 a) Si necesitaba vehículo (ya le fue asignado uno tras el primer *select output*),
927 entonces el vehículo debe redirigir el contenedor hacia su bloque de destino,
928 para que sea agarrado por una grúa y depositado en su lugar correspondiente.
929 Este procedimiento es parte del trabajo de almacenamiento; es exactamente el
930 mismo que se realiza a partir del primer *delay* del diagrama de almacenamiento.
931 Por ello, la salida *true* conecta con ese *delay*, ya que no tiene sentido crear dos
932 veces un camino idéntico.

933 Capítulo 5

934 Codificación del modelo

935 5.1. Consideraciones generales

936 En esta sección, se describe cómo se ha desarrollado el modelo. Como es lógico, se abor-
937 dan conceptos de programación que son desconocidos para aquellos no experimentados
938 con al menos un lenguaje de programación orientado a objetos.

939 La librería de simulación, como ya se ha comentado, proporciona clases que permiten
940 crear diagramas o modelos similares a los que permite realizar *AnyLogic*, pero por medio
941 de su programación directa en Java. Las clases principales son *BaseModelBuilder* y
942 *AgentBase*.

- 943 ■ La clase *BaseModelBuilder* es la clase de la que se debe heredar para poder construir
944 el modelo. Obliga a implementar un método *build*, que es donde se lleva a cabo la
945 especificación de la construcción. Debe especificarse un agente root, que va a ser la
946 raíz del diagrama, de tal forma que todos los demás agentes van a “colgar” de él.
- 947 ■ La clase *AgentBase* proporciona una plantilla vacía para la creación de clases agente
948 que se puedan necesitar en una simulación. Derivando de esta clase, se ha progra-
949 mado la jerarquía de clases *Container*, la jerarquía de clases *TerminalResource*, y
950 prácticamente todas las clases que representan entidades dentro de la simulación.
951 La mayoría de estas clases son descritas en esta sección.

952 A grandes rasgos, para hacer funcionar la simulación, se crea un objeto de una clase
953 derivada de *BaseModelBuilder*, y se asocia su agente root a un objeto de la clase *Simu-*
954 *latorEngine*, que es la que gestiona todos los eventos dentro de la simulación. Podría
955 decirse que es lo que la hace funcionar. En este caso, un objeto *TerminalModelBuilder*,
956 con un agente root del tipo *TerminalRootAgent* se asocia a un objeto *SimulatorEngine*. A
957 este último se le establece la unidad de tiempo en segundos.

958 La clase *TerminalRootAgent* es una clase básica que extiende a *AgentBase* sin demasia-
959 da funcionalidad extra, salvo por el hecho de que almacena los datos del resto de agentes
960 en formato JSON. Cuando la simulación termina, el método *toJSONFile()* lo escribe todo
961 en ficheros externos para su posterior análisis y uso en Vega-Lite. Además, según se
962 ejecuta la simulación, este agente va escribiendo todo lo que ocurre en un fichero.

963 5.1.1. Representación del espacio

964 La representación del espacio del interior de la terminal es fundamental para que
965 la simulación tenga utilidad real. Principalmente, se necesita saber a qué bloque de

966 contenedores se dirige un contenedor o de cuál sale. Asimismo, para los contenedores que
 967 llegan o que se marchan de la terminal, se tiene que determinar una forma de diferenciar
 968 a través de qué salida por tierra lo hacen, o a través de qué punto de atraque (por mar).
 969 Toda esta información es relevante para calcular los tiempos de desplazamiento de los
 970 vehículos. Por la misma razón, los propios vehículos tienen que ir registrando su posición.

971 Esta representación no es del todo sencilla. Se podría utilizar un plano en dos dimensiones,
 972 pero habría que calcular su extensión (cuántos valores de x y cuántos de y) en tiempo
 973 de ejecución, puesto que el número de puntos de atraque, bloques de contenedores y
 974 accesos por tierra son variables en cada ejecución.

975 Una opción más sencilla es utilizar un sistema propio que distinga las diferentes
 976 posiciones posibles en las diferentes áreas de la terminal. Como ya se sabe, existen tres
 977 áreas: área de tierra, área marítima y zona de enlace o patio. En el área de tierra, habrá
 978 un número determinado de accesos, al igual que en el área marítima. Y en la zona de
 979 enlace, habrá un número determinado de bloques de contenedores.

980 Al construir una clase que almacene el área y el número del acceso/bloque, se tiene un
 981 sistema sencillo y útil. Simplemente habría que calcular los desplazamientos posición a
 982 posición (número a número), y establecer un tiempo para cambio entre áreas. Lo único
 983 que haría falta calcular en tiempo de ejecución es qué bloque de contenedores conecta
 984 con cada uno de los accesos. Así, se define la clase *TerminalPosition*, que almacena ambas
 985 variables en un único objeto.

986 El diagrama de la figura 5.1 trata de representar cómo se gestionan internamente las
 987 posiciones, para una terminal con cinco bloques de contenedores, tres puntos de atraque
 988 para barcos y tres accesos terrestres.

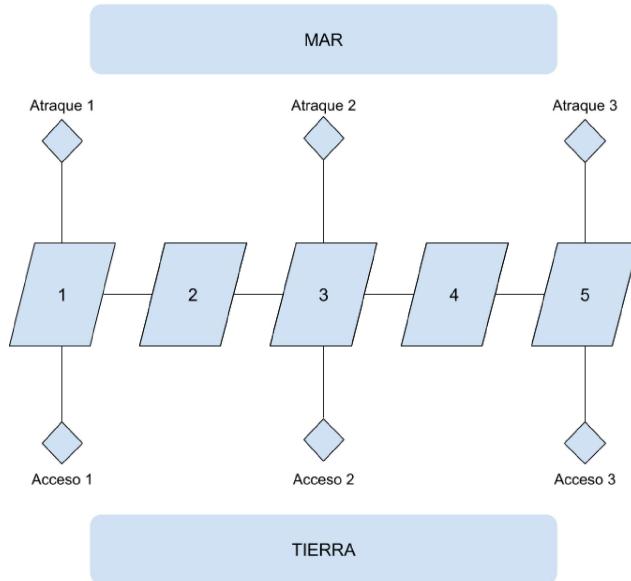


Figura 5.1: Representación interna del espacio de la terminal

989 Dentro de los bloques de contenedores también hay que diferenciar las distintas
 990 posiciones. Esto es fundamental para saber entre qué dos puntos se mueve una grúa y así
 991 calcular el tiempo estimado que tarda. Esta representación interna es bastante simple: se
 992 hace por bahía, pila y altura, a modo de espacio de coordenadas en tres dimensiones (x, y,
 993 z). Así, se crea la clase *Position* para almacenar los tres valores en un solo objeto.

994 5.1.2. Gestión del tiempo

995 Con respecto al tiempo, como ya se señala en la sección 2.4, es una buena práctica
 996 simular algo de tiempo residual antes y después del tiempo de simulación especificado,
 997 para tratar de acercar la simulación a un estado estacionario.

998 Así, se simula siempre tres veces el tiempo establecido como parámetro, y se utiliza
 999 la clase *TimeManager* para gestionarlo correctamente. Por ejemplo, esta clase permite
 1000 comprobar si un instante determinado de la simulación es parte del tiempo simulado del
 1001 que interesa extraer datos (segundo tercio del tiempo).

1002 5.2. Agentes

1003 Todas las clases que se describen en este apartado derivan de *AgentBase*, la plantilla
 1004 para crear agentes de forma sencilla. Además, almacenan gran cantidad de información
 1005 útil que el agente root termina exportando a ficheros externos para su posterior análisis.
 1006 Esta información depende del tipo de agente e incluye, por ejemplo, los tiempos de
 1007 creación y final, tiempos de espera, tiempos de desplazamiento, etc.

1008 5.2.1. La jerarquía de clases *Container*

1009 Los agentes que “fluyen” a través de los nodos son los contenedores. Los diagramas
 1010 vistos en el apartado anterior representan su ciclo de vida. La clase *Container* surge con
 1011 el objetivo de representar los contenedores dentro de la simulación. Tiene una serie de
 1012 variables internas que permiten su gestión por el resto de agentes. Algunas son:

- 1013 ■ Ruta: instancia del tipo enumerado *ContainerRoute*, que indica si su ruta de entrada
 1014 o salida es por tierra o por mar.
- 1015 ■ Origen y destino: guarda el número del bloque de contenedores del que procede y
 1016 el número del bloque de contenedores al que se dirige.
- 1017 ■ Posición: guarda y actualiza periódicamente su posición actual, así como la posición
 1018 de la que procede dentro de su bloque de origen y la posición a la que se dirige
 1019 dentro de su bloque de destino.
- 1020 ■ Punto de acceso a la terminal (*throughLink*). Por ejemplo, si hay seis puntos de
 1021 atraque para barcos, el punto a través del cual accedió a la terminal o a través del
 1022 cual se dispone a marcharse. Lo mismo ocurre con los accesos por tierra.

1023 En algunos casos, la gestión de variables podría complicarse. Por ejemplo, para los
 1024 contenedores que llegan, interesa medir el tiempo que pierden haciendo cola para que
 1025 les sea asignado un vehículo, puesto que se trata del tiempo que se hace esperar a un
 1026 camión o a un barco antes de que pueda desentenderse de ese contenedor.

1027 Por esto último, se tomó la decisión de crear tres clases derivadas de *Container*: *Arri-*
 1028 *vingContainer* (llegada), *OutgoingContainer* (salida) y *RelocatingContainer* (reubicación).
 1029 Estas clases aportan poca funcionalidad extra al comparar con su clase madre, pero sí
 1030 almacenan la información que es relativa únicamente a esos contenedores.

1031 Los diferentes bloques *Source* que forman parte del modelo, generarán diferentes
 1032 tipos de contenedores en función del diagrama al que pertenezcan. Así, aquellos *Source*

que se conecten al primer *Seize* del diagrama de almacenamiento, generarán agentes de la clase *ArrivingContainer*. Esto se puede especificar fácilmente en el constructor de la clase *Source*.

El código para la inicialización de contenedores puede observarse en el apéndice B.1.

5.2.2. La jerarquía de clases *TerminalResource*

Ya se ha aclarado que, en simulación basada en agentes, prácticamente todos los elementos de la simulación son agentes, no sólo aquellas entidades que fluyen por el modelo. Por tanto, los recursos también serán tipos de agente. En este caso, el modelo necesita de dos tipos de recursos: grúas (clase *Crane*) y vehículos (clase *AGVehicle*).

Estas clases comparten gran cantidad de variables y métodos, por lo que derivan de una clase abstracta *TerminalResource*. Dentro de las variables que almacena, destacan los tiempos de uso, la posición, o el tiempo fuera de servicio.

Cada objeto *ResourcePool* (conjunto de recursos) que se crea en el *ModelBuilder* tiene asignados como recursos instancias de una de las clases hijas de *TerminalResource*. Ambas clases están también muy relacionadas con los tipos enumerados *DelayPolicy* y *AllocationPolicy*, las políticas en los retrasos y en la asignación de contenedores a bloques, que representan dos de los parámetros más importantes de la simulación en general.

5.2.3. La clase *ContainerBlock*

Esta clase derivada de *AgentBase* representa los bloques de contenedores de la terminal, de los cuales se viene hablando desde la definición del problema. Se generan tantas instancias de esta clase como bloques de contenedores haya en la terminal, lo cual es un parámetro del problema.

Esta clase es necesaria para mantener un recuento de los contenedores que quedan almacenados, así como del número de contenedores que permanecen en cola para ser almacenados o extraídos. Aparte de eso, mantiene una referencia al recurso *Crane* asociado y, como es lógico, el número de bahías y número de pilas por bahía y su altura.

También es una clase interesante para posibles mejoras futuras. De momento, el modelo otorga posiciones aleatorias a los contenedores dentro de los bloques, pero esto podría mejorarse en el futuro, y para ello esta clase sería fundamental.

5.3. Principales políticas

5.3.1. Asignación de contenedores a bloques

Las políticas de asignación de contenedores a bloques son uno de los aspectos fundamentales a la hora de medir la eficiencia de las operaciones de la terminal. En la programación, se trata de una serie de funciones o métodos que reciben al agente contenedor que se quiere almacenar y devuelven el número del bloque que se les asigna. Estas funciones o métodos se definen en el tipo enumerado *AllocationPolicy*.

Cada instancia del *enum* tiene un atributo llamado *function*, de tipo *Function*¹, que define el cálculo del bloque de destino. Así, tenemos cinco instancias del *enum*, cada una con un *Function* que recibe un *Agent* y devuelve un *Boolean*:

¹<https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html>

- 1072 ■ **RANDOM**. Se trata de una política de asignación aleatoria.
- 1073 ■ **SMALLEST_QUEUE_BLOCK**. Asigna al bloque de contenedores que tenga la me-
1074 nor cola, es decir, aquel en el que se espera que el contenedor podrá ser atendido
1075 antes por la grúa.
- 1076 ■ **LAZIEST**. Asigna al bloque de contenedores cuya grúa haya realizado un menor
1077 trabajo.
- 1078 ■ **EMPTIEST**. Redirige el contenedor al bloque más vacío.
- 1079 ■ **CLOSEST_BLOCK**. Redirige al contenedor más cercano. Cada acceso a la terminal
1080 tiene un bloque de contenedores que se encuentra a menor distancia (consul-
1081 tar figura 5.1). Para la recolocación de contenedores, se aplica la política **SMA-**
1082 **LLEST_QUEUE_BLOCK** de entre los dos bloques contiguos.

1083 Estos métodos, una vez le asignan un bloque de destino a un contenedor, también le
1084 asignan una posición dentro del mismo (consultando al objeto *ContainerBlock* corres-
1085 pondiente), e incrementan su cola en 1. El código al completo puede observarse en el
1086 apéndice B.4.

1087 Cabe mencionar que se ha creado otros dos *enum*, *CollectionPolicy* y *RelocationPolicy*,
1088 que simplemente le asignan un bloque de origen a los *OutgoingContainer* y *Relocating-*
1089 *Container*. Funcionan de forma similar a *AllocationPolicy*, pero sus métodos definen
1090 el contenedor y la posición de origen, no de destino. Además, tan sólo tienen una op-
1091 ción **RANDOM_ORIGIN** y **RANDOM_RELOCATION**, que siguen la misma filosofía que
1092 **RANDOM** (asignación aleatoria). **RANDOM_RELOCATION** también determina si el *Re-*
1093 *locatingContainer* necesita un vehículo o no, comprobando si el bloque de origen y destino
1094 son el mismo.

1095 5.3.2. Retrasos

1096 Las políticas de retrasos son una serie de métodos que calculan el tiempo estimado de
1097 desplazamiento de los recursos, tanto las grúas como los vehículos (AGV). Se definen de
1098 forma similar a las políticas de asignación de contenedores a bloques, aunque en este
1099 caso, se dividen entre dos *enum*: uno que contiene los métodos que calculan los tiempos
1100 de desplazamiento transportando un contenedor, y otro que contiene los métodos que
1101 calculan los tiempos de desplazamiento para ir a buscar un contenedor que aún no ha
1102 sido recogido. Este último tipo es muy importante: supóngase que un vehículo deja un
1103 contenedor en el bloque 1 e, inmediatamente, se le asigna un contenedor que espera en
1104 el bloque 8 para ser transportado fuera de la terminal. El tiempo que el vehículo pierde
1105 transportándose del bloque 1 al 8 es muy relevante para el resultado final.

1106 A la hora de realizar el cálculo de los tiempos, se ha hecho uso de distribuciones de
1107 probabilidad triangulares. Se trata de una distribución de probabilidad continua que tiene
1108 un valor máximo posible, un valor mínimo posible, y una moda (valor más probable o más
1109 repetido). La densidad de probabilidad es cero para los extremos (máximo y mínimo), e
1110 igual entre cada extremo y la moda. Así, se establece unos tiempos máximo y mínimo
1111 necesarios para realizar un movimiento, junto con el valor más habitual para ese tipo de
1112 movimiento. Con esos valores, y haciendo uso de la clase *KaizenProbabilityDistribution*
1113 de la librería de simulación, se calculan los tiempos.

1114 ***DelayPolicy***

1115 El tipo enumerado *DelayPolicy* se define exactamente igual que las políticas de lo-
 1116 calización: cada instancia es un *Function* que, en este caso, devuelve un valor decimal
 1117 que representa el tiempo de retraso. Estas políticas de retraso deben ser aplicadas a los
 1118 contenedores en cada nodo *Delay* mediante su componente *delayTime*. El valor returnedo
 1119 por esta componente define el tiempo que el *Delay* esperará antes de enviar el contenedor
 1120 por su puerto de salida hacia el siguiente nodo.

1121 Adicionalmente tiene definidas las constantes necesarias para el uso de la distribución
 1122 de probabilidad triangular (máximo, mínimo y moda). Con esta se calculan los tiempos que
 1123 necesita una grúa para moverse una posición y un vehículo para moverse una distancia
 1124 equivalente a un bloque. Además de eso, define un extra de altura en el que necesita
 1125 posicionar la grúa para moverse a una pila distinta sin impactar con ninguna otra, entre
 1126 otras constantes necesarias para realizar los cálculos.

- 1127 ■ **CRANE_DELAY_TRIANGULAR.** Calcula el tiempo que necesita una grúa para
 1128 transportar un contenedor entre dos posiciones del bloque de contenedores.
- 1129 ■ **AGV_DELAY_TRIANGULAR.** Se estima el tiempo necesario para que un vehículo
 1130 transporte un contenedor hasta su destino. El cálculo se hace bloque por bloque
 1131 (consultar figura 5.1), acumulando el tiempo en una variable.

1132 ***DelayInSeize***

1133 Se trata de un *enum* interno al anterior, que aprovecha sus métodos para calcular
 1134 los tiempos que necesitan los recursos para llegar hasta los contenedores que los han
 1135 reservado. En este caso, se definen por medio de objetos *BiFunction*² (de la librería nativa
 1136 de Java), que reciben el agente contenedor y el recurso y devuelven el tiempo de retraso.
 1137 Se establecen como componente *delayInSeize* de los objetos *Seize*, lo cual los aplica a un
 1138 recurso cuando es reservado pero todavía no ha sido completamente ocupado. El retraso
 1139 a la hora de ocupar el recurso por completo es el valor devuelto por el *BiFunction*.

1140 Las dos instancias son **CRANE_BEFORE_PICK**, para grúas, y **AGV_BEFORE_PICK**,
 1141 para vehículos.

1142 **5.4. Generación de contenedores**

1143 La generación de contenedores se da en los diferentes nodos *Source* que pueden
 1144 observarse en los tres diagramas. Aquellos asociados al diagrama de almacenamiento
 1145 generan agentes de tipo *ArrivingContainer*; los asociados al diagrama de recogida generan
 1146 agentes de tipo *OutgoingContainer*; y los asociados al diagrama de reubicación generan
 1147 agentes de tipo *RelocatingContainer*.

1148 Ahora bien, como se menciona en la sección 4.4 (aquella en la que se presentan los
 1149 diagramas), en realidad no se tiene necesariamente un único nodo *Source*. De hecho, el
 1150 único diagrama que no cumpliría esta regla es el de reubicación de contenedores, para
 1151 el cual hay un único *Source* permanente que genera agentes de forma exponencial.

1152 Para la llegada y recogida de contenedores, se tienen que contemplar las diferencias
 1153 entre la llegada o salida por mar y la llegada o salida por tierra. El caso de tierra es

²<https://docs.oracle.com/javase/8/docs/api/java/util/function/BiFunction.html>

1154 bastante simple: un nodo *Source* para generar contenedores que llegan y uno para
 1155 generar contenedores que se marchan. Ambos generan agentes de forma exponencial.

1156 El caso de mar es algo más complicado, principalmente por una razón: los barcos
 1157 no traen o se llevan un único contenedor de la terminal, como los camiones de tierra,
 1158 sino que normalmente traen decenas y se llevan decenas de una sola vez. Por ello, la
 1159 implementación es bastante diferente. Esta puede observarse en el apéndice B.2.

1160 **5.4.1. Enlace marítimo**

1161 En un principio, mientras se planteaba el diseño que iba a tener el modelo, el enlace
 1162 marítimo no iba a ser diferente del enlace por tierra. De hecho, al principio se utilizaban
 1163 los mismos nodos *Source*, con la excepción de que a los contenedores generados se les
 1164 asignaba una ruta aleatoria, por tierra o por mar. Sin embargo, en una terminal real, un
 1165 barco solicita el almacenaje y la retirada de múltiples contenedores de una sola vez. Esto
 1166 claramente afecta a la carga de trabajo de los recursos de un momento a otro, por lo que
 1167 era fundamental que en el modelo funcionase de la misma forma.

1168 La principal complicación en la implementación era lograr que, al realizarse una
 1169 petición de retirada y de almacenamiento de contenedores, se solicitases varios de una
 1170 sola vez, y luego no hubiese más peticiones hasta que se procesaran los anteriores. Esto
 1171 representa un barco que llega a la terminal, solicita llevarse X contenedores y dejar Y
 1172 contenedores, y ningún otro barco atraca en el mismo punto de atraque hasta que este ha
 1173 terminado sus operaciones y se ha marchado.

1174 Así, se creó una nueva clase agente *Berth* para representar cada punto de atraque.
 1175 Estos agentes se conectan como si fuesen los *Source* en sí, con la salvedad de que se
 1176 conectan tanto al diagrama de almacenamiento como al de retirada. Internamente, tienen
 1177 dos componentes *Source*, uno para contenedores de retirada y otro para contenedores de
 1178 llegada.

1179 Programar la generación de varios contenedores simultáneamente era sencillo. La clase
 1180 *Source* de la librería tiene una componente *agentsPerGeneration*, un objeto *Function* que
 1181 devuelve un número entero que va a indicar cuántos agentes deben generarse en cada
 1182 generación.

1183 Cuando ocurre una generación, ambos objetos *Source* son destruidos, guardando
 1184 el número de agentes generados en una variable. Cuando uno de los contenedores de
 1185 llegada ocupa un vehículo, esta variable se decrementa (el barco puede olvidarse de él).
 1186 Ocurre lo mismo cuando un contenedor de retirada llega al *Sink* (ya ha sido extraído y
 1187 guardado en el barco). Cuando el valor de la variable llega a cero, se crean de nuevo
 1188 los *Source* y se conectan en tiempo de ejecución. De esta forma, se representa el hecho
 1189 de que no llega ningún barco hasta que no se marcha el anterior. La forma en la que se
 1190 crean dinámicamente los objetos *Source* puede observarse en el apéndice B.2.

1191 Como es lógico, hay un objeto *Berth* en el modelo por cada punto de atraque, gestionan-
 1192 dos conjuntamente por una sencilla clase agente *SeaLink*. El número de atraques es un
 1193 parámetro de ejecución.

1194 **5.4.2. Frecuencia de generación**

1195 El número de contenedores a generar es un parámetro de la simulación. El número
 1196 indicado se reparte entre los distintos *Source* según unas proporciones prestablecidas.
 1197 Así, estos nodos generan agentes de forma exponencial, con un ritmo de generación que

1198 permite seguir generando hasta el final del tiempo de simulación.

1199 En el caso del enlace marítimo, este sistema funciona de forma diferente, ya que
 1200 los *Source* se crean y se destruyen constantemente. Los objetos *Berth* mantienen una
 1201 variable que indica el número total de contenedores que tienen que generar y el número
 1202 que ya han generado.

1203 Dependiendo de la calidad del modelo, los contenedores asociados a los barcos podrían
 1204 procesarse muy rápido o muy despacio. Si se procesan muy rápido, los *Berth* podrían
 1205 generar todos los contenedores que les corresponden en muy poco tiempo de simulación.
 1206 Para evitar mucho tiempo de simulación sin la llegada de ningún barco, se ha hecho que
 1207 los *Berth*, cuando llegan a esa situación, esperen aproximadamente un día y medio y
 1208 reseteen la cuenta de contenedores generados.

1209 Esto implica que, cuanto mejor procese el modelo los contenedores que llegan o que se
 1210 van por mar, más contenedores se irán generando. El parámetro inicial de contenedores
 1211 a generar, por tanto, indicará sólo un número aproximado. Si el modelo es eficiente,
 1212 generará más; si no lo es, generará menos.

1213 **5.5. Periodos de indisponibilidad de recursos**

1214 **5.5.1. Indisponibilidad planificada**

1215 Los períodos de indisponibilidad planificados, como su propio nombre indica, son
 1216 conocidos de antemano e, incluso, son algo flexibles. De esta forma, al inicio de la
 1217 simulación se pueden planificar los tiempos aproximados que cada recurso va a estar
 1218 fuera de servicio, en base al ratio de disponibilidad que tengan.

1219 Cada recurso contiene una variable relacionada con la disponibilidad; una instancia del
 1220 enum *ResourceAvailability* que define el porcentaje del tiempo total de simulación que
 1221 el recurso está disponible. Utilizando este valor, se calcula el tiempo total de simulación
 1222 que el recurso debe estar fuera de servicio y se distribuye de forma aleatoria. Una vez se
 1223 tienen estos tiempos, se programan una serie de eventos que dejan a los recursos fuera
 1224 de servicio, programando a su vez un nuevo evento que los devolverá al servicio pasado
 1225 el tiempo necesario. Todo esto se realiza antes de comenzar con la simulación.

1226 Existe una alta probabilidad de que la ejecución de estos eventos coincida con períodos
 1227 en los que los recursos están ocupados. Para estos casos, se asigna una variable al
 1228 recurso que indica su tiempo pendiente fuera de servicio. Se comprobará la existencia de
 1229 esta variable en una función asignada a cada componente *delayInRelease* de los objetos
 1230 *Release*, un objeto *Function* que se aplica al recurso para determinar el tiempo desde que
 1231 el contenedor deja de usar el recurso hasta que éste vuelve a estar de nuevo disponible.
 1232 Si el recurso tiene tiempo fuera de servicio pendiente, se utilizará ese valor como retraso.

1233 En el apéndice B.3, se pueden observar los principales métodos que influyen en la
 1234 planificación y gestión de los períodos de indisponibilidad planificados.

1235 **5.5.2. Indisponibilidad no planificada**

1236 Este caso se refiere a la probabilidad de fallo de los recursos. Una variable instancia
 1237 del enum *ResourceFallibility* define la probabilidad de fallo de un recurso en cada
 1238 desplazamiento.

1239 Así, en cada aplicación de los ya explicados *DelayPolicy* y *DelayInSeize*, se determina

1240 si hay fallo o no utilizando la probabilidad que tenga el recurso. Si hay fallo, se extiende
 1241 el valor devuelto por estos componentes con un tiempo de reparación, extraído de una
 1242 distribución de probabilidad triangular.

1243 5.6. Prioridad de contenedores

1244 La implementación de la prioridad de contenedores implica la ordenación de las colas
 1245 en función de la prioridad que tengan los contenedores, la cual se define por medio
 1246 de una variable en *Container* que almacena una instancia de *ContainerPriority*. Los
 1247 objetos *Queue*, por defecto, son colas FIFO (*First In First Out*), pero se puede modificar
 1248 este comportamiento para que ordenen sus agentes utilizando una implementación de
 1249 *Comparator*.

1250 Ahora bien, la librería tiene las colas implementadas de tal manera que permiten
 1251 continuar a los agentes hacia el siguiente nodo siempre que el siguiente nodo lo permita.
 1252 Si no se realizan algunos ajustes, los contenedores pasarían por la cola sin espera alguna,
 1253 llegando de forma inmediata al *Seize* que la sigue.

1254 Por esto, cada vez que un recurso es ocupado o liberado, se llama a sus métodos
 1255 *seized()* y *released()*, respectivamente. En estos se comprueba si el *pool* de recursos al
 1256 que pertenecen tiene recursos disponibles.

- 1257 ■ Si no los hay, se desactiva el puerto de entrada de todos los objetos *Seize* que
 1258 utilicen dicho *pool* de recursos.
- 1259 ■ Si hay recursos disponibles, se lleva a cabo el proceso contrario, activando los
 1260 puertos de entrada de los *Seize*.

1261 De esta manera, los contenedores permanecen en las colas mientras no haya recursos
 1262 disponibles en el *Seize* siguiente. Según llegan contenedores nuevos, la cola se reordena
 1263 en función de la prioridad que tengan asignada.

1264 En un principio, se barajó la posibilidad de implementar la configuración de conte-
 1265 nedores con prioridad extrema. Esta prioridad se utilizaría para representar mercancía
 1266 sensible. Por ejemplo, material radiactivo o material quirúrgico. La terminal detendría
 1267 todas sus operaciones o parte de las mismas para ocuparse de tratar estos contenedores.

1268 Sin embargo, la librería de simulación no contemplaba la posibilidad de “pausar”
 1269 ciertos nodos o ciertos agentes, por lo que habría sido bastante costoso implementar este
 1270 sistema. Además, la llegada de este tipo de material sensible es extremadamente poco
 1271 común (un contenedor al día como mucho). Esto, sumado a que, como se explica en la
 1272 sección 6.7, la introducción de prioridades no tiene un efecto real sobre la eficiencia final,
 1273 hizo que la idea fuese descartada y el sistema no llegase a implementarse.

1274 Capítulo 6

1275 Experimentación

1276 6.1. Consideraciones previas

1277 En este capítulo, se resumen las conclusiones obtenidas a partir de una experimenta-
1278 ción realizada con la librería final. El objetivo de esta experimentación es demostrar la
1279 validez y la utilidad del trabajo desarrollado.

1280 Ejecutar directamente todas las posibles combinaciones de parámetros que la librería
1281 permite, arrojaría como resultado los datos de miles de ejecuciones. En su lugar, se ha
1282 seguido un procedimiento por fases en el que se van seleccionando los parámetros que
1283 otorgan mejores resultados en cada fase, para aplicarlos en la fase siguiente, a modo de
1284 algoritmo evolutivo.

1285 Es importante tener en cuenta las probables variaciones en los resultados obtenidos
1286 en varias ejecuciones con los mismos parámetros. Dos ejecuciones en exactamente las
1287 mismas condiciones podrían arrojar resultados considerablemente diferentes. Con el fin
1288 de obtener un promedio con resultados fiables, cada ejecución se realizará un mínimo de
1289 10 veces. Con los datos de estas 10 ejecuciones, se calculará el promedio de las variables
1290 más interesantes, como pueden ser el tiempo de trabajo de los recursos o el número de
1291 contenedores totales procesados.

1292 Todas las ejecuciones de la simulación se han realizado para una semana, con las
1293 consiguientes semanas anterior y posterior residuales para aproximarse a un estado
1294 estacionario (sección 2.4).

1295 6.2. Diseño de la terminal

1296 La librería permite especificar hasta cierto punto el diseño de la terminal. Por ejemplo,
1297 estableciendo el número de accesos por tierra o el número de atraques. Probar variaciones
1298 de estos valores podría aumentar considerablemente el número de datos a interpretar,
1299 pero también podría apreciarse una clara mejoría en los resultados obtenidos para ciertos
1300 casos.

1301 Sin embargo, este enfoque no es del todo aplicable en situaciones reales. El diseño de
1302 una terminal es un procedimiento que se desarrolla a lo largo de dos fases [13]:

- 1303 1. Análisis de, primero, la necesidad económica de establecer una terminal y, segundo,
1304 las infraestructuras necesarias. La segunda parte implica un análisis de las vías
1305 de comunicación, que en muchos casos dependen directamente de la orografía del
1306 terreno.

1307 2. Definir en más detalle ciertas necesidades estructurales y políticas internas en
 1308 función de ciertos parámetros y su proyección en el tiempo.

1309 La primera de las fases, por tanto, requiere fundamentalmente de información que
 1310 la librería desarrollada no contempla. Esta está pensada más bien para ser útil en la
 1311 segunda fase: recibe unos parámetros de diseño prestablecidos y simula la terminal para
 1312 determinar las mejores políticas internas, cantidad de recursos, etc.

1313 Por tanto, a la hora de realizar la experimentación, se ha supuesto una terminal con una
 1314 infraestructura predefinida, como si se estuviese simulando para una terminal marítima
 1315 real, ya existente. Dicha infraestructura sería la siguiente:

- 1316 ■ Número de atraques: 4.
- 1317 ■ Número de accesos por tierra: 3.
- 1318 ■ Número de bahías por bloque de contenedores: 5.
- 1319 ■ Número de pilas por bahía: 48.
- 1320 ■ Altura de cada pila: 8 contenedores.
- 1321 ■ El número aproximado de contenedores por semana que llegan a la terminal, se
 marchan o son reubicados en ella es de 5000.
- 1322 ■ Los valores utilizados para definir las distribuciones de probabilidad triangulares
 de las que se obtienen los tiempos de desplazamiento de grúas y vehículos se
 han definido de forma manual. No se tiene ninguna referencia real acerca de la
 velocidad de movimiento de estos recursos. Si se tuviese, podría aplicarse sin
 mayores problemas.

1328 **6.3. Políticas de asignación**

1329 La política de asignación de contenedores a bloques es posiblemente uno de los
 1330 parámetros más importantes y que más afectan a la eficiencia final del sistema. Teniendo
 1331 un enfoque en el que se van seleccionando los mejores parámetros dentro de un grupo
 1332 antes de proceder a seleccionar los mejores del siguiente, interesa determinar desde un
 1333 principio la política que mejores resultados otorgue. Esto garantiza los mejores resultados
 1334 posibles en experimentaciones posteriores, como por ejemplo la del número de recursos
 1335 (sección 6.4).

1336 Ahora bien, al tratarse del primer experimento, se debe determinar el valor que se
 1337 otorga a aquellos parámetros para los que no se experimentará hasta fases posteriores.
 1338 En estos casos, para las primeras fases, conviene establecer unos valores aceptables y
 1339 asumibles. Demasiados recursos podrían facilitar demasiado el trabajo y no se apreciarían
 1340 diferencias claras entre las diferentes políticas. Fijar muy pocos recursos podría causar
 1341 el mismo efecto al sobresaturar el tráfico en la terminal.

1342 En el apartado 6.4, el conjunto de posibilidades para el número de vehículos es de 15,
 1343 20, 25 y 30. El conjunto de posibilidades para el número de grúas es de 5, 10 y 15. En
 1344 este apartado se tomarán valores intermedios: 25 vehículos y 10 grúas. Los resultados
 1345 obtenidos, ordenados de mejor a peor, se muestran en la tabla 6.1.

Política	Contenedores procesados	Tiempos medios (s)		
		Contenedor	Trabajo grúas	Trabajo vehículos
Aleatoria	9084	7630	297513	391066
Menor cola	8886	10035	299550	431581
Grúa con menos trabajo	8287	14510	283632	484625
Bloque más vacío	8287	20874	284916	554310
Bloque más cercano	6369	63640	227587	559625

Tabla 6.1: Resultados de la experimentación con políticas de asignación

Como se puede observar, la mejoría de uno de los parámetros suele venir acompañada por la mejoría del resto. Si con una política se procesan más contenedores, el tiempo medio que permanecen estos en el sistema también es menor, así como el trabajo que tienen que realizar los vehículos. Esto indica directamente que la política es capaz de reducir el tráfico en la terminal.

La excepción a esto se encuentra en el trabajo realizado por las grúas. En el caso de esta variable, la relación funciona de forma casi inversa: cuanto menos tráfico en la terminal, más trabajo realizan. Esto puede deberse a dos factores principales:

- Baja eficiencia. Que se procesen menos contenedores implica que las grúas van a manejar menos contenedores. Por lo tanto, menos trabajo.
- Vehículos excesivamente ocupados. La mayoría de contenedores pasa por al menos un vehículo durante su ciclo de vida, y además siempre trata de ocuparlo antes de ocupar una grúa. Se podría decir que el trabajo de las grúas está supeditado al de los vehículos. Si los vehículos están saturados, tardan más tiempo en ser asignados a contenedores o en transportar a los mismos, retrasando las tareas de las grúas. Esto reduce el trabajo que terminan realizando éstas.

En cualquier caso, las diferencias en el trabajo realizado por las grúas siempre es menor a un día. En el caso de los vehículos, en cambio, es de más de un día. Y aún más importantes son las diferencias en el tiempo que permanecen los contenedores en el sistema: casi un día entero más. Todo esto teniendo en cuenta considerables diferencias en el número total de contenedores procesados.

La ventaja que tienen las cuatro primeras políticas con respecto a la que asigna al bloque más cercano es muy clara. El bloque más cercano a, por ejemplo, un atraque en concreto, es siempre el mismo. Por esto, los contenedores que llegan de un mismo barco se envían repetidamente al mismo bloque de contenedores, saturando la grúa y aumentando su cola, mientras otros bloques de contenedores quedan desaprovechados. En los siguientes gráficos puede observarse la gran descompensación en el uso de las grúas utilizando la política del bloque más cercano (figura 6.1) con respecto a cuando se hace uso de la política aleatoria (figura 6.2).

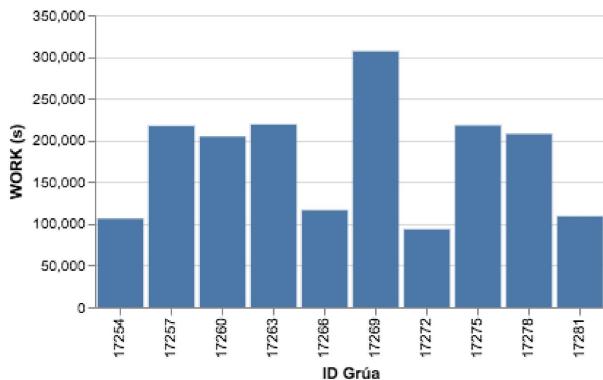


Figura 6.1: Uso de las grúas con la política del bloque más cercano

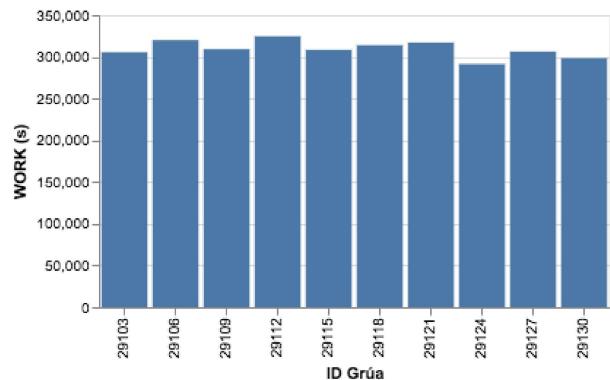


Figura 6.2: Uso de las grúas con la política aleatoria

1375 De entre las otras cuatro políticas, la asignación aleatoria se impone de forma clara.
 1376 Esto probablemente se deba a la forma en la que funciona la librería en sí, muy similar a
 1377 cómo funciona la propia política. Los números pseudo-aleatorios que se obtienen en las
 1378 asignaciones, se calculan de la misma forma que los bloques de origen de los contenedores
 1379 de retirada o reubicación. Al haber exactamente la misma probabilidad de que salga
 1380 cualquiera de los números posibles, los contenedores se reparten casi a la perfección.

1381 De hecho, las colas de espera por las grúas se mantienen bastante contenidas con la
 1382 política aleatoria, casi igual de bien que utilizando la política del bloque con la menor
 1383 cola (figuras 6.3 y 6.4). Esta última era de esperar que destacase claramente en este
 1384 sentido, pero el funcionamiento aleatorio de la librería hace que ambas arrojen resultados
 1385 bastante parejos. Pueden compararse los resultados de ambas con los que se obtienen
 1386 con la política del bloque más cercano, donde las colas llegan a crecer bastante más y
 1387 durante más tiempo (figura 6.5).

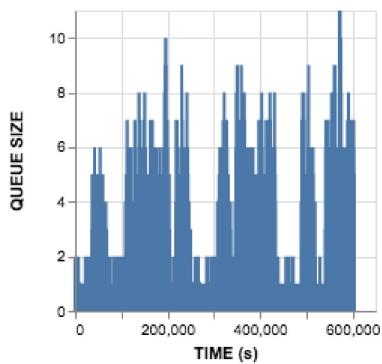


Figura 6.3: Colas con la política aleatoria

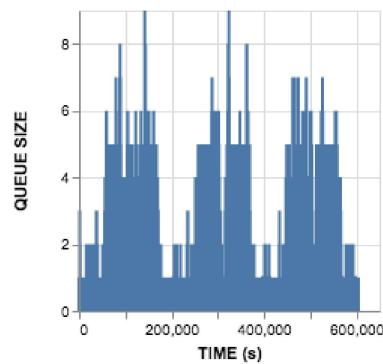


Figura 6.4: Colas con la política de la menor cola

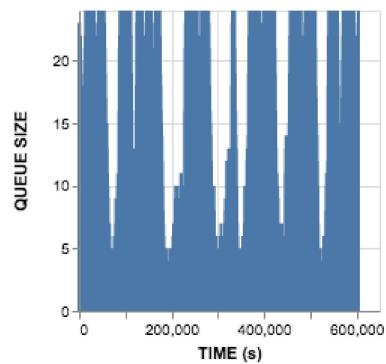


Figura 6.5: Colas con la política del bloque más cercano

1388 En cuestión de costes, no hay nada que plantear cuando se habla de políticas de
 1389 asignación. Se trata de algoritmos de planificación sin coste alguno. La asignación
 1390 aleatoria es incluso especialmente “barata”, ya que no requiere de comprobar el estado de
 1391 las colas, ni el número de contenedores por bloque, ni distancias, ni el trabajo realizado
 1392 hasta el momento por cada grúa.

1393 Se selecciona entonces al política de asignación aleatoria para las próximas fases de
 1394 la experimentación. Sin embargo, para casos en los que el modelo no funcione de forma
 1395 tan aleatoria, habría que considerar realizar más pruebas con la política que asigna los
 1396 contenedores al bloque con la menor cola.

1397 6.4. Recursos

1398 Seleccionada la mejor política de asignación, la siguiente fase es experimentar con el
 1399 número de recursos de la terminal. Este caso es algo más complejo que el anterior, puesto
 1400 que no basta con escoger el conjunto de recursos que mejores resultados otorga. Los
 1401 recursos tienen asociados un coste real que debe tenerse en cuenta a la hora de valorar
 1402 cuántos se necesitan.

1403 En la simulación, los recursos son de dos tipos: vehículos (AGV) y grúas. En este
 1404 apartado se experimenta con el número de vehículos y el número de grúas (número de
 1405 bloques de contenedores), analizando los resultados. En el apartado anterior se adelantó
 1406 que el conjunto de posibilidades para el número de vehículos sería de 15, 20, 25 y 30,
 1407 mientras que para el número de grúas sería de 5, 10 y 15. Estas cifras fueron tomadas
 1408 de la terminal de contenedores de Santa Cruz de Tenerife¹, añadiendo variaciones para
 1409 experimentar.

1410 Los resultados obtenidos pueden observarse en la tabla 6.2. De un primer vistazo, las
 1411 conclusiones más importantes que se podrían extraer son que más recursos no implican
 1412 mejores resultados. Comparando la sección de 10 bloques de contenedores (grúas) con
 1413 la de 15 bloques, vemos que para un mismo número de vehículos se obtienen siempre
 1414 peores resultados con más bloques. Esto a excepción del trabajo realizado por las grúas,
 1415 algo que ya se observaba en el experimento anterior y que se debe a la menor eficiencia y
 1416 a la saturación de los vehículos.

1417 La principal razón de que 10 bloques otorguen mejores resultados que 15 es la distancia
 1418 que los vehículos tienen que recorrer. En la tabla, puede observarse cómo el trabajo
 1419 de los vehículos se incrementa considerablemente, llegando a haber una diferencia de
 1420 más de 50 horas (180 mil segundos) para los casos con 25 vehículos. Por tanto, un
 1421 mismo número de vehículos está realizando muchísimo más trabajo en unas condiciones
 1422 con aproximadamente el mismo tráfico. Al haber más bloques, la terminal en sí es más
 1423 grande y los vehículos tienen que desplazarse mayores distancias. Esto, con un tráfico de
 1424 contenedores más alto, podría ayudar a reducir las colas, pero con el tráfico actual no
 1425 compensa y termina empeorando la situación.

1426 Se pueden sacar otras conclusiones de la tabla. Con respecto a la sección con 5 bloques
 1427 de contenedores, puede observarse que las mejoras son muy pequeñas cuando se dobla
 1428 el número de vehículos. En una terminal relativamente pequeña en la que no se puedan
 1429 situar demasiados bloques, podría no merecer la pena aumentar la inversión en vehículos.
 1430 Apenas se procesan 500 contenedores más, y estos se procesan con apenas dos horas
 1431 de diferencia. Las variaciones en el trabajo realizado por las grúas y por los vehículos
 1432 también son bajas, y más teniendo en cuenta que cuando las grúas ganan, los vehículos
 1433 pierden, y viceversa.

1434 El equilibrio se encuentra con 10 bloques de contenedores. Variar el número de
 1435 vehículos no varía tanto los resultados como con 15 bloques, pero es más recomendable
 1436 que con 5 bloques, ya que las mejoras se aprecian más fácilmente. El doble de vehículos

¹TCTenerife - Terminal de contenedores de Tenerife

			Tiempos medios (s)		
AGV	Bloques	C. procesados	Contenedor	Trabajo grúas	Trabajo vehículos
15	5	7197	30594	493950	577382
20	5	7437	26979	505945	554215
25	5	7700	23067	524753	567355
30	5	7609	22793	518686	544600
15	10	8564	16744	287799	521080
20	10	8698	11516	289444	432953
25	10	9012	7650	296398	389470
30	10	9466	6516	308645	373855
15	15	6039	54128	142245	604734
20	15	7746	22505	173745	587593
25	15	8843	14256	194270	551283
30	15	8724	10700	190604	462529

Tabla 6.2: Resultados de la experimentación con el número de recursos

¹⁴³⁷ proceza aproximadamente mil contenedores más, lo cual podría ser o no recomendable
¹⁴³⁸ dependiendo del coste de los vehículos. Donde más se aprecia una mejora es en la cantidad
¹⁴³⁹ de trabajo que tienen que realizar los propios vehículos y, sobre todo, en el tiempo que se
¹⁴⁴⁰ tarda en procesar los contenedores. Una mejoría de casi tres horas (10 mil segundos) se
¹⁴⁴¹ debe tener en cuenta.

¹⁴⁴² Para cualquier número de vehículos, los mejores resultados siempre se obtienen con
¹⁴⁴³ 10 bloques de contenedores. De entre estas opciones, se continúa la experimentación con
¹⁴⁴⁴ 25 vehículos. Aunque 30 vehículos consiguen mejores resultados, estos posiblemente no
¹⁴⁴⁵ compensen la inversión extra. Puede asumirse que el gasto en vehículos aumentaría en
¹⁴⁴⁶ un 20 %, mientras que el tiempo de procesado de contenedores mejoraría únicamente un
¹⁴⁴⁷ 15 % y, en los recursos, los vehículos trabajarían únicamente un 4 % menos y las grúas un
¹⁴⁴⁸ 4 % más. Mejoras inapreciables si se comparan con el incremento de 20 a 25 vehículos:
¹⁴⁴⁹ un 44 % menos de tiempo en procesar los contenedores, con un 2 % más de trabajo para
¹⁴⁵⁰ las grúas pero un 10 % menos para los vehículos.

¹⁴⁵¹ 6.5. Análisis de probabilidad de fallo

¹⁴⁵² En este apartado se introduce la probabilidad de fallo de los recursos en la simulación.
¹⁴⁵³ Hasta el momento, un recurso que trataba de realizar un trabajo iba a poder desempeñarlo
¹⁴⁵⁴ sin ningún problema. Sin embargo, en situaciones reales, siempre existe una pequeña
¹⁴⁵⁵ probabilidad de que haya algún error en cada tarea que realiza un recurso. Por ejemplo,
¹⁴⁵⁶ podría darse un fallo mecánico que obligase a realizar reparaciones por un tiempo.

¹⁴⁵⁷ A día de hoy, la probabilidad de que ocurran este tipo de fallos es bastante pequeña,
¹⁴⁵⁸ ya que la tecnología es cada vez más fiable. Sin embargo, se ha experimentado con
¹⁴⁵⁹ tres probabilidades de fallo diferentes, para poder observar hasta qué punto los errores
¹⁴⁶⁰ pueden afectar a la eficiencia del sistema.

¹⁴⁶¹ Además, podría darse el caso de que, a la hora de realizar una inversión en recursos,
¹⁴⁶² el proveedor garantice un cierto nivel de calidad en función del precio del recurso.
¹⁴⁶³ Simulando diferentes probabilidades de fallo, se puede observar hasta qué punto es

1464 admisible ahorrar en costes sacrificando en la calidad en el recurso.

1465 Pese a haber concluido en el apartado 6.4 que 25 vehículos y 10 bloques es probable-
 1466 mente la distribución de recursos más eficiente, en este apartado también se experimenta
 1467 con otras distribuciones. Una de ellas es la de 15 vehículos y 5 bloques, con el objetivo de
 1468 observar cómo los fallos afectan a terminales más pequeñas. La otra es la de 30 vehículos
 1469 y 15 bloques: aunque se pudo observar que ofrecía peores resultados, tener recursos
 1470 adicionales podría ser útil al introducir la probabilidad de fallo.

1471 Una probabilidad de fallo baja es del 1 %. La probabilidad media sube al 2,5 % y la alta
 1472 a un 5 %.

* G.FDS / V.FDS - Tiempo fuera de servicio para grúas / vehículos

* C.P. - Contenedores procesados

* T.G. / T.V. - Tiempo de trabajo de grúas / vehículos

P. Fallo	AGV	Bloques	C.P.	Tiempos medios (s)				
				Contenedor	T.G.	T.V.	G.FDS	V.FDS
BAJA	15	5	6232	54783	435375	584927	49357	12853
BAJA	25	10	8933	10332	295562	443411	32368	11158
BAJA	30	15	8962	10428	194497	509118	23108	9938
MEDIA	15	5	5184	105092	364098	561559	98293	28663
MEDIA	25	10	8620	15971	288016	507517	81974	29849
MEDIA	30	15	8706	15088	191416	529539	54074	24022
ALTA	15	5	4052	187775	289625	537955	161642	44458
ALTA	25	10	7203	32265	247559	520819	136590	48327
ALTA	30	15	7698	25832	173204	532659	97761	43467

Tabla 6.3: Resultados de la experimentación con la probabilidad de fallo

1473 Con respecto a las ejecuciones con 25 vehículos y 10 bloques, se puede observar (tabla
 1474 6.3) cómo se mantiene cierta calidad en los tres casos, siempre procesando más de 7000
 1475 contenedores. Como es lógico, el tiempo que los contenedores permanecen en el sistema
 1476 se incrementa, debido al tiempo que se pierde en reparaciones cada vez que un recurso
 1477 falla. En los casos en los que, además, lo que falla es una grúa, esto puede afectar no sólo
 1478 al contenedor que ocupa la grúa, sino también a aquellos que están en cola para ocuparla
 1479 en cuanto esta quede libre.

1480 En estos casos, utilizar únicamente 15 vehículos y 5 bloques empeora considerable-
 1481 mente los resultados. Con una probabilidad de fallo baja, los resultados continúan siendo
 1482 medianamente aceptables, aunque los vehículos desarrollan demasiado trabajo, teniendo
 1483 menos de medio día de tiempo libre. Además, este trabajo excesivo de los vehículos no le
 1484 ahorra trabajo a las grúas, las cuales muestran los índices de ocupación más altos de todos
 1485 los experimentos hasta el momento. En cuanto la probabilidad de fallo se hace media o
 1486 alta, los resultados pasan a ser inadmisibles. El número de contenedores procesados cae
 1487 considerablemente, pero sobre todo se incrementa demasiado el tiempo de procesado de
 1488 los contenedores, que crece a más de un día entero para probabilidad media, y más de
 1489 dos días para probabilidad alta.

1490 Al introducir la probabilidad de fallo, disponer de un mayor número de recursos ayuda
 1491 a mantener la eficiencia del sistema. Cuando se produce un fallo, el resto de recursos
 1492 sigue funcionando sin retrasar excesivamente el sistema. Esto puede observarse en las

ejecuciones con 30 vehículos y 15 bloques. Mientras que en la experimentación con los recursos (apartado 6.4) los resultados terminaban siendo incluso peores que con menos bloques, en este caso son ligeramente mejores para probabilidades de fallo bajas, y la mejoría se incrementa según crece la probabilidad de fallo. Ahora bien, el tiempo medio de trabajo de los vehículos continúa siendo más elevado que en terminales con 10 bloques, debido a las mayores distancias que recorren.

En cuestión a los tiempos fuera de servicio, están dentro de lo esperado en el sentido de que crecen según crece la probabilidad de fallo. Lo más relevante a destacar es la clara diferencia entre los tiempos fuera de servicio de las grúas y los de los vehículos. El de las grúas es considerablemente mayor, probablemente debido a que realizan más trabajos. El número de grúas es siempre menor al número de vehículos.

Es importante tener en cuenta que una probabilidad de fallo baja ya supone que, de cada 50 trabajos, los recursos fallarán probablemente en uno. Recordar que se determina si se da un fallo o no dos veces por trabajo: cuando el recurso se desplaza a buscar el contenedor, y cuando el recurso transporta el contenedor (consultar sección 5.5.1). Entonces, en una terminal en la que cada recurso lleva a cabo 50 trabajos al día, lo cual no es nada excesivo, estos necesitarían reparaciones una vez al día. Considerar una probabilidad de fallo mayor a esa en un caso real sería innecesario. Por esta razón, en fases posteriores de la experimentación, se establecerá permanentemente la probabilidad de fallo como baja (1 %).

6.6. Análisis de períodos de indisponibilidad

Hasta el momento, los experimentos realizados han asumido la disponibilidad absoluta de los recursos durante todo el tiempo de la simulación. Salvo que ocurriese un fallo que obligase a los recursos a quedarse fuera de servicio por reparaciones, se garantizaba que en cualquier momento todos los recursos estarían en activo. Sin embargo, esto no se aplica en casos reales.

En una terminal marítima, los recursos están sujetos a múltiples políticas que los dejan fuera de servicio durante amplios intervalos de tiempo. Esto puede ser por cuestión de horarios de trabajo o revisiones de seguridad, entre otros motivos. Por ello, se ha implementado la posibilidad de modificar la disponibilidad de los recursos de la simulación (sección 5.5.1). Con una disponibilidad alta, el recurso está disponible el 95 % del tiempo total. Con una disponibilidad media, el porcentaje desciende al 85 %. Finalmente, con una disponibilidad baja, desciende al 70 %.

En un principio, podría parecer que estos porcentajes de disponibilidad siguen siendo excesivamente altos. Únicamente teniendo en cuenta los horarios nocturnos, la mayoría de recursos debería permanecer fuera de servicio aproximadamente un 40 % del tiempo. Sin embargo, aquellos horarios en los que hay menos recursos suelen ser también aquellos horarios en los que el tráfico de contenedores es menor. La forma en la que funciona la librería de simulación utilizada no permite controlar horarios de forma directa ni modificar el ritmo de generación de contenedores en función del tiempo. Sí se baraja la idea de implementar estas funciones en un futuro. Por ejemplo, poder establecer que entre las cuatro y las seis de la tarde el tráfico crece un 30 %.

En esta fase de la experimentación, se ha hecho uso de las mismas distribuciones de recursos que en la fase anterior. Además de esto, la probabilidad de fallo se establece permanentemente como baja. Los resultados aparecen en la tabla 6.4.

- * G.FDS / V.FDS - Tiempo fuera de servicio para grúas / vehículos
- * C.P. - Contenedores procesados
- * T.G. / T.V. - Tiempo de trabajo de grúas / vehículos

Disponibilidad	Tiempos medios (s)							
	AGV	Bloques	C.P.	Contenedor	T.G.	T.V.	G.FDS	V.FDS
ALTA	15	5	5054	89525	356538	556732	69081	41166
ALTA	25	10	7672	23307	260234	526969	59903	40085
ALTA	30	15	7486	20845	167695	527281	49049	37627
MEDIA	15	5	2923	164041	203344	503989	113580	93310
MEDIA	25	10	3265	111077	113050	491501	103817	90824
MEDIA	30	15	3290	127502	75965	497116	99021	90566
BAJA	15	5	1601	213424	109284	421533	193881	170899
BAJA	25	10	1419	148691	46768	422201	186509	173099
BAJA	30	15	1178	157867	25592	409263	183671	165310

Tabla 6.4: Resultados de la experimentación con la disponibilidad de recursos

1538 Efectivamente, la disponibilidad de recursos demuestra ser una variable extremada-
 1539 mente importante y que afecta de forma muy clara a los resultados que se obtienen. Con
 1540 una disponibilidad alta, las tres distribuciones de recursos seleccionadas son capaces
 1541 de mantener cifras admisibles, procesando siempre más de 5000 contenedores incluso
 1542 con solo 15 vehículos y 5 bloques. El principal inconveniente en estos casos es el claro
 1543 incremento en el tiempo de procesado de los contenedores, lo cual es un grave problema
 1544 para la extracción de contenedores, en las cuales los transportistas tienen que esperar a
 1545 que se complete todo el proceso de extracción. Es un problema menor en contenedores
 1546 que son almacenados, puesto que el transportista puede desentenderse de él después de
 1547 una primera cola.

1548 Cuando la disponibilidad cae a media o baja, los resultados empeoran de forma con-
 1549 siderable, casi dejando de ser admisibles. El hecho de que procesen menos de 5000
 1550 contenedores no es especialmente preocupante, puesto que, como se explica en el apar-
 1551 tado 5.4.2, el procesamiento más lento de los contenedores de los barcos podría estar
 1552 resultando en la generación de menos contenedores. Sí es más problemático el tiempo de
 1553 procesamiento medio por contenedor, que alcanza valores excesivamente grandes.

1554 También es importante tener en cuenta el desaprovechamiento de las grúas cuando la
 1555 disponibilidad disminuye. Aunque también ocurre con los vehículos, la caída no es tan
 1556 pronunciada. Por ejemplo, en la distribución con 30 vehículos y 15 bloques, las grúas
 1557 apenas se utilizan 7 horas en toda la semana. Esto sugiere que el “atasco” se produce
 1558 en los vehículos, que tienen problemas para llevar los contenedores a los bloques o para
 1559 extraerlos de los mismos.

1560 Las operaciones internas de la terminal, dentro de unos límites, pueden retrasarse sin
 1561 afectar negativamente. Un contenedor que es almacenado puede tardar más de un día
 1562 en almacenarse, pero si no va a ser extraído en otros tres días, no supone un problema
 1563 mayor. En cambio, existen dos tiempos que sí son fundamentales: el tiempo medio de
 1564 espera en las colas de llegada y el tiempo total de los contenedores que son extraídos.
 1565 Estos tiempos afectan directamente a los transportistas que traen contenedores para su
 1566 almacenamiento o que solicitan llevarse contenedores almacenados.

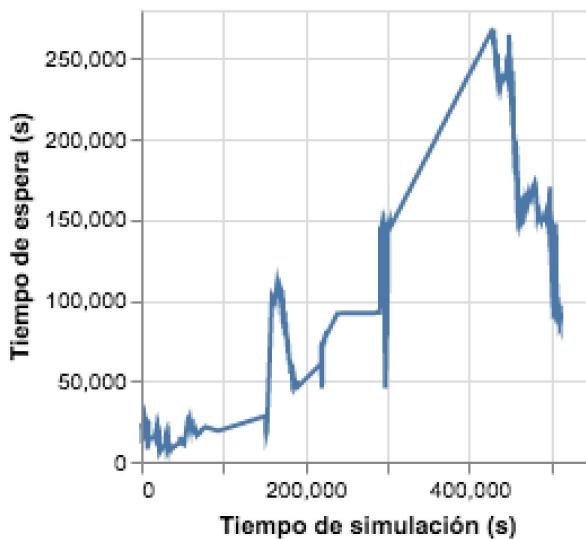


Figura 6.6: Colas de llegada con una disponibilidad media para 25 vehículos y 10 bloques

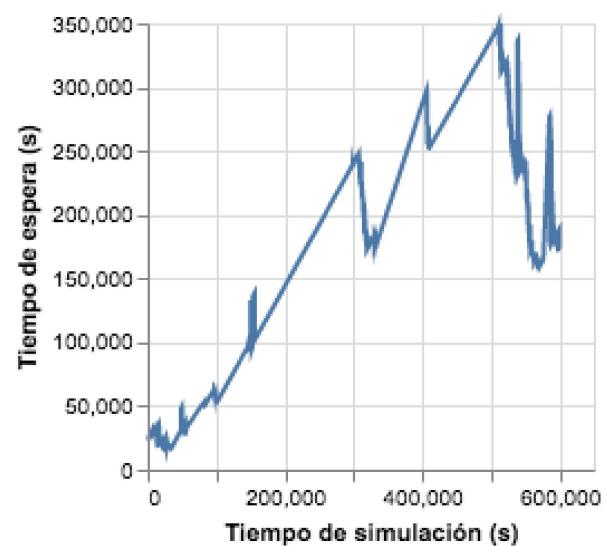


Figura 6.7: Tiempos de salida con una disponibilidad media para 25 vehículos y 10 bloques

1567 En las gráficas 6.6 y 6.7, puede observarse cómo los tiempos se hacen excesivamente
 1568 grandes. Pese a que se recuperan considerablemente según se llega a finales de la
 1569 semana, se trata de tiempos de espera inaceptables para un transportista. Para cuando
 1570 la disponibilidad es baja, los resultados son aún peores (gráficas 6.8 y 6.9), ni siquiera
 1571 siendo capaces de recuperarse a finales de semana.

1572 Es importante tener en cuenta, eso sí, que estos tiempos incluyen los de aquellos
 1573 contenedores que llegan o se marchan tanto por tierra como por mar. Para un barco,
 1574 esperar varios días es admisible. Por tanto, se podría solventar ligeramente esta situación
 1575 otorgando mayor prioridad a los contenedores por tierra (sección 5.6). Sin embargo, esto
 1576 solo sería una pequeña ayuda, no solucionaría totalmente la situación.

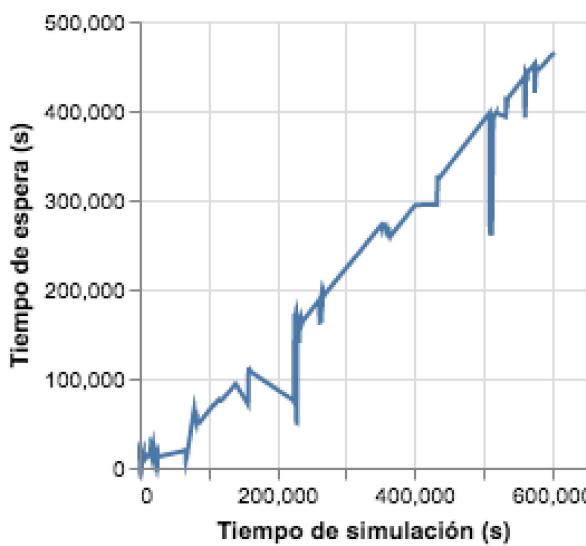


Figura 6.8: Colas de llegada con una disponibilidad baja para 25 vehículos y 10 bloques

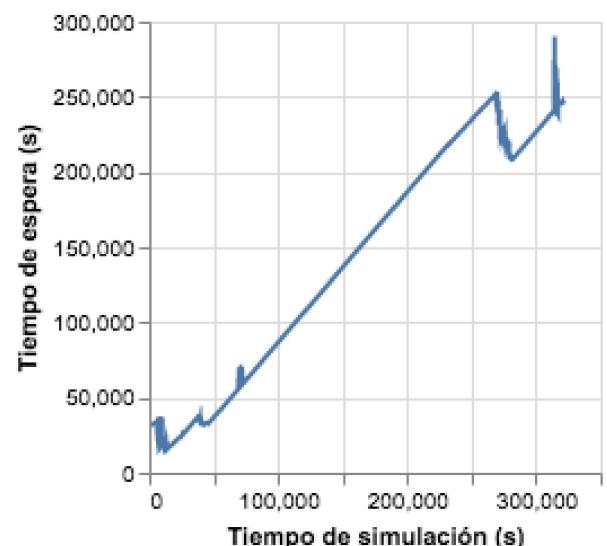


Figura 6.9: Tiempos de salida con una disponibilidad baja para 25 vehículos y 10 bloques

1577 Principalmente, se hace necesaria una política de asignación algo más compleja para
 1578 mantener una terminal de este tamaño con una disponibilidad de recursos media o baja.
 1579 Probablemente, esta nueva política se basaría en realizar una planificación previa por
 1580 día o incluso por semana, aprovechando algún tipo de conocimiento previo o solicitando
 1581 información de los movimientos de contenedores antes de que estos ocurran para poder
 1582 organizar los recursos. Es decir, algún tipo de política de asignación heurística.

1583 Cabe destacar que algunos de los mejores resultados con disponibilidad baja son obtenidos
 1584 por la distribución que hace uso de menos recursos, con 15 vehículos y 5 bloques.
 1585 Es la distribución que mejor aprovecha las grúas y la que más contenedores procesa,
 1586 aunque tarda mucho más tiempo en hacerlo. Esto sugiere que, en estas condiciones, el
 1587 ahorro en recursos podría ser lo más recomendable.

1588 Para la última fase de la experimentación, se utilizará la disponibilidad alta. Aparte
 1589 del hecho de que proporciona claramente los mejores resultados, es más indicada para
 1590 la forma en la que funciona la librería. La generación de contenedores se reparte a lo
 1591 largo del tiempo sin realizar ninguna distinción en los horarios y sin modificar el ritmo de
 1592 generación. Por tanto, no hay forma de programar disponibilidades de forma eficiente.

1593 6.7. Análisis de prioridad de contenedores

1594 Un factor muy común en cualquier terminal marítima de contenedores es la gestión de
 1595 acuerdo a una prioridad. Hasta el momento, todos los contenedores se procesaban como
 1596 iguales, en el mismo orden en el que llegaban al sistema. Sin embargo, en terminales
 1597 reales es muy común que lleguen contenedores con mayor prioridad que otros.

1598 En la librería, se ha implementado un orden de prioridades para los contenedores.
 1599 La más común es la prioridad normal, que es la que se ha estado aplicando a todos los
 1600 contenedores en todos los experimentos hasta el momento. A partir de ahora, habrá
 1601 un 5 % de posibilidades de que un contenedor tenga prioridad baja. Representaría un
 1602 contenedor para el que se sabe que se tiene tiempo de sobra para procesar. Finalmente,
 1603 está la prioridad alta, superior a las otras dos. Cada contenedor tiene también un 5 % de
 1604 posibilidades de tener prioridad alta, la cual hará que se sitúe de forma inmediata en las
 1605 primeras posiciones de cada cola a la que entre.

1606 Este experimento se ha realizado con el objetivo de analizar los posibles impactos en la
 1607 eficiencia del sistema que puede tener la integración de prioridades en los contenedores.
 1608 La tabla 6.5 deberá compararse con la tabla 6.4 para comprobar si existen diferencias
 1609 palpables. Más allá de esto, la implementación de este sistema también otorga una cierta
 1610 seguridad al modelo, pues permitirá garantizar un menor tiempo de procesamiento a
 1611 ciertos contenedores.

AGV	Bloques	C. procesados	Tiempos medios (s)		
			Contenedor	Trabajo grúas	Trabajo vehículos
15	5	5143	79917	358140	557128
25	10	7446	20703	251994	478065
30	15	7560	21284	169343	536655

Tabla 6.5: Resultados de la experimentación con prioridad de contenedores

1612 Observando los resultados y comparando con los de la tabla 6.4, no se puede concluir

que la integración de prioridades afecte a los resultados generales. Las cifras obtenidas son aproximadamente las mismas, con alguna desviación residual fruto del azar. La ventaja real de la implementación de esta funcionalidad, como ya se ha mencionado, está en la posibilidad de garantizar a ciertos contenedores un paso más rápido por el sistema.

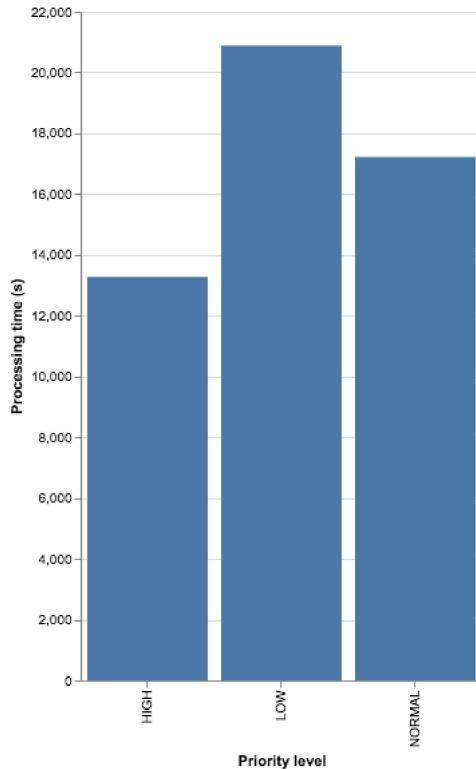


Figura 6.10: Tiempo de procesado según el nivel de prioridad

El gráfico 6.10 se ha elaborado a partir de una muestra representativa del experimento anterior. Agrupa los contenedores según su prioridad y muestra en un gráfico de barras el tiempo medio que tardan en ser procesados por el sistema. Como puede observarse, los contenedores con alta prioridad tardan más de una hora menos en ser procesados que los contenedores con prioridad normal. Estos, a su vez, tardan más de una hora menos en ser procesados que los de prioridad baja.

Claramente, los problemas que suponen los períodos de indisponibilidad, los fallos y, en algunos casos, la acumulación de un excesivo tráfico sobre los vehículos, afectan también a los contenedores prioritarios. Aunque se puede garantizar un procesamiento más rápido, seguirá dependiendo enormemente del estado del sistema. Estas son las principales razones que hacen que la prioridad de contenedores no afecte a los resultados generales de la simulación.

¹⁶²⁹ Capítulo 7

¹⁶³⁰ Conclusiones y líneas futuras

¹⁶³¹ 7.1. Librería desarrollada

¹⁶³² En la sección 1.2, se estableció que el objetivo era diseñar una librería de simulación
¹⁶³³ que permitiese simular las operaciones de una terminal marítima de contenedores ge-
¹⁶³⁴ nérica. Teniendo en cuenta los parámetros definidos en la sección 3.4, se podría decir
¹⁶³⁵ que este objetivo se ha cumplido de forma satisfactoria. Si bien es cierto que aspectos
¹⁶³⁶ como la prioridad de contenedores no se implementó exactamente como estaba planeado
¹⁶³⁷ en un principio, los resultados no se ven afectados. En cambio, se han implementado
¹⁶³⁸ sistemas adicionales que sí mejoran la fidelidad a la realidad, y que en un principio
¹⁶³⁹ tampoco estaban planeados. El mejor ejemplo de esto es la representación de la llegada y
¹⁶⁴⁰ salida de barcos de forma realista.

¹⁶⁴¹ Respecto a la fidelidad a la realidad de los resultados que se obtienen mediante el
¹⁶⁴² uso de la librería, estos pueden garantizarse para terminales marítimas genéricas, con
¹⁶⁴³ operaciones básicas. Básicamente, los resultados son fiables para una terminal que
¹⁶⁴⁴ funcione de igual manera que el modelo en sí. Aunque no se dispone de datos reales
¹⁶⁴⁵ de una terminal para contrastar, el uso del paradigma de simulación basada en agentes
¹⁶⁴⁶ permite diseñar un entorno útil, simplemente a partir una información de base sobre qué
¹⁶⁴⁷ agentes componen el sistema, qué operaciones realizan y cómo se relacionan entre sí.

¹⁶⁴⁸ Esta información de base es fundamental, y define hasta qué punto ha sido posible
¹⁶⁴⁹ diseñar el modelo de forma fiel a la realidad. Se conoce el funcionamiento básico de
¹⁶⁵⁰ una terminal marítima de contenedores, pero muchos detalles eran desconocidos. Por
¹⁶⁵¹ ejemplo, el rendimiento de los recursos: la velocidad de desplazamiento de los vehículos y
¹⁶⁵² las grúas, la probabilidad que tienen de fallar, la libertad de movimiento que tienen, etc.
¹⁶⁵³ Estos eran factores que tuvieron que estimarse según se creía conveniente. Habría sido
¹⁶⁵⁴ de utilidad disponer de datos reales.

¹⁶⁵⁵ Por supuesto, se debe tener en cuenta que las capacidades del modelo han estado
¹⁶⁵⁶ limitadas durante todo el desarrollo por la librería de simulación. Contemporáneamente
¹⁶⁵⁷ al desarrollo de este trabajo de fin de grado, la librería de simulación iba evolucionando
¹⁶⁵⁸ y adquiriendo nuevas funcionalidades que otorgaban nuevas opciones. Por ejemplo, las
¹⁶⁵⁹ colas con un orden de prioridad son una opción que la librería no ponía a disposición
¹⁶⁶⁰ hasta las últimas fases de desarrollo del modelo.

¹⁶⁶¹ A día de hoy, la librería de simulación sigue evolucionando e introduciendo nuevas
¹⁶⁶² opciones. Según continúe mejorando, el modelo desarrollado se podría ir modificando
¹⁶⁶³ para aprovechar estas nuevas funciones. Esto, progresivamente, asemejaría aún más el
¹⁶⁶⁴ comportamiento al de una terminal de contenedores real.

1665 El modelo desarrollado se ha compilado en una nueva librería para simulación de
 1666 terminales marítimas de contenedores. Esta se encuentra disponible para su uso en un
 1667 repositorio público en Github¹. Se debe clonar este repositorio para descargar el fichero
 1668 *jar* con la librería. Este fichero puede ser importado como dependencia en cualquier otro
 1669 proyecto para ser utilizado.

"url": "<https://github.com/alu0100892833/Terminal-Simulator-Library.git>"

1670 Aunque la totalidad de la librería está perfectamente documentada (en inglés), lo más
 1671 sencillo y más recomendable es hacer uso de la clase *TerminalSimulator*. Se trata de
 1672 una clase para instanciar objetos que permiten realizar simulaciones de forma rápida y
 1673 personalizando multitud de parámetros. De hecho, es la clase que se ha utilizado para
 1674 realizar la experimentación (capítulo 6). En el apéndice A se encuentra toda la información
 1675 para hacer uso de esta clase. También se hace una explicación básica de como hacer uso
 1676 de la clase *TerminalModelBuilder*, para aquellos casos en los que sea necesario un mayor
 1677 control sobre la ejecución.

1678 7.2. Posibles usos

1679 El propósito principal del capítulo 6 de este documento era demostrar el potencial
 1680 del modelo desarrollado. Conociendo los principales parámetros de una terminal de
 1681 contenedores cualquiera, se puede simular sus operaciones con diferentes políticas y
 1682 recursos para tratar de encontrar la distribución que mejores resultados otorga.

1683 En el caso de la experimentación del capítulo 6, los parámetros se modifican en
 1684 diferentes fases, seleccionando los mejores en cada fase para aplicarlos en la siguiente.
 1685 La experimentación se lleva a cabo, entonces, a modo de algoritmo evolutivo, en el que
 1686 los mejores elementos de cada fase sobreviven.

1687 Las conclusiones que se pueden ir extrayendo son múltiples, y no siempre las esperadas.
 1688 Destaca el hecho de que tener un mayor número de recursos no siempre significa mejores
 1689 resultados (sección 6.4). Además, siempre es importante tener en cuenta que hay factores
 1690 muy importantes que la librería no es capaz de contemplar y que deben estudiarse y
 1691 analizarse desde fuera. Este es el caso de parámetros como el coste de los recursos, el
 1692 coste de su mantenimiento y reparaciones, etc.

1693 En el caso de la experimentación realizada, según se iban introduciendo nuevos ele-
 1694 mentos al modelo (principalmente los períodos de disponibilidad), la principal conclusión
 1695 que se puede extraer es que, para terminales grandes, una política de asignación básica
 1696 no es lo bastante eficiente. En cuanto la disponibilidad de los vehículos caía a baja (70 %
 1697 del tiempo), los resultados empeoraban considerablemente. Una disponibilidad como esa
 1698 podría ser perfectamente común en cualquier terminal, lo cual sugiere que hacen falta
 1699 ciertas mejoras en la forma de mover los contenedores por la terminal.

1700 En concreto, se pueden extraer conclusiones muy importantes de las gráficas 6.6, 6.7,
 1701 6.8 y 6.9. Sabiendo que hay problemas a la hora de garantizar tiempos de espera bajos a
 1702 los transportistas, se pueden aplicar nuevas políticas que mantengan estos tiempos lo
 1703 más bajos posible.

1704 Este método de experimentación puede aprovecharse para el resto de objetivos o
 1705 variables a optimizar de la terminal. La mayoría de datos importantes de cada ejecución
 1706 se pueden exportar fácilmente utilizando funciones que la librería desarrollada pone

¹<https://github.com/alu0100892833/Terminal-Simulator-Library.git>

1707 a disposición. Estos se analizan para concluir si son correctos o tienen que mejorarse.
 1708 A partir de ahí, se toman las medidas adecuadas. En el capítulo 6 se exponen muchos
 1709 ejemplos de variables a optimizar: por ejemplo, tiempos de trabajo de los recursos o los
 1710 tiempos de procesado de contenedores. Pero hay muchas otras: tiempos de espera de
 1711 los contenedores para que se les asigne una grúa, tiempo en el que son agarrados por la
 1712 grúa, tiempo en el que son colocados en un vehículo, tiempos de desplazamiento, etc.

1713 7.3. Líneas futuras

1714 En general, como ya se ha mencionado en la sección 7.1, el desarrollo de la librería
 1715 como simulador de operaciones de una terminal de contenedores está muy completo.
 1716 Las operaciones principales se simulan correctamente. Añadir nuevas funcionalidades
 1717 en este nivel supondría integrar infraestructuras de una terminal que no eran objetivo
 1718 de este trabajo de fin de grado, pero que sin duda podrían ser útiles. Algunas de estas
 1719 infraestructuras pueden extraerse de la imagen 3.1 (sección 3.1), como son la aduana,
 1720 la zona para contenedores refrigerados, zona para contenedores radioactivos, etc.. O,
 1721 incluso, simular también la carga y descarga de contenedores de los barcos, teniendo
 1722 estos sus propias grúas y sus propias políticas.

1723 Dentro de las funcionalidades que formaban parte de los objetivos iniciales, la imple-
 1724 mentación de la prioridad extrema (consultar sección 5.6) es la única pendiente. Las
 1725 limitaciones actuales de la librería utilizada lo hacen muy complicado, pero en un futuro
 1726 podría ser posible su implementación.

1727 Más allá de esto, también es interesante la posibilidad de integrar más de una grúa
 1728 por bloque de contenedores. Aunque no es una funcionalidad que se encuentre entre
 1729 los objetivos iniciales, sí supone la expansión de uno de estos. Otorgaría una mayor
 1730 flexibilidad en el control de recursos, además de que en cualquier terminal es muy
 1731 común encontrar varias grúas por bloque de contenedores. Sin embargo, no bastaría
 1732 con simplemente añadir un recurso extra: los raíles sobre los que se moverían ambas
 1733 grúas serían los mismos, con lo cual, si por ejemplo hubiese dos grúas, una no podría
 1734 sobrepasar a la otra. Esto exigiría una planificación extra en la forma en la que las grúas
 1735 se asignan a los contenedores.

1736 Como se menciona en la sección 7.1, disponer de información de una terminal real
 1737 para poder contrastar la información de la simulación sería extremadamente útil. No
 1738 sólo permitiría asegurar la calidad de los resultados de la simulación, sino que además
 1739 permitiría el desarrollo de políticas de asignación basadas en algoritmos heurísticos que
 1740 aprovechasen esta información para mejorar considerablemente los resultados.

1741 Por último, sería interesante simplificar la forma de especificar los parámetros de
 1742 ejecución. Cada terminal marítima tiene sus propias especificaciones para cada parámetro
 1743 de la simulación. Al haber tantas opciones, se podría codificar toda esta información
 1744 en un fichero, haciendo que el simulador lea esa información y cargue los parámetros
 1745 adecuados de forma más cómoda para el usuario.

1746 **Capítulo 8**

1747 **Summary and Conclusions**

1748 **8.1. Resulting Library**

1749 In section 1.2, it was declared that the objective of this project was to design and
1750 develop a Java library for the simulation of a generic maritime terminal. Considering the
1751 main parameters defined in section 3.4, it could be said that this objective was effectively
1752 accomplished. Although some functionalities were not implemented as it was planned
1753 in the beginning, like the priority of containers, the final results are not affected. In
1754 exchange, additional functionalities were implemented. These features, not originally
1755 planned, bring the simulation model closer to reality. The best example of this is the
1756 realistic representation of the arrival and departure of the ships.

1757 About the the results that are obtained using the library, they could be considered reliable
1758 for generic maritime terminals, whose operations are mostly based on the three basic
1759 operations contemplated in section 3.2: arrival, collection and relocation of containers.
1760 Although there is no real data available to contrast, the usage of agent based modeling
1761 allows to design a realistic environment, just based on fundamental information about
1762 what agents are part of the system, what operations do they perform and how do they
1763 relate with each other.

1764 This basic information is very important. It defines to what extent it has been possible
1765 to design a reliable model. It is well known how a terminal works at a basic level, but
1766 a lot of details were unknown. For example, the resource's performance: speed, failure
1767 probability, movement limitations, etc. The values of these variables had to be set as it
1768 was thought convenient. Having real data would have been useful.

1769 Naturally, the development of the model has been limited by the simulation library it is
1770 based on. While the model was being developed, the simulation library was also improving
1771 and receiving new methods that allowed the implementation of new functionalities. For
1772 example, the priority queues were not possible until very late in the development.

1773 Today, the simulation library is still in development. As it gets better, the model could
1774 take advantage of the new functionalities it provides. This way, the model would get closer
1775 to reality overtime.

1776 The simulation model has been compiled in a new simulation library, limited to the
1777 simulation of maritime terminals. This is available to be used by any developer, just
1778 by accesing a public GitHub¹ repository. It is as simple as cloning the repository and
1779 importing the main *jar* file as a dependency on any project.

¹<https://github.com>

"url": "<https://github.com/alu0100892833/Terminal-Simulator-Library.git>"

1780 Although the library is perfectly documented using Javadoc, the simplest and most
 1781 recommendable way of using it is through the *TerminalSimulator* class. It allows the
 1782 instantiation of objects that allow to run simulations in an easy way, allowing the custo-
 1783 mization of multiple parameters. In fact, this class was used for performing the experi-
 1784 mentation (chapter 6). In Appendix A, all the information about this class can be found,
 1785 with the explanation of how its main methods work. It is also explained how to use the
 1786 *TerminalModelBuilder* class, for those cases in which more control over the simulation is
 1787 needed.

1788 8.2. Potential Applications

1789 The main purpose of chapter 6 is to demonstrate the potential of the developed library.
 1790 Setting the values of the main parameters equal to those in a real terminal, its operations
 1791 can be simulated using different policies and resources, hoping to find the distribution
 1792 that gives the best results.

1793 In the case of the experimentation of chapter 6, the parameters are modified in
 1794 different stages, selecting the best in each stage to apply them again in the next one.
 1795 The experimentation is carried out, therefore, in a similar way to how an evolutionary
 1796 algorithm works. The best values of each stage survive into the next one.

1797 Several useful conclusions can be extracted, which by any means are the ones that
 1798 were expected. Highlights the fact that having more resources does not always mean
 1799 that better results will be obtained (section 6.4). Besides, it is important to keep in mind
 1800 that there are some very influential factors that the developed library is not able to
 1801 contemplate, and must be studied and analyzed from the outside. This is the case of
 1802 some parameters like the cost of each resource, the cost of their maintenance and their
 1803 reparations, etc.

1804 In the case of the experimentation of chapter 6, with each new element introduced
 1805 in the model (mainly the availability of the resources), the main conclusion that can be
 1806 extracted is that, in big terminals, a basic container allocation policy is not good enough.
 1807 When the availability of the resources is low (70 % of time), the results get considerably
 1808 worst. Such an availability percentage could be perfectly normal in any terminal, which
 1809 suggests that some improvements are necessary in the way containers are handled.

1810 More specifically, some very important conclusions can be extracted from the graphs
 1811 6.6, 6.7, 6.8 and 6.9. Knowing that the model is not capable of guaranteeing low waiting
 1812 times for carriers, some new allocation policies could be applied, in an attempt to reduce
 1813 these times.

1814 This experimentation method can be used to improve the other objectives (variables
 1815 to optimize). Most relevant data of each execution can be easily exported to external
 1816 files using methods that the developed library makes available. This data is analyzed,
 1817 in an attempt to conclude if they are good enough or they need to be improved. From
 1818 there, the right actions are taken. In chapter 6, a lot of examples of objectives that need
 1819 improvement are exposed: it is the case of the mean working time of the resources or
 1820 the mean processing time of the containers. But, in addition to this, there are a lot more:
 1821 waiting time of the containers that are requesting to seize a crane, time when they are
 1822 picked up by the crane, time when they are placed on a vehicle, movement time, etc.

1823 8.3. Future Possibilities

1824 In general, as mentioned in section 8.1, the development of the library as a container
 1825 management simulator is pretty much complete. The main operations are correctly
 1826 simulated. Adding new functionalities at this level would mean the representation of other
 1827 infrastructures of maritime terminals that were not in the scope of this project, although
 1828 they would be a nice addition in the future. Some of these infrastructures can be observed
 1829 on image 3.1 (section 3.1), such as the customs office, the refrigerated merchandise area,
 1830 the radioactive merchandise area, etc.. Or even the integration of the simulation of the
 1831 loading and unloading of containers from ships, assigning the ships their own cranes and
 1832 policies.

1833 Within the functionalities that were planned from the beginning, the implementation
 1834 of extreme container priorities (section 5.6) is the only one that is pending. The current
 1835 limitations of the simulation library make this very complicated to program, but it could
 1836 become possible in a near future.

1837 Beyond this, it would also be very interesting the possibility of integrating multiple
 1838 cranes in each container block. It would mean the expansion of a functionality that was
 1839 originally planned for the project, allowing more flexibility in how the resources are
 1840 controlled. Besides, in any terminal, it is very common to find more than one crane per
 1841 container block. However, simply adding one extra resource would not be enough: the
 1842 rails on which the cranes move would be the same for all cranes of a container block, so
 1843 when having, for example, two cranes, none of them could never surpass the other one.
 1844 This would demand extra planning in the way cranes are assigned to containers.

1845 As mentioned in section 8.1, having information of a real maritime terminal would
 1846 be extremely useful. This would not only allow to contrast the results obtained with the
 1847 simulation, but it would also allow to develop new allocation policies based on heuristic
 1848 algorithms. These would take advantage of this information to considerably improve the
 1849 final results.

1850 Finally, it would be interesting to simplify the way parameters are specified for the
 1851 model. Every maritime terminal has its own specifications for each parameter of the
 1852 simulation. With so many options, it would be easier for the user to encode all this
 1853 information in a single file, making the simulator read that file to load the appropriate
 1854 values on each simulation parameter.

1855 **Capítulo 9**

1856 **Presupuesto**

1857 Para calcular el presupuesto estimado necesario para llevar a cabo el proyecto, se
1858 realizará un cálculo básico en función de las horas de trabajo necesarias. En principio,
1859 no es necesario realizar ninguna adquisición de hardware, si se tiene en cuenta que
1860 el ordenador personal necesario para programar el modelo ya se posee. Los gastos en
1861 electricidad, internet, etc. se desestiman a la hora de realizar el cálculo, ya que se asume
1862 que están garantizados.

1863 Para calcular el salario por hora, se toma como referencia el salario medio de un
1864 especialista en simulación en Estados Unidos. De acuerdo con la web *PayScale.com*¹,
1865 un servicio donde profesionales pueden compartir su salario de forma anónima para
1866 elaborar estadísticas, el salario anual medio de un especialista en simulación es de 74.323
1867 dólares USD. Estos datos están actualizados a enero de 2018. Esto, convertido a euros y
1868 considerando unas ocho horas de trabajo diarias, resulta en un salario por hora de 30
1869 euros. Las tablas 9.1 y 9.2 resumen entonces el presupuesto final.

Módulo	Horas
Documentación en terminales marítimas	18
Diseño de la terminal y modelado del problema	25
Implementación de un modelo básico	44
Implementación de políticas de asignación	4
Implementación de retrasos	6
Implementación de los atraques	23
Probabilidad de fallo de los recursos	16
Disponibilidad de recursos	30
Prioridad de contenedores	20
Exportación de resultados	15
Mejoras varias	30
Creación de la clase simulador	4
Experimentación	36

Tabla 9.1: Horas de trabajo por módulo

¹PayScale.com - Sueldo medio de un especialista en simulación

Precio por hora	30€
Horas totales	271
Presupuesto total	8130€

Tabla 9.2: Presupuesto total

1870 Apéndice A

1871 Uso de la librería

1872 A.1. Simulación asistida

1873 Para la creación de simuladores de forma sencilla, se ha creado la clase *TerminalSimulator*.
1874 En el siguiente segmento de código se especifica su uso:

```
/**  
 * Crea el objeto simulador.  
 */  
TerminalSimulator simulator = new TerminalSimulator();  
  
/**  
 * Método de inicialización. Se debe invocar antes de cualquier otro método.  
 * Carga los principales parámetros del modelo según los argumentos que reciba.  
 * Existe la opción de llamarlo sin argumentos, y carga valores por defecto.  
 *  
 * @param nags Número de vehículos.  
 * @param nb Número de bloques de contenedores.  
 * @param nStacks Número de pilas en cada bahía de cada bloque.  
 * @param height Altura de las pilas.  
 * @param nBays Número de bahías en cada bloque.  
 * @param policy AllocationPolicy: política de asignación.  
 * @param availability ResourceAvailability: enum que designa la  
 * disponibilidad de los recursos.  
 * @param fallibility ResourceFallibility: enum que designa la  
 * falibilidad de los recursos.  
 */  
simulator.init(int nags, int nb, int nStacks, int height,  
                int nBays, AllocationPolicy policy,  
                ResourceAvailability availability, ResourceFallibility fallibility);  
*****  
simulator.init();
```

```

/**
 * Método de configuración para controlar cuántos contenedores se generan,
 * y su distribución. Ya es llamado por init con valores por defecto,
 * pero se puede llamar manualmente para valores personalizados.
 *
 * @param landPer Porcentaje de contenedores con ruta terrestre.
 * @param seaPer Porcentaje de contenedores con ruta marítima.
 * @param totalContainers Número de contenedores aproximado a generar.
 * @param arrivingPer Porcentaje de contenedores que serán de llegada.
 * @param outgoingPer Porcentaje de contenedores que serán de salida.
 * @param relocPer Porcentaje de contenedores que serán de reubicación.
 * @throws IllegalStateException Si no se llama a init antes.
 */
simulator.configure(double landPer, double seaPer, int totalContainers,
                    double arrivingPer, double outgoingPer, double relocPer);

/**
 * Ejecuta la simulación una única vez el tiempo que se especifica
 * como parámetro.
 */
simulator.singleRun(long time);
simulator.singleRun();           // por defecto una semana de tiempo

/**
 * Exporta los resultados de la última ejecución.
 */
simulator.gatherResults();

/**
 * Ejecuta singleRun diez veces, exporta los resultados de cada
 * ejecución. Calcula la media para todos esos datos y la exporta.
 */
simulator.run(long time);
simulator.run();                 // por defecto una semana de tiempo

/**
 * Permite ejecutar run múltiples veces combinando los datos recibidos
 * en los arrays que se reciben como parámetros. Combina todas las opciones
 * de cada array con todas las opciones del resto de arrays.
 * Arrays muy largos podrían resultar en demasiado tiempo de ejecución.
 *
 * @param simulationTime Tiempo de simulación.
 * @param agentsToGenerate Número aproximado de contenedores a generar.
 * @param nAGV Array con diferentes posibilidades para el número de vehículos.
 * @param nBlocks Array con posibilidades para el número de bloques.
 * @param policies Array con las posibles políticas de asignación.
 */
simulator.runSet(long simulationTime, int agentsToGenerate, Integer[] nAGV,
                  Integer[] nBlocks, AllocationPolicy[] policies);

```

1875 A.2. Simulación manual

1876 Para casos en los que es necesario mayor control sobre los diferentes parámetros del
 1877 modelo, se puede utilizar directamente la clase *TerminalModelBuilder*. En el siguiente
 1878 fragmento de código se resumen los principales métodos. Consultar la documentación de
 1879 la librería para obtener información más detallada.

```
// crea el objeto para el modelo, y el controlador de tiempo
// el controlador especifica el tiempo de simulación (604800 por ejemplo)
TerminalModelBuilder builder = new TerminalModelBuilder(...);
TimeManager timeManager = new TimeManager(builder, 604800);

// construye el modelo
builder.build();
builder.setTimeManager(timeManager);

// crea el objeto simulador. Hay que poner el tiempo en segundos
SimulatorEngine simulator = new SimulatorEngine();
simulator.setRootAgent(builder.getRootAgent());
simulator.setTimeUnit(TimeUnits.TIME_UNIT_SECOND);
simulator.setStopTime(builder.getTime().getStopTime());

// ejecuta la simulación
simulator.run();

// exporta los datos
TerminalRootAgent rootAgent = builder.getRoot();
rootAgent.loadSeaLinkData();
rootAgent.loadCranesData(builder);
rootAgent.loadAGVData(builder);
rootAgent.toJSONfile();

// resetea y reconstruye antes de volver a ejecutar
builder.reset();
builder.build();
```

1880 Apéndice B

1881 Códigos destacados

1882 Todos los códigos que figuran en este apéndice fueron escritos por el autor de esta
1883 memoria y trabajo de fin de grado: Óscar Darias Plasencia. Es importante tener en cuenta
1884 que muchas de las clases utilizadas pertenecen a la librería de simulación.

1885 B.1. Inicialización de contenedores (`Container.java`)

```
/**  
 * Initializes some fundamental data of the container.  
 *  
 * @param context TerminalModelBuilder: the simulation context,  
 *                 a reference to the model builder.  
 * @param route ContainerRoute: the route that the container is going to  
 *                 follow (land or sea).  
 * @return The adequate Function to be applied that Container object for  
 *                 its initialization.  
 */  
public static Function<TerminalModelBuilder, ContainerRoute> init(TerminalModelBuilder context,  
                      ContainerRoute route) {  
    return (Object agent) -> {  
        Container container = (Container) agent;  
        container.context = context;  
        container.setContainerRoute(route);  
        container.setCreationTime(context.getTime().getRealTime());  
        if (agent instanceof ArrivingContainer) {  
            container.setThroughLink(route);  
            container.setTerminalPosition  
                (new TerminalPosition(TerminalPosition.getArea(route),  
                                      container.getThroughLink()));  
        } else if (agent instanceof OutgoingContainer) {  
            container.setThroughLink(route);  
            container.setFromBlock();  
        } else {  
            // then it is a RelocatingContainer  
            container.setFromBlock();  
            context.getAllocationPolicyFunction().apply(container);  
        }  
        if (context.getTime().isSimulationTime())  
            context.addGeneratedAgent();  
        return true;  
    };  
}
```

1886 B.2. Control de atraques (Berth.java)

1887 Control de generaciones

```
/*
 * Function that determines if the generation of a
 * container is accepted or not.
 */
private final Function<Object> acceptGeneration = (object) -> {

    // check if there are still agents to be generated
    // if not, then destroy the associated Source block
    // and return false
    if (object instanceof ArrivingContainer
        && arrivingContainers >= agentsToStore) {
        tearDown(storeSource);
        return false;
    } else if (object instanceof OutgoingContainer
        && outgoingContainers >= agentsToLeave) {
        tearDown(collectSource);
        return false;
    }

    // if this is the first generation, create an event
    // that will restrict further ones
    // a ship has arrived and there should not be any more
    // generations until it leaves and another one arrives
    if (processingAgents == 0) {
        createRestrictGenerationsEvent();
        // save the time when the ship arrived
        Variable<AgentVariables.NonData, Long> arrival =
            new Variable<>(AgentVariables.NonData.LAST_SHIP_ARRIVED);
        arrival.setValue(context.getTime().getInnerTime());
        addVariable(arrival);
    } else if (processingAgents > agentsToStore + agentsToLeave)
        // if there have already been generated enough containers
        return false;

    // set the throughLink property to this link
    // and initialize the container
    Container container = (Container) object;
    container.setThroughLink(berthNumber.getValue());
    Container.init(context, ContainerRoute.BY_SEA).apply(object);

    // keep track of the generated agents
    // and save the information of the new one but just if
    // we are in simulation time
    processingAgents++;
    if (object instanceof ArrivingContainer) {
        arrivingContainers++;
        if (context.getTime().isSimulationTime())
            requestedArrivals
                .setValue(requestedArrivals.getValue() + 1);
    } else {
        outgoingContainers++;
        if (context.getTime().isSimulationTime())
            requestedCollections
                .setValue(requestedCollections.getValue() + 1);
    }
}
```

```

    }
    return true;
};

```

1888 Creación dinámica de un Source

1889 Ejemplo con el *Source* que genera contenedores de llegada o almacenamiento.

```

// manually create the generations cyclic event
Event generationCyclicEvent = new EventCyclic();
generationCyclicEvent.setRelative(true);

// create the object itself
storeSource = new Source(ArrivingContainer.class);
storeSource.setName(SimulationAgentNames.STORAGE_SEA_NAME);
storeSource.setMaxGenerations(Integer.MAX_VALUE);
// make sure it does not generate more agents than
// the desired number
storeSource.setMaxAgentsToGenerate(agentsToStore - arrivingContainers);

// define how many agents will be generated per generation
storeSource.setAgentsPerGeneration(seaAgentsPerGeneration);

// set the cyclic event as the source's event
storeSource.setEvent(generationCyclicEvent);

// add it to the root agent
context.getRoot().addAgent(storeSource);

// connect it
storeSource.getOut().connect(portForStore);

// set the acceptGeneration Function as its
// onGenerated component, so it is applied every
// time an agent is generated
storeSource.setOnGenerated(acceptGeneration);

```

1890 B.3. Disponibilidad de recursos

```

/****************************************************************************
***** TerminalResource.java *****
***** Abstract class for defining resources *****
****/


/**
 * This method creates the events that will set the resource unavailable, based on
 * its availability value. Distributes the time randomly,
 * using between 1 and 5 time intervals.
 */
public void calculateAvailabilities() {
    if (availability.getValue() == ResourceAvailability.TOTAL) return;

    Random rand = new Random();
    int nIntervals = rand.nextInt(5) + 1;
    Double unavailabilityInterval = ResourceAvailability
        .getUnavailabilityIntervals(availability.getValue(),
            context.getTime().getSimulationTime(), nIntervals);
    Double variation = unavailabilityInterval / 20;
    variation = rand.nextDouble() * (variation * 2 + 1) - variation;
    double intervalDuration = context.getTime().getSimulationTime() / nIntervals;
    for (int i = 0; i < nIntervals; i++) {
        double start = rand.nextDouble() * intervalDuration
            + intervalDuration * i + variation;
        programUnavailabilityEvent(start, unavailabilityInterval);
    }
}

/**
 * Method that programs an event that will set the resource unavailable.
 *
 * @param start ABSOLUTE starting time of the event.
 * @param duration Time that the resource will remain out of service.
 */
private void programUnavailabilityEvent(Double start, double duration) {
    // create the event action
    final Consumer<Event> unavailabilityEventAction = (e) -> {
        // the resource might be reserved or seized
        // in that case, add a new variable to it to remember it
        // the Variable class is defined in the simulation library
        // and it is normally used for defining agent properties
        if (!this.isIdle()) {
            Variable<AgentVariables.NonData, Double> pending =
                this.getVariable(AgentVariables.NonData.PENDING_UNAVAILABILITY);
            if (pending != null && pending.getValue() != null) {
                pending.setValue(pending.getValue() + duration);
            } else {
                pending = new Variable<>(AgentVariables.NonData.PENDING_UNAVAILABILITY);
                pending.setValue(duration);
                this.addVariable(pending);
            }
        } else {
            // set the resource unavailable
        }
    }
}

```

```

        this.setAvailable(false);

        // Set the resource available again after the specified time
        programAvailabilityEvent(duration);
        this.addOutOfServiceTime((long) duration);
    }
};

// program the event itself
// the EventProgrammer class just makes programming events easier
EventProgrammer.newAbsolute(context.getRoot(),
    context.getTime().getInnerTime(start.longValue()), unavailabilityEventAction);
}

double time) {
    EventProgrammer.newRelative(context.getRoot().getEngine(), time,
        (e) -> this.setAvailable(true));
}

/*****
***** TerminalModelBuilder.java *****
***** Class where the model is defined *****
*****/

double delay = 0.0;
    Variable<AgentVariables.NonData, Double> pending =
        resource.getVariable(AgentVariables.NonData.PENDING_UNAVAILABILITY);
    if (pending != null && pending.getValue() != null) {
        delay = pending.getValue();
        resource.addOutOfServiceTime((long) delay);
        pending.setValue(null);
    }

    // create an event to notify that the resource has been released
    EventProgrammer.newRelative(rootAgent.getEngine(), delay + 0.001,
        (e) -> resource.released());

    return delay;
};

```

1891 B.4. Políticas de asignación

```

/****************************************************************************
***** AllocationPolicy.java *****
***** Enum type with the allocation policies *****
****/


/**
 * Enum type that defines a set of allocation policies.
 *
 * @author Óscar Darias Plasencia
 */
public enum AllocationPolicy {

    /**
     * Predefined Function that redirects the agent to a random output.
     * It is meant to be used to redirect a container to a random block.
     * If the container already had an assigned block, uses it.
     */
    RANDOM((agent) -> {
        Container container = (Container) agent;
        TerminalModelBuilder builder = container.getContext();
        if (container.getToBlock() == null) {
            Random rand = new Random();
            int output = rand.nextInt(builder.getnBlocks());
            builder.getContainerBlocks().get(output).increaseQueue();
            container.setToBlock(output);
        }

        container.setPosition
            (builder.getContainerBlocks()
                .get(container.getToBlock()).askForPosition());
        return true;
    }),

    /**
     * Redirects the container to the block with less elements in its queue.
     */
    SMALLEST_QUEUE_BLOCK((agent) -> {
        Container container = (Container) agent;
        TerminalModelBuilder builder = container.getContext();
        RANDOM.function.apply(container);
        int assigned = container.getToBlock();
        for (int i = 0; i < builder.getnBlocks(); i++) {
            if (builder.getContainerBlocks().get(i).getCurrentQueue() <
                builder.getContainerBlocks().get(assigned).getCurrentQueue() - 1)
                assigned = i;
        }
        container.setBlock(assigned);
        container.setPosition(builder.getContainerBlocks().get(assigned).askForPosition());
        return true;
    }),
}

```

```


    /**
     * Predefined Function that defines an allocation policy.
     * Redirects the container to the container block that has the crane
     * that has been unused for the most time.
     */
LAZIEST((agent) -> {
    Container container = (Container) agent;
    TerminalModelBuilder builder = container.getContext();
    RANDOM.function.apply(container);
    int assigned = container.getToBlock();
    for (int i = 0; i < builder.getnBlocks(); i++) {
        if (builder.getContainerBlocks().get(i).getCrane().getUsageTime() <
            builder.getContainerBlocks().get(assigned).getCrane().getUsageTime())
            assigned = i;
    }
    container.setToBlock(assigned);
    container.setPosition(builder.getContainerBlocks().get(assigned).askForPosition());
    return true;
}),


    /**
     * Predefined Function that defines an allocation policy.
     * Redirects the containers to the emptiest container block.
     */
EMPTIEST((agent) -> {
    Container container = (Container) agent;
    TerminalModelBuilder builder = container.getContext();
    RANDOM.function.apply(container);
    int assigned = container.getToBlock();
    for (int i = 0; i < builder.getnBlocks(); i++) {
        if (builder.getContainerBlocks().get(i).getNumberOfContainers() <
            builder.getContainerBlocks().get(assigned).getNumberOfContainers())
            assigned = i;
    }
    container.setToBlock(assigned);
    container.setPosition(builder.getContainerBlocks().get(assigned).askForPosition());
    return true;
}),


    /**
     * Redirects the container to its closest block.
     * If it is a RelocatingContainer, then moves it to the block on the right
     * or the block on the left, depending on which one has the smallest queue.
     * Not really a good idea, because a few blocks would remain mostly empty,
     * unless there were too many relocations.
     */
CLOSEST_BLOCK((agent) -> {
    Container container = (Container) agent;
    TerminalModelBuilder builder = container.getContext();
    int assigned;
    if (agent instanceof RelocatingContainer) {
        assigned = container.getFromBlock();
        int leftBlock = container.getFromBlock() - 1;
        int rightBlock = container.getFromBlock() + 1;
        if (leftBlock >= 0 && leftBlock < builder.getnBlocks())
            assigned = leftBlock;
        if (rightBlock >= 0 && rightBlock < builder.getnBlocks()) {
            if (builder.getContainerBlocks().get(assigned).getCurrentQueue() >


```

```
        builder.getContainerBlocks().get(rightBlock).getCurrentQueue()
        || assigned == container.getFromBlock())
    assigned = rightBlock;
}
} else {
    assigned = builder.getExitBlock(container.getContainerRoute(),
        container.getThroughLink());
}
container.setToBlock(assigned);
builder.getContainerBlocks().get(assigned).increaseQueue();
container.setPosition(builder.getContainerBlocks().get(assigned).askForPosition());

return true;
});

/**
 * Function property.
 */
public final Function<Agent, Boolean> function;

/**
 * Enum constructor.
 */
private AllocationPolicy(Function<Agent, Boolean> describe) {
    this.function = describe;
}
}
```

Bibliografía

1892

- 1893 [1] Martín Jorge Agüero and Luciana C. Ballejos. Resolución más eficiente de dependencias Java. In *XXI Congreso Argentino de Ciencias de la Computación (Junín, 2015)*, 2015.
- 1894
- 1895
- 1896 [2] Peter J. Bentley and Jonathan P. Wakefield. Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In *Soft computing in engineering design and manufacturing*, pages 231–240. Springer, 1998.
- 1897
- 1898
- 1899 [3] Andrei Borshchev, Yuri Karpov, and Vladimir Kharitonov. Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems*, 1900 18(6):829–839, 2002.
- 1901
- 1902 [4] Andrew T. Crooks and Alison J. Heppenstall. Introduction to agent-based modelling. In *Agent-based models of geographical systems*, pages 85–105. Springer, 2012.
- 1903
- 1904 [5] Jay Wright Forrester. Industrial dynamics. *Journal of the Operational Research Society*, 48(10):1037–1041, 1997.
- 1905
- 1906 [6] Carlos A. García, Edwin García, and Fernando Villada. Implementación del algoritmo 1907 evolutivo multi-objetivo de frente de pareto (SPEA) para la planeación de sistemas 1908 eléctricos de distribución incluyendo huecos de voltaje. *Información tecnológica*, 1909 26(5):155–168, 2015.
- 1910
- 1911 [7] Maxim Garifullin, Andrei Borshchev, and Timofei Popkov. Using AnyLogic and agent-based approach to model consumer market. In *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*, pages 1–5, 2007.
- 1912
- 1913 [8] Nicoletta González Cancelas. *Metodología para la determinación de parámetros de diseño de terminales portuarias de contenedores a partir de datos de tráfico marítimo*. PhD thesis, Caminos, 2007.
- 1914
- 1915
- 1916 [9] Wilfredo Guaita. *Desarrollo de un modelo de simulación para ensayar políticas operacionales en cadenas de suministros de PYMES transformadoras*. PhD thesis, 1917 Industriales, 2008.
- 1918
- 1919 [10] David Mautner Himmelblau and Kenneth B. Bischoff. *Análisis y simulación de procesos*. Reverté, 1992.
- 1920
- 1921 [11] Luis R. Izquierdo, José M. Galán, José I. Santos, and Ricardo Del Olmo. Modelado de 1922 sistemas complejos mediante simulación basada en agentes y mediante dinámica de 1923 sistemas. *EMPIRIA. Revista de Metodología de las Ciencias Sociales*, (16), 2008.

- 1924 [12] Norm Matloff. *Introduction to discrete-event simulation and the SimPy language.*
 1925 *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on*
 1926 *August, 2(2009), 2008.*
- 1927 [13] Victor Eusebi Muñoz Cinca. *Optimización de la producción en una terminal marí-*
 1928 *tima de contenedores. Umbrales y punto de equilibrio.* Universitat Politècnica de
 1929 Catalunya, 2008.
- 1930 [14] Vilfredo Pareto. *Manuale di economia politica*, volume 13. Societa Editrice, 1906.
- 1931 [15] Juan Pavón Mestras, Adolfo López Paredes, José Manuel Galán Ordax, et al. Modelado
 1932 basado en agentes para el estudio de sistemas complejos. *Novática.* 2012, n. 218, p.
 1933 13-18, 2012.
- 1934 [16] Miquel Àngel Piera. *Modelado y simulación. Aplicación a procesos logísticos de*
 1935 *fabricación y servicios*, volume 118. Universitat Politècnica de Catalunya. Iniciativa
 1936 Digital Politècnica, 2004.
- 1937 [17] Carlos Rúa Costa. Los puertos en el transporte marítimo. 2006.
- 1938 [18] R. Sapiña, I. Yarza, A.M. Martín-Soberón, A. Monfort, and N. Monterde. Herramientas
 1939 de simulación en terminales portuarias de contenedores, 2012.
- 1940 [19] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer.
 1941 Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization*
 1942 and Computer Graphics, 23(1):341–350, 2017.
- 1943 [20] Robert Shannon and James D. Johannes. Systems simulation: the art and science.
 1944 *IEEE Transactions on Systems, Man, and Cybernetics*, (10):723–724, 1976.
- 1945 [21] Enrique Eduardo Tarifa. Teoría de modelos y simulación. *Facultad de Ingeniería,*
 1946 *Universidad de Jujuy*, 2001.
- 1947 [22] Christian Von Lücken, Augusto Hermosilla, and Benjamín Barán. Algoritmos evoluti-
 1948 vos para optimización multiobjetivo: un estudio comparativo en un ambiente paralelo
 1949 asíncrono. In *X Congreso Argentino de Ciencias de la Computación*, 2004.