UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

Trabajo Fin de Grado

CURSO 2016/17

DESARROLLO DE UN MODELO DE SIMULACIÓN PARA FABRICACIÓN BASADO EN UNITY3D

Grado en Ingeniería en Tecnologías Industriales

ALUMNO

Javier Pernas Álvarez

TUTOR

Diego Crespo Pereira

FECHA

JULIO 2017

Resumen

Desarrollo de un modelo de simulación para fabricación basado en Unity3D

En el presente trabajo se lleva a cabo el desarrollo de una serie de librerías que permiten la implementación de un modelo de eventos discretos en el motor de videojuegos Unity3D. Para ello, primeramente se crea el código necesario para gobernar el comportamiento y la interacción de una serie de elementos que constituyen la simulación de un proceso genérico. Posteriormente, se diseña una interfaz gráfica de usuario que permita al mismo interactuar con la simulación y tener control sobre la misma. A continuación, se define un caso de demostración en el que se realiza una modelización conceptual y se desarrolla un entorno virtual que simula el real. Una vez implementado el modelo en Unity3D, se estudian las potencialidades que pueden ofrecer estos motores frente un *software* comercial de eventos discretos como FlexSim. Finalmente, se experimenta con la aplicación creada en entornos de realidad virtual.

Resumo

Desenvolvemento dun modelo de simulación para fabricación baseado en Unity3D

No presente traballo lévase a cabo o desenvolvemento dunha serie de librerías que permiten a implementación dun modelo de eventos discretos no motor de videoxogos Unity3D. Para iso, primeiramente créase o código necesario para gobernar o comportamento e a interacción dunha serie de elementos que constitúen a simulación dun proceso xenérico. Posteriormente, deséñase unha interfaz gráfica de usuario que permita ó mesmo interactuar coa simulación e ter control sobre a mesma. De seguido, defínese un caso de demostración no que se realiza unha modelización conceptual e se desenvolve un entorno virtual que simula ó real. Unha vez implementado o modelo de Unity3D, estúdanse as potencialidades que poden ofertar estos motores fronte a un software comercial de eventos discretos como FlexSim. Finalmente, experiméntase coa aplicación creada en entornos de realidade virtual.

Abstract

Simulation model development for manufacturing based on Unity3D

Within the following Bachelor thesis, several programming libraries have been developed with the aim of implementing a discrete-simulation-model in the Unity3D game engine. Initially, there is the need of creating the code which controls the behaviour and interaction of a determined number of elements, conforming a generic manufacturing process. Afterwards, a graphic user interface (GUI) is designed in order to allow the user to interact and manage the model itself. Furthermore, a demonstration case is developed, including the conceptual modelling and the virtual environment. Once the Unity3D engine model is implemented, a comparison between both models (Unity and FlexSim) is performed, emphasizing the advantages provided by the game engines. Finally, we proceed to the experimentation with virtual reality environments

.

UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

TRABAJO FIN DE GRADO

CURSO 2016/17

DESARROLLO DE UN MODELO DE SIMULACIÓN PARA FABRICACIÓN BASADO EN UNITY3D

Grado en Ingeniería en Tecnologías Industriales

Documento

MEMORIA

Índice

Índice de figuras

Índice de tablas

Capítulo 1

1.1. Introducción

En la actualidad, vivimos en un mundo con continuos cambios en todas las facetas y aspectos de nuestra vida, caracterizada tanto en lo personal como en lo profesional por el uso continuo de las tecnologías de la información, desarrolladas por una inmensa industria, causa y a la vez resultado de la ya conocida globalización.

En este contexto, son la flexibilidad y la adaptabilidad entre otros, dos conceptos fundamentales en el éxito de cualquier empresa. Técnicas de producción como *Lean Manufacturing* o *Just in Time* se aplican desde hace ya años en múltiples e importantes compañías que comparten generalmente el mismo objetivo: la introducción rápida de nuevos productos y servicios, con la meta de atraer nuevos clientes y mejorar el servicio a los actuales. Sin embargo, esta notable empresa conlleva asimismo poseer un sistema productivo lo más flexible posible y con gran capacidad para adaptarse a las necesidades que el mercado actual tiene; esto es, tiempos de entrega de producto y ciclos de producción más cortos, económicos y con la calidad requerida. Para todo ello es fundamental la ciencia, la tecnología y la innovación, conceptos intrínsecamente relacionados.

Con todo lo anterior, la cuestión que se plantea es cómo medir la flexibilidad y la adaptabilidad de un sistema productivo. Para poder avanzar hacia un objetivo, es requisito indispensable conocer con detalle en qué situación se halla uno, a través de parámetros y variables de referencia que permitan generar marcos de comparación en los que evaluarse. En este sentido, la simulación de procesos se erige como una de las herramientas responsables en dar respuesta a este aspecto.

Es innegable que las simulaciones de productos, materiales y procesos de producción están a la orden del día en países que poseen los más sólidos sistemas productivos. No obstante, actualmente se está empezando a profundizar en un uso más extensivo y completo de la simulación. La idea es que estas simulaciones puedan revelar en tiempo real todos aquellos datos necesarios con el fin de reflejar el mundo real – un proceso de producción dado – en un modelo virtual que incluya, por tanto, todos los elementos que conforman el proceso en cuestión.

Este desarrollo de la propia simulación de procesos como técnica de estudio y optimización está inherentemente ligado al avance de la industria en general, y a las tecnologías 3D en particular. El *software* de eventos discretos dio salto cualitativo incorporando la visualización en 3 dimensiones de los procesos. El concepto sobre el cual se asienta todo esto no es únicamente la actualización: cuanto más cercano sea en todos los sentidos el proceso modelado al real, menor esfuerzo será necesario a la hora de estudiar y practicar mejoras en el mismo. Sin embargo, las posibilidades relativas a la versatilidad, la parte gráfica y al rendimiento que proporciona este *software* son aún muy limitadas, quedando por tanto mucho margen de mejora.

Por otra parte, en los últimos años, un sector que está teniendo un progreso extraordinario en estos aspectos es el de los videojuegos. Los conocidos como motores de videojuegos son la base para la programación de videojuegos y proporcionan una herramienta de gran interés para la simulación de procesos, ya que proporcionan funcionalidades básicas de un simulador como la gestión del tiempo, el entorno virtual, el desarrollo de interfaces gráficas y otras que, no siendo comunes en los simuladores de eventos discretos, ofrecen capacidades nuevas e interesantes como la detección de colisiones, los motores físicos o la posibilidad de integración con tecnologías de realidad virtual . Si bien el objetivo para el que son diseñados es el propio desarrollo de juegos, su versatilidad permite realizar proyectos virtuales de diferente índole, debido en gran parte a la libertad que ofrece la programación.

Por tanto, la cuestión a la que pretende dar respuesta este trabajo es evaluar la factibilidad y demostrar el uso de un motor gráfico como simulador de eventos discretos, aprovechando las

ventajosas capacidades que brindan, y teniendo en cuenta al mismo tiempo que su diseño no está orientado a este tipo de proyectos.

1.2. Objetivo del proyecto

La idea principal de este trabajo parte de dos elementos fundamentales: por un lado, las potencialidades gráficas y de procesado que ofrecen los motores de videojuegos, las cuales serán analizadas en el *Capítulo 3*; y por otro, el creciente uso de *software* de simulación de eventos discretos con entornos 3D (*Capítulo 2*). Por tanto, el **objetivo fundamental** del proyecto es desarrollar un prototipo de un simulador de eventos discretos aprovechando las funcionalidades que ofrecen los motores de videojuegos, realizando a partir del mismo un estudio comparativo de las ventajas que presenta frente a un *software* de eventos discretos comercial en tres dimensiones

No obstante, el objetivo principal se puede desagregar en una serie de hitos, algunos de los cuales surgen en el propio curso del trabajo, y que se exponen a continuación:

- Desarrollo de unas librerías genéricas que modelen el comportamiento de los principales elementos existentes en un modelo de simulación. El funcionamiento de estas librerías será de acuerdo con el cumplimiento de los eventos de una lista según intervalos de tiempos discretos.
- Aplicación de las librerías desarrolladas a un caso concreto de demostración a través de un motor de videojuegos en 3 dimensiones.
- Desarrollo, en la medida de lo posible, del entorno gráfico concerniente al caso.
- Experimentación con entornos de realidad virtual a partir del trabajo desarrollado.
- Realización de la comparativa entre ambas plataformas anteriormente descritas, objetivo principal del proyecto.

1.3. Recursos del proyecto

Para la ejecución del proyecto, se elige el motor de videojuegos Unity3D (en adelante, Unity), especialmente por las altas posibilidades que ofrece su versión gratuita así como por la gran cantidad de recursos gratuitos disponibles en la red. Junto a ello, se emplearán las herramientas de SketchUp, Photoshop y Blender para el modelado del entorno y los edificios y de los elementos que conforman la simulación. Como lenguaje de programación se emplea C#, un lenguaje orientado a objetos perteneciente a la plataforma .NET y, como entorno de programación, Visual Studio.

Con respecto al caso de demostración, la elección del proceso tiene como requisito primordial la existencia de un modelo de simulación de eventos discretos en 3 dimensiones - en este caso, en el *software* de simulación FlexSim -, el cual se empleará para la comparación. Para ello, se escoge un proceso cuyo modelo de FlexSim es proporcionado por la Unidad Mixta de Investigación de la Universidade da Coruña, y que se explica en *4.4 Descripción del caso de demostración*. Del mismo se realizará una descripción breve de las etapas que lo conforman, de cara a entender las diferentes partes que conformarán el modelo de Unity. Asimismo, se emplearán los resultados de la simulación de FlexSim para definir los tiempos de las diferentes actividades del modelo, que en ningún caso será igual al real.

Figura 1. Logo del motor de videojuegos Unity3D. Fuente: www.unity3d.com.

Capítulo 2

Tras haber establecido el objetivo del proyecto, debemos definir primeramente el concepto de simulación de manera genérica, situándola en un marco contextual adecuado. Será fundamental, a la vista del proyecto, describir qué es y qué conlleva un modelo de simulación. Posteriormente, definiremos el tipo de simulación que se va a llevar a cabo, aportando algunas de las ventajas que posee, pero no sin dejar de considerar qué inconvenientes o malas praxis puede llevar asociado. Asimismo, se comentan brevemente las etapas de una simulación para, por último, introducir en este capítulo el concepto de visualización 3D.

2.1. Contexto actual

Así como la invención de la máquina de vapor en el siglo XIX, la producción en serie a principios del siglo XX o la automatización de la industria en la década de 1970, originaron un cambio revolucionario en la industria en general – cambio que recibió el nombre de Revolución Industrial –, las transformaciones actuales de las que somos testigos nos sitúan ya en el medio de una cuarta ola de avance tecnológico: “La Industria 4.0” o “La Cuarta Revolución Industrial”.

En esta gran metamorfosis, donde Internet tiene el papel protagonista, sensores, máquinas, materiales, productos y sistemas de información estarán conectados a lo largo de la cadena de valor más allá de una empresa o proyecto único. Estos sistemas interconectados e *inteligentes* tendrán la capacidad de interactuar a través de protocolos estándar y analizar información para, con ello, predecir fallos, auto–configurarse y adaptarse a cualquier cambio.

Junto con la innovación tecnológica, la estructura de organización de los medios de producción se ha visto sometida a grandes transformaciones en el pasado para afrontar los cambios en los mercados. La producción industrial tiene su origen en el paso de la fabricación artesanal a la producción en masa, junto con la aparición de conceptos como estandarización. En un mercado donde la producción se erige como el gran cuello de botella, la estructura de organización se centró en incrementar enormemente la producción y la productividad, despreocupándose por completo de un entorno cambiante y, sobretodo, de las necesidades del consumidor.

Posteriormente, con la saturación de los principales mercados mundiales, estos mismos se convirtieron en los denominados *buyer’s markets*, donde al superar la oferta a la demanda, es la figura del consumidor la que adquiere realmente importancia. Como consecuencia, las empresas manufactureras comienzan a basar su producción en la diferenciación del producto (Lasi Hans-Georg Kemper, Peter Fettke, Thomas Feld, & Michael Hoffmann, 2014). Asimismo, de cara a incrementar la competitividad y la productividad en un momento en que se convierte en necesidad fundamental la fabricación de distintas variedades de cualquier producto, el *Lean Manufacturing* comienza a extenderse como técnica de producción, al eliminar los despilfarros y las colas de espera con la regla de los 5 segundos (Crespo Pereira, 2013).

No obstante, y ya más cerca de la actualidad, la demanda creciente de productos cada vez más adaptados al consumidor junto con el decrecimiento simultáneo de los ciclos de vida de los mismos, exige estructuras de organización mucho más complejas e inteligentes que las empleadas hasta el momento (Brettel, Friederichsen, & Keller, 2014). Son los sistemas distribuidos los que ofrecen el punto de partida para hacer frente a la creciente complejidad de las compañías y, especialmente, para la llamada cibernética de control y gestión – *Management Cybernetics* –, la cual incorpora los sistemas inteligentes de los que se habló en anteriormente. Llegados a este punto, Internet se ha establecido como el instrumento sumamente poderoso para controlar sistemas distribuidos y tecnologías como la Identificación por Radiofrecuencia (RFID, por sus siglas en inglés), que pueden ser utilizadas para registrar y hacer un seguimiento de cada producto individual a través de la cadena de valor, una parte fundamental de lo que será la “Smart Factory”.

En definitiva, la Industria 4.0 hará posible obtener procesos considerablemente más rápidos, flexibles y eficientes – cualidades que destacábamos en la introducción – que permitirán producir entregables de mayor calidad a costes reducidos. Todo ello conllevará un incremento de la productividad, cambios en la economía, fomentará el crecimiento industrial y modificará el perfil del trabajador deseado – cambiando en última instancia la competitividad de empresas y regiones.

Figura 2. Los nueve pilares de La Cuarta Revolución Industrial. Fuente: BCG.

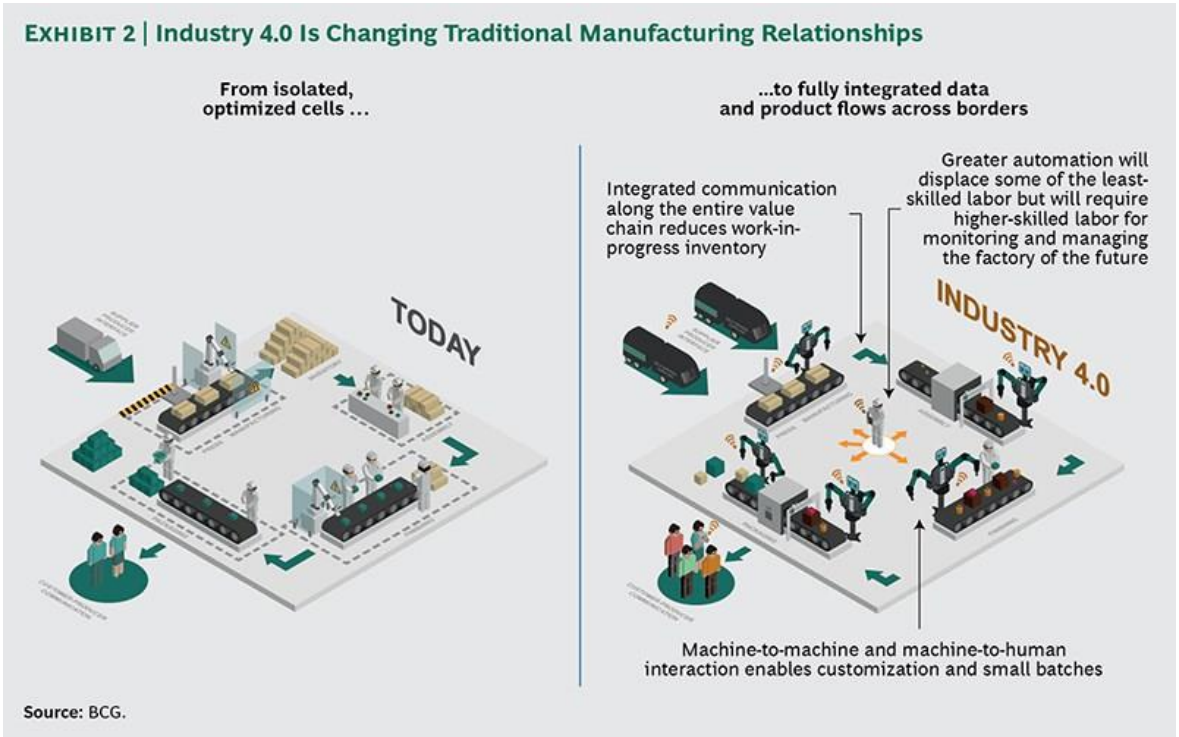

Figura 3. La Industria 4.0. Fuente: BCG.

Con mayor concreción, podemos definir nueve avances tecnológicos en los que se fundamenta la Industria 4.0: El “Big Data”, los Robots autónomos, Integración Vertical y

Horizontal de los sistemas de información, el conocido como “Internet de las Cosas”, la Ciberseguridad, “la Nube”, “Additive Manufacturing”, Realidad Aumentada y **Simulación** (Rüßmann, Lorenz, Gerbert, & Waldner, 2015). Y es esta última y su papel en esta revolución tecnológica lo que nos concierne en este trabajo.

2.2. Simulación de procesos

Los procesos de producción descansan, fundamentalmente, en cuatro factores: 1) datos disponibles y confiables, 2) pronósticos adecuados, 3) cadena de suministro rápida y 4) adecuada planeación de los inventarios y precios (Peralta Abarca, 2011).

En lo que se refiere a los dos primeros, hablamos de un entorno fuertemente dinámico y variable que afecta de manera directa al éxito de nuestra empresa. Los cambios en el suministro o en las características de materiales o máquinas, retrasos o adelantos en la llegada de productos, o cualquier cambio en el entorno que rodea el proceso, pueden tener un impacto significativo en nuestro sistema. De esta manera, es fundamental emplear estrategias o métodos de organización de la producción que proporcionen las herramientas necesarias para un diseño robusto de los sistemas.

No obstante, esa variabilidad también se encuentra intrínsecamente arraigada en los dos últimos factores mencionados. La eficiencia en las operaciones realizadas y procedimientos seguidos, las variaciones en tiempos de ciclo, la disponibilidad de las máquinas por fallos, o las lógicas de operación, pueden causar un impacto negativo en la capacidad efectiva de un sistema de producción y, por consiguiente, en los resultados de una empresa.

Hablamos por tanto de uno de los principales retos en la organización en la producción, una necesidad latente y cada vez más explícita en cualquier ámbito de la industria y que, como tal, requiere soluciones a la altura de las transformaciones que está sufriendo el conjunto de la misma.

Y es la simulación la encargada de cubrir esta necesidad. Ya habiendo avanzado alguna de sus potencialidades anteriormente, podemos definirla formalmente como *la imitación de las operaciones de un proceso o sistema real que conlleva la creación de un escenario artificial y el análisis del mismo para detectar los problemas que presenta el sistema operacional representado. Es una herramienta indispensable para resolver muchos de los problemas que se presentan en la realidad. La simulación es empleada para representar y analizar distintas alternativas de un sistema productivo, y finalmente ayudarnos a decidir cuál es la mejor de ellas* (Banks, II, & Barry, 2005).

2.3. El modelo de simulación

Ese escenario artificial del que nos habla Banks es el conocido como modelo de simulación. El modelo de simulación tiene la forma de un conjunto de suposiciones o hipótesis derivadas del funcionamiento del sistema simulado. Estos supuestos se expresan en relaciones matemáticas, lógicas y simbólicas entre entidades o ítems del sistema. No obstante, se ha de tener en cuenta las siguientes consideraciones derivadas de un concepto más general de modelo pero que igualmente son aplicables a nuestro caso[1]:

- Un modelo se desarrolla siempre a partir de una serie de aproximaciones e hipótesis y, consecuentemente, representa tan sólo parcialmente la realidad.
- Un modelo se construye para una finalidad específica y debe ser formulado para que sea útil a dicho fin.
- Un modelo tiene que ser, por necesidad, un compromiso entre simplicidad y la necesidad de recoger todos los aspectos esenciales del sistema de estudio.

Es especialmente importante estimar estas dos últimas apreciaciones para este trabajo. Más adelante podrá comprobar el lector la razón de ello.

[1] (Guasch, Piera, Casanovas, & Figueras, 2003)

Una vez simulado y validado, un modelo puede usarse, por ejemplo, para investigar distintas posibilidad de configuración del proceso real. De la misma manera, podemos saber hasta qué punto afecta un pequeño cambio en un determinado aspecto del sistema en el resto del mismo y en sus resultados.

Con todo ello, de forma genérica, la simulación de procesos podrá emplearse tanto como herramienta de análisis para predecir el efecto de cambios o de la variabilidad del entorno en sistemas reales, así como herramienta de diseño para predecir el comportamiento de nuevos sistemas en diferentes condiciones de funcionamiento.

2.4. La simulación de eventos discretos

El campo de la simulación engloba un amplio abanico de posibilidades y métodos. Por ello, en lo que se refiere a nuestro caso de aplicación, antes de todo, de cara a expresar rigurosamente el concepto de simulación de eventos discretos, es necesario caracterizar el tipo de sistemas al que se aplica y de los cuales surge este tipo de simulación.

2.4.1. Caracterización

Considérese el ejemplo de un manipulador debe recoger piezas diferentes de una misma caja y clasificarlas en distintas en función del tipo de pieza de que se trata. El manipulador es capaz de detectar a través de sensores el tipo de pieza que tiene a su alcance, así como su posición y orientación, información fundamental para que el robot sea capaz de distinguir cómo y dónde ha de colocar la pieza. Como comprenderá el lector, el tiempo necesario para completar un ciclo (colocar una pieza en su respectiva caja) dependerá de las características de la pieza o de su orientación o posición inicial y final entre otros. En caso de conocer todas las variables que afectan al proceso así como las ecuaciones que rigen la dinámica del manipulador, podríamos conocer con exactitud el tiempo que tardará en recoger y colocar el siguiente ítem. Tendríamos por tanto la posibilidad de desarrollar en un ordenador un modelo de simulación determinista que generaría unos outputs exactamente idénticos a los reales.

Sin embargo, si extrapolamos el ejemplo a un proceso mayor, como puede ser una cadena de producción en una planta, es muy probable que sea imposible controlar todas las variables que afectan al sistema – además de que resultaría costoso e ineficiente. Simplemente con pensar en la llegada del material necesario por parte del proveedor, no podríamos controlar con certeza si estamos ante un día con mucho tráfico, o que cierta máquina falle y que, por tanto, suponga un retraso de “x” minutos en el conjunto del proceso. Esto nos demuestra que en la realidad de un proceso de producción, estamos expuestos a múltiples variables sobre las cuales, a simple vista, no tenemos un control directo. En otras palabras, en este tipo de procesos, uno o varios inputs del sistema son de carácter aleatorio y, por tanto, lo serán también las variables calculadas a partir de ellos. Esto hace que la evolución de este tipo de sistemas deba estudiarse en términos probabilísticos, y que el modelo desarrollado se denomine estocástico.

De modo análogo, es necesario tener en cuenta que, como se ha dicho, el modelo de simulación ha de ser una representación lo más simplificada posible de las dinámicas de interés. Todo aquello que resulte ser superfluo al estudio o que no tenga un efecto en el modelo, deberá excluirse del mismo. Como en cualquier ámbito de la industria y la computación, nos atenemos a la máxima de compromiso entre coste, tiempo y esfuerzo. Por ello, el uso de modelos estocásticos para representar la secuencia de eventos de un sistema facilita las tareas de modelado y validación del modelo.

Paralelamente al tipo de modelo, el propio sistema objeto de la simulación será estudiado como un sistema orientado a eventos discretos. Esto es, aunque no se trate necesariamente de un sistema discreto en sí mismo, es decir, cuyas propiedades cambian únicamente en un cierto instante o secuencia de instantes permaneciendo constantes el resto del tiempo, será tratado como tal. El proceso de producción genérico considerado anteriormente que en realidad, es un proceso continuo y continuado en el tiempo, será considerado para nuestros intereses como un sistema discreto, asumiendo por tanto que los parámetros de interés

permanecen constantes entre instante e instante. Evidentemente, el tiempo será una variable fundamental en nuestra simulación, con lo que lo definiremos como un modelo dinámico.

De esta manera, podemos concretar con rigor y claridad el concepto de simulación de eventos discretos definiéndolo como una técnica informática de modelado de sistemas dinámicos, de carácter estocástico, y orientados a eventos discretos. Caracterizada por un control de la variable tiempo que le permite a la simulación avanzar en intervalos variables, nos permite capturar la aleatoriedad y operatividad del sistema.

2.4.2. Ventajas e inconvenientes

Aunque ya se han ido avanzando algunos de ellos, se exponen a continuación algunas de las ventajas que proporciona la simulación de eventos discretos para la mejora de procesos productivos:

- Análisis y desarrollo de nuevas estrategias, procedimientos de operación y organización, reglas de decisión y flujos de información sin necesidad de interrumpir en ningún caso el proceso real.
- Capacidad de modelado y análisis de sistemas complejos y de la interacción entre las variables de interés para la obtención de conocimientos y conclusiones no inmediatamente deducibles a partir del comportamiento de los elementos individuales.
- Diseño y estudio de nuevos procesos productivos, de *layouts*, o sistemas de transporte entre otros sin necesidad de adquirir ningún recurso físico. El mundo virtual nos permite responder a las cuestiones de "¿Qué pasaría si…?".
- Optimización de proceso productivo: Equilibrado de líneas, estudio de almacenes (necesidades, dimensionamiento de espacios…), reparto de cargas de trabajo, identificación de cuellos de botella, dimensionamiento de cintas transportadoras…, etc.

Sin embargo, a pesar de que la simulación aporta una importante ventaja competitiva a las empresas que la emplean, presenta asimismo ciertos problemas o inconvenientes:

- Dificultad en la interpretación de los resultados obtenidos y en la extracción de conclusiones de los mismos.
- Un estudio completo y eficaz puede ser en un principio costoso tanto en tiempo como en dinero.
- La adquisición de datos puede requerir en ocasiones más tiempo del planificado, debido principalmente a que un gran número de empresas no tiene controlados los tiempos de ciclos, de cambio y otros datos de su proceso productivo.

No obstante, frente a las posibles dificultades que puede presentar la simulación, en respuesta a este tipo de problemas, Banks y Gibson (1997) proporcionan una serie de reglas para evaluar cuando la simulación no es apropiada (Banks et al., 2005). Entre otras, por ejemplo, la simulación no debe de ser empleada en aquellos casos en que el problema pueda ser resuelto analíticamente, o bien por sentido común. Además, frente al problema del coste y la adquisición de datos, afirman que carecerá de sentido simular un proceso donde los costes exceden los ahorros previstos con el estudio, o bien, cuando no existen datos fundamentales disponibles. Esto puede disparar el presupuesto necesario para llevarla a cabo. Finalmente, en su última regla, remiten a lo comentado previamente en este trabajo sobre la simplicidad del modelo de simulación: si un sistema es demasiado complejo o es imposible de definir, la simulación no es recomendable.

2.4.3. Etapas de la simulación

Con todo lo anterior, pueden definirse una serie de pasos básicos a seguir a la hora de realizar una simulación. Existen diferentes modelos para representar los pasos de un proceso de simulación. El propio Banks cita en "Discrete Event System Simulation" a algunos como los de Gordon (1978), o Law and Kelton (2000). En nuestro caso, nos basaremos en el modelo descrito por Banks en el mismo libro para completar este punto. El autor, define un total de 12 pasos que vienen resumidos en la *Figura 4*. A continuación, se explica cada uno de ellos:

I. Formulación del problema. Toda simulación empieza con un enunciado del problema que se plantea. En el caso de que el enunciado venga dado por un cliente, el ingeniero deberá asegurarse primeramente que comprende completa y perfectamente el problema. Si, por el contrario, es el ingeniero el que realiza el enunciado, será muy importante que el cliente lo comprenda y esté de acuerdo con el planteamiento. A pesar de ello, la naturaleza del problema puede ser cambiante y, por tanto, es posible que el enunciado del mismo tenga que ser reformulado nuevamente a medida que el proceso de simulación progrese.
II. Planteamiento de los objetivos y plan general del proyecto. Los objetivos indican las cuestiones a las que ha de responder la simulación. Llegados a este punto, se tomará la decisión de si, a la vista de la formulación del problema y de sus objetivos, la simulación es la herramienta adecuada para realizar el trabajo. Asumiendo que se decide que la simulación es la metodología adecuada, se realiza la planificación del proyecto, que deberá contemplar los distintos escenarios que se deben estudiar. Asimismo, será necesario incluir el número de personas involucradas, el coste y el plazo tanto del estudio en general como de cada fase en particular.
III. Diseño del modelo conceptual. A partir del estudio del sistema real, deberá realizar el modelo conceptual. Sería la primera fase del modelo de simulación, donde se establecen las principales relaciones matemáticas y lógicas correspondientes a la estructura y los componentes del sistema. A partir de esta base, se irá desarrollando hasta completarlo, con la máxima ya mencionada de no hacerlo más complejo de lo necesario. Se recomienda en esta etapa la implicación del cliente, pues con ello se conseguirá aumentar la calidad de los resultados finales además de la confianza del cliente en el modelo.
IV. Toma de datos. En el mejor de los casos, el cliente tendrá recogidos todos los datos y se podrán introducir fácilmente en el modelo. No obstante, estamos ante una etapa clave. Existe la posibilidad de que el cliente no posea esos datos, o bien que los que tenga no sean los que se necesitan. Esto incurre en un aumento del tiempo necesario para desarrollar el modelo, así como del coste del trabajo. Por ello es tan fundamental la etapa II donde se planifica el proyecto. De hecho, la etapa III puede realizarse en paralelo con esta. Si no se tiene esto en cuenta, la viabilidad económica del proyecto podría verse afectada significativamente y, con ello, la rentabilidad del trabajo.
V. Realización de la simulación. Se vuelcan los datos recogidos en el modelo y se ejecuta.
VI. Verificación del modelo. Deberá realizarse con cuidado una completa revisión del modelo y de los datos introducidos comprobando que el modelo conceptual se ajusta al modelo virtual. Se trata de una etapa fundamental para el desarrollo correcto del estudio.
VII. Validación del modelo. En esta etapa se plantea la siguiente pregunta: ¿Puede el modelo virtual sustituir al modelo real para los propósitos del proyecto? A través de la calibración del modelo y del proceso iterativo de comparación de ambos sistemas, el real y el virtual, se reajustará el modelo desarrollado hasta que la convergencia entre ambos sea justificadamente aceptable.
VIII. Diseño de los experimentos. Para cada escenario contemplado inicialmente, será necesario introducir todos los datos que correspondan a dicho escenario y definir los parámetros a analizar. Es fundamental identificar qué información es realmente de nuestro interés.
IX. Realizar la simulación y analizar los datos.
X. ¿Más simulaciones? Será necesario realizar más simulaciones en aquellos casos en que los datos extraídos no sean coherentes o suficientes para alcanzar los objetivos del estudio.
XI. Informes y documentación. Los resultados de las distintas simulaciones deben ser entregados de la forma más clara y concisa posible al cliente. Así, el cliente podrá analizar el planteamiento de problemas, los datos de partida, los distintos escenarios planteados y, finalmente, los resultados obtenidos de cada uno de ellos. De esta manera, será posible obtener las conclusiones pertinente que resuelvan las cuestión

planteadas, tanto si se trata de un diseño de un proceso productivo, de un replanteamiento de uno ya existente..., etc.

XII. Implementación. En los casos en los que el modelo sea desarrollado como un programa que será reutilizado más adelante, deberá compilarse la documentación del propio programa constituida por todo aquello necesario para explicar cómo funciona. En nuestro caso, este paso podría ser de gran ayuda, especialmente para que el proyecto tenga una mayor rentabilidad económica. Y, de esta manera, el cliente contaría con una herramienta adaptada completamente a su proceso productivo y válida para realizar nuevas simulaciones, pruebas y, por tanto, nuevas tomas de decisión.

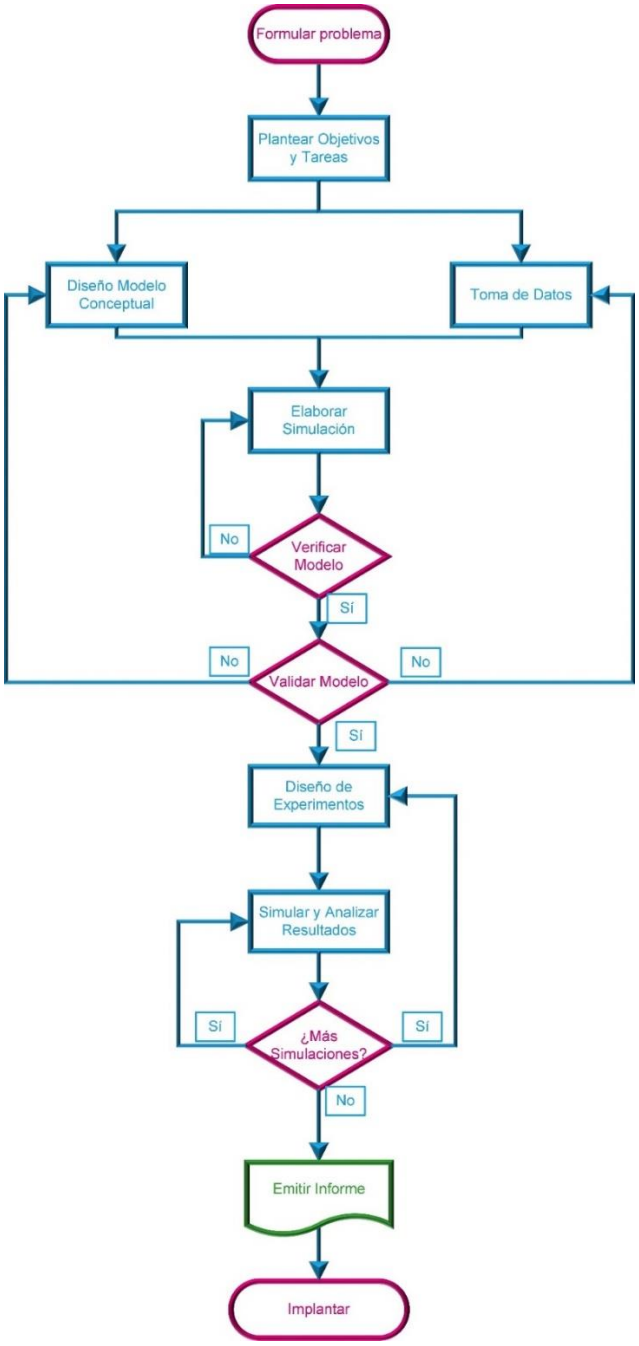

Figura 4. Etapas de la simulación. Fuente: elaboración propia a partir de (Castrilllón, 2008).

2.4.4. Visualización de la simulación

Finalmente, dado el trabajo a realizar en este proyecto, es fundamental dar una breve introducción a las técnicas de visualización.

Actualmente, dentro de la simulación de eventos discretos, la visualización se ha convertido en un componente crítico. La tan conocida frase “una imagen vale más que mil palabras” nos afirma una verdad tan clara como que el cerebro humano procesa las imágenes y los modelos de forma mucho más fácil y rápida que los números y los datos. De hecho, hoy en día, no podemos imaginar hacer una simulación sin ningún tipo de visualización que resulte de ayuda para la comunicación de los resultados, o para un mejor entendimiento del comportamiento del proceso en cuestión. No obstante, partiendo de lo evidente de la visualización 2D como pieza fundamental para el análisis de datos – por ejemplo, sin ir más allá, podemos rápidamente conocer la tendencia de cierta variable a partir de una gráfica construida a partir de una tabla de datos sobre la cual, analizándola por sí sola, tomaría más tiempo extraer la misma conclusión –, actualmente la mayor parte de los *software* de simulación convencionales como por ejemplo ExtendSim, ofrecen la herramienta de visualización o animación 3D. Con ello, presentan un entorno en 3D simplificado que refleja en la pantalla el modelo 2D desarrollado previamente en el propio programa. Se ha de aclarar que esta herramienta no presenta ningún interés para el proyecto actual, pues lo que se contempla aquí es una simulación en 3D, sin necesidad de trabajar previamente en dos dimensiones.

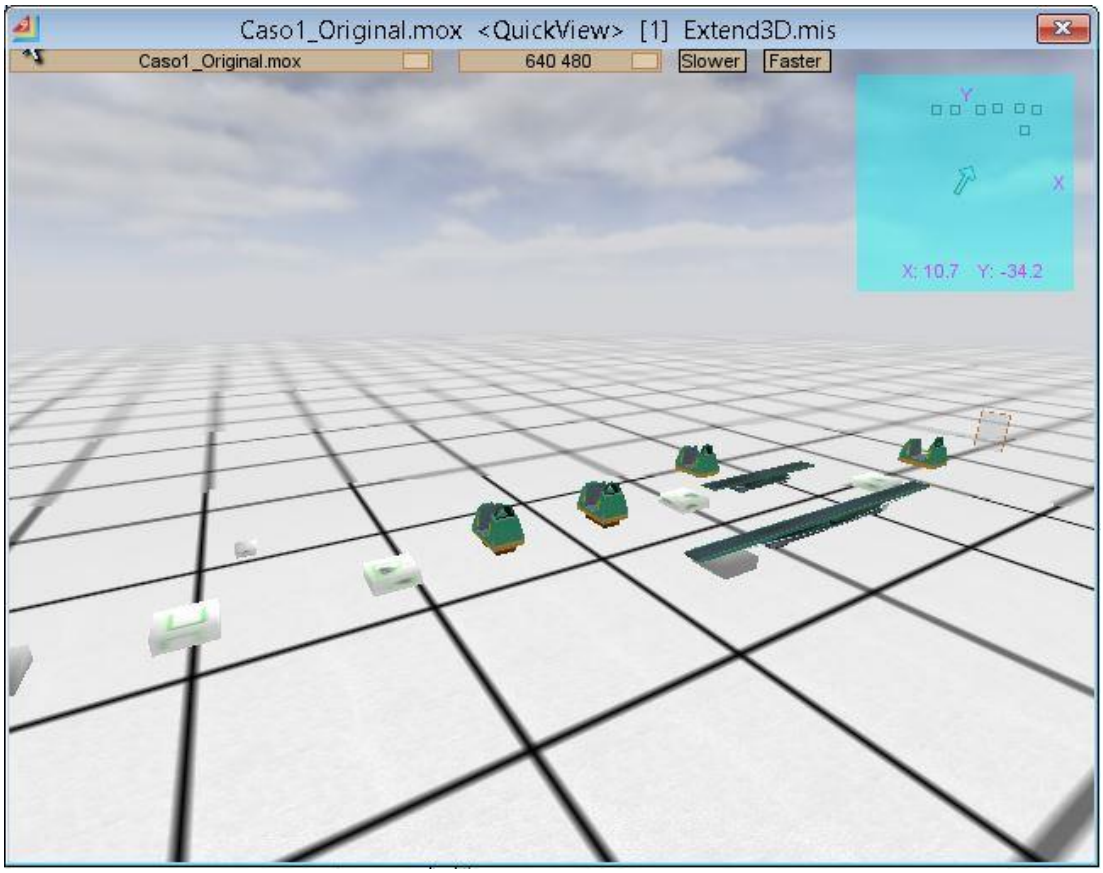

Figura 5. Visualización 3D de un modelo de simulación en ExtendSim. Fuente: elaboración propia.

Capítulo 3

Tras la breve introducción en el segundo capítulo al concepto de visualización, se decide concebir el capítulo actual como un paso más allá del anterior, donde se examinaban las bases de la simulación de eventos discretos. Con este pretexto, es posible entrar de lleno en el concepto de simulación 3D, que constituye realmente la idea fundamental de la que parte este trabajo. Para abarcar todos los aspectos de interés, se comenzará por destacar las potencialidades que aporta aplicar esta técnica a cualquier proceso productivo. No obstante, al tratarse de una herramienta que está al orden del día, deberemos hacer un pequeño análisis de algunos de los programas desarrollados con este fin – *software* comercial de simulación o *software* de eventos discretos –, para conocer con claridad que ventajas aportan estos *software*. Finalmente, introduciremos el papel de los motores de videojuegos en la actualidad, especificando algunas de sus características principales.

3.1. Simulación 3D

Antes de todo, como ya ha sido mencionado, se propone aquí un tipo de simulación donde el ingeniero, una vez desarrollado el modelo conceptual del proceso productivo, y entendiendo todas las particularidades del mismo, construye virtualmente la propia planta de la fábrica – considérese un caso genérico de esta manera – en el ordenador, ajustando la posición de las máquinas, el comportamiento de las mismas, la llegada de material, la salida de producto…, etc. De alguna manera, se asemeja a la simulación en 2D con la diferencia fundamental del entorno en el cual nos estamos moviendo.

Lo que se obtiene por tanto es un mundo virtual, donde el usuario o cliente puede ver construido su proceso productivo o bien, el diseño que tenía en mente hecho “realidad”. De la misma manera que un videojuego, podrá moverse en primera persona por la planta de la fábrica, observar como el producto va avanzando por el proceso a medida que avanza la simulación y, lo que realmente es interesante, obtener cualquier dato de tiempos, cantidades, fallos, cuellos de botella…, etc., del mundo que observa. No obstante, el lector puede comprobar que muchas de estas utilidades o bien han sido descritas anteriormente, o bien son posibles de obtener con una simulación “típica”. La pregunta por tanto es: ¿Qué valor aporta realmente una simulación en 3 dimensiones de cara a la consecución de los objetivos de cualquier simulación? ¿Tiene sentido por tanto desarrollar esta herramienta? Iremos respondiendo a esta pregunta en los siguientes puntos.

3.1.1. Ventajas e inconvenientes

De esta manera, es primordial estudiar que avances y mejoras nos proporciona desarrollar un modelo en 3D de manera genérica, para luego concretarlo al caso que nos concierne.

No obstante, primeramente, es necesario echar la vista atrás, hasta los comienzos del concepto de visualización 3D, la cual ha sido debatida en numerosas ocasiones en el pasado. En el propio libro de Banks encontramos citas como las de R.J. Paul, quien discute los posibles riesgos en su artículo “Recent Developments in Simulation Modeling” (Wenzel & Jessen, 2001):

“En la actualidad, hay un problema con el uso común de sistemas gráficos y de animación en color. Como el usuario ve lo que el modelo de simulación hace, podría creer que entiende el sistema que el modelo está tratando de emular. La visualización de un modelo de simulación en ejecución constituye una regla peligrosa para determinar lo que sucede en el sistema en cualquier momento”.

Otros autores escriben sobre inconvenientes derivados de “el análisis de los 15 segundos”, el “sobre–modelado” o el coste del desarrollo (Wenzel & Jessen, 2001). Afirman que la animación es una herramienta suplementaria que nunca debería reemplazar a las técnicas de análisis estándares. En suma, coinciden con R.J.Paul en que “uno no puede concluir que un sistema está bien definido observado la animación durante un período corto de tiempo”.

Es importante analizar algunos de los inconvenientes que han sido extraídos del análisis de estas técnicas. Esto nos permitirá, si bien no rebatir su veracidad, tener en cuenta algunos de los errores que debemos evitar a la hora de emplear esta herramienta.

Comenzando por el problema del "análisis de los 15 segundos", es evidente que en ningún tipo de simulación de cualquier proceso productivo basta una pequeña visualización para comprender la complejidad del proceso. Sin embargo, pretendiendo dar respuesta a este problema, se propone considerar dos ideas fundamentales.

Por un lado, partimos de la idea base de que el aspecto más importante de cualquier simulación son los datos "virtuales" obtenidos que "simulan" a los posibles reales. Y como la simulación en 3 dimensiones tiene como objetivo potenciar las técnicas de simulación y optimización existentes, es obvio que para ello ha de recoger todas las posibilidades de la simulación convencional posibilitando que el usuario obtenga cualquier tipo de dato que la simulación pueda proveer. De manera más tangible, si nos imaginamos en primera persona dentro de ese entorno de 3 dimensiones, cada máquina que constituya el proceso podrá informarnos de la media de productos que es capaz de realizar cada cierto periodo de tiempo, de la media de revisiones que necesita, de si existe algún tipo de cuello de botella o momento en el que se encuentre parada sin razón aparente…, etc. Lo que se quiere hacer ver es que, conociendo las reglas de uso del modelo desarrollado, es posible obtener cualquier información del proceso en cualquier momento. Esta es una de las mayores virtudes de esta herramienta: La **simulación del proceso en tiempo real**.

Por otro lado, respecto a la creencia de que uno entiende el proceso simulado virtualmente por ver lo que muestra la pantalla, se trata de una visión errónea de lo que realmente aporta esta herramienta. Podemos enumerar una serie de áreas en las cuales las tres dimensiones potencian enormemente la simulación (Rohrer, 2000):

- Desarrollo del *layout* del modelo. En aquellos casos en los que el objetivo sea implementar un proceso productivo, una de las partes fundamentales en el desarrollo del modelo es la disposición en planta de todos los elementos que lo conforman. En este sentido, la simulación 3D nos proporciona una perspectiva realmente ventajosa frente a cualquier modelo en dos dimensiones. Además, haciendo uso de las herramientas que tenemos a nuestra disposición, podremos conseguir niveles de detalle muy altos al ser capaces de obtener un modelo 3D de la superficie donde se ubica nuestro proceso. A partir de ahí, podremos situar con precisión todas las partes que conforman el sistema y estudiar las diferentes alternativas que se pueden conformar el *layout* óptimo.
- Etapas de verificación y validación. El modelo conceptual y el proceso real pueden ser verificados observando el modelo. En los casos en que se simulan procesos reales, existentes, donde tenemos datos sobre ellos, la validación del modelo puede llevarse a cabo correctamente. Sin embargo, cuando el sistema productivo no existe, el proceso de validación se vuelve más complicado. Es aquí donde la simulación 3D juega un papel importante. Debemos tener en cuenta que a la hora de desarrollar y validar un modelo, pueden entrar en juego diferentes especialistas que se encarguen de una parte del trabajo, esto es, un equipo interdisciplinario. Cada uno tendrá dominio y experiencia en una cierta área de conocimiento, pero comprender el proceso en su totalidad podrá ser necesario por cada uno de ellos de cara a realizar un buen trabajo. De esta manera, la visualización se constituye como un instrumento de comunicación para que los diferentes expertos alcancen un nivel compresión del sistema adecuado y común antes de validar el modelo. Se trata de un ahorro de tiempo. La animación se convierte en una manera de acelerar el entendimiento del funcionamiento de cualquier parte del sistema por cualquier persona que necesite o pretenda estudiarlo.
- Compresión de los resultados y obtención de conclusiones. En muchos casos, el analista no es capaz de interpretar los resultados que resultan de simular el modelo o, por otro lado, el conocimiento que se puede extraer de la parte estadística es limitado. Gracias a la simulación 3D, el ingeniero puede observar el área de interés y comprobar

lo que sucede. Con ello, es muy posible que entienda cómo el comportamiento dinámico del sistema afecta a los resultados obtenidos. Debemos considerar que los procesos de producción son inherentemente complejos. En consecuencia, los resultados de las simulaciones pueden ser también muy complejos o no intuitivos. Junto con ello, una de las funciones del ingeniero analista es la de ser capaz de explicarle al cliente esos mismos resultados. La simulación 3D se convierte en estos casos en una herramienta realmente potente al uso del ingeniero para poder explicar el comportamiento del sistema.

- Comunicación de los resultados. Existe también un "público no técnico" a la hora de comunicar los resultados de nuestra simulación. De hecho, generalmente la dirección de proyecto o de la empresa no posee los conocimientos técnicos para comprender los resultados estadísticos. Por ello, cuando es posible utilizar la animación como técnica de comunicación, se es capaz de alcanzar un mejor nivel de compresión. El director puede comprobar por sí mismo la existencia de un cuello de botella o de una *buffer* colapsado. Por tanto, junto con lo anterior, la simulación 3D permite eliminar largas horas de discusión así como la necesidad de presentar, explicar, justificar y cuestionar datos estadísticos.
- Credibilidad de la simulación. Cabe decir que pueden darse casos donde ciertas personas involucradas en el proyecto pueden ser escépticas respecto a las posibilidades de la simulación. Debemos tener en cuenta que, a nivel del proyecto, es fundamental que cualquier trabajador esté convencido plenamente del trabajo a realizar. La posibilidad de observar y comprobar la reacción del sistema al cambiar cualquier parámetro de entrada, convencerá a aquellos posibles escépticos de las capacidades de esta herramienta. Una vez que se ha alcanzado cierto nivel de aceptación del modelo, clave a la hora de realizar cualquier simulación, podrán compararse los resultados de distintos escenarios de cara a la toma de decisiones sobre cualquier futuro o existente proceso productivo.

Finalmente, cabe comentar que, de la misma manera que citábamos alguna de las reglas que establecían Banks y Gibson sobre la necesidad del desarrollo de un modelo de simulación, podríamos reflexionar acerca de en qué casos es adecuado o no realizar una simulación 3D de un proceso productivo. Si bien no es el objeto de este estudio, con el *software* de simulación 3D existentes, empieza a cuestionarse la necesidad de realizar previamente o, preferiblemente, una simulación 2D. Las interfaces de estos programas están cada vez más adaptadas al usuario analista que pretende realizar la simulación. Junto con ello, como hemos visto, una simulación en un entorno de 3 dimensiones siempre aportará un potencial mucho mayor a cualquier caso de simulación. Podremos empezar a hablar de los conceptos de Realidad Virtual y Realidad Aumentada, donde el puente que une ambos con la modelización en 3D es bastante pequeño.

Cabría considerar aquellos casos medianamente simples donde no existe un presupuesto realmente suficiente, un equipo capacitado, o donde no hay necesidad de visualizar el modelo. Puede ser que nuestro único interés sea obtener un informe final de tiempos de proceso y productos procesados sin que la simulación en tiempo real o la visualización del *layout* sea pertinente. Claro está, sí que tendría entonces sentido volver al mundo de las dos dimensiones.

3.2. Análisis del *software* de simulación comercial

Por tanto, será muy importante tener una visión general y en pocas palabras de algunos de los más destacados *software* de simulación existentes en el mercado. Para ello, el estudio que se muestra a continuación es recogido del artículo (Bijl & Boer, 2011). La lista se elabora teniendo en cuenta los programas más especializados en la visualización 3D. Además, el análisis se lleva a cabo considerando dos aspectos: 1) la manera en la que se emplea la visualización y 2) la claridad y el realismo de la simulación.

3.2.1. Anylogic

Se trata de un *software* basado en Java en el cual las simulaciones se crean empleando bloques de estado, gráficos o diagramas de acción y eventos. Para cada componente de la simulación puede seleccionarse un objeto en 3D que lo representará. Para ello, se puede emplear la librería de objetos en 3D que trae el propio programa o bien, importar el ente de otra fuente. Con Anylogic, los diagramas que definen la lógica de la simulación se encuentran separados del *layout* de la visualización. De esta manera, los diagramas pueden estructurarse de manera ordenada y clara mientras que la animación sigue siendo fiel a la realidad, ya sea más o menos ordenada. Además, el usuario puede diseñar nuevas disposiciones en planta del proceso, añadir gráficos de estadísticas y configurar numerosos parámetros para obtener una visión general completa del progreso y los resultados de la simulación.

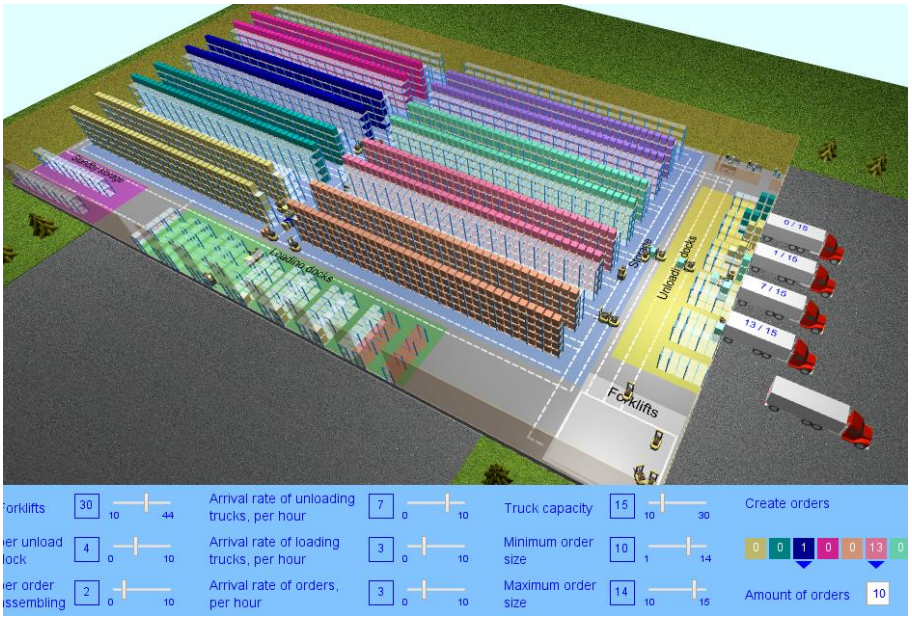

Figura 6. Interfaz gráfica de Anylogic. Fuente: http://www.benjamin-schumann.com.

3.2.2. Enterprise Dynamics

Dentro de Enterprise Dynamics (ED), a pesar de que la animación 3D juega un rol principal, el modelo conceptual es plasmado en el *software* utilizado para ello diagramas en 2D. Posteriormente, se constituye un equivalente 3D para todos los elementos de la simulación, creando así el entorno 3D. Su apariencia se define usando simples scripts.

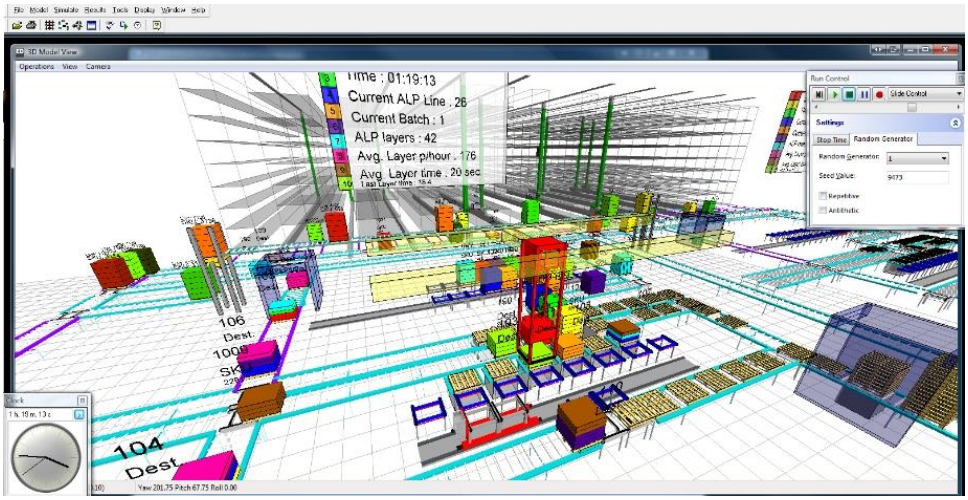

Figura 7. Interfaz gráfica de Enterprise Dynamics. Fuente: http://www.incontrolsim.com.

3.2.3. FlexSim

FlexSim posee un entorno 3D donde se puede construir directamente el modelo de simulación y ejecutarlo una vez validado. Los elementos de la simulación están representados por modelos 3D, los cuales pueden situarse en el entorno 3D. Posteriormente, estos componentes pueden contactarse a otros, y su apariencia puede modificarse en cualquier momento. El modelo 3D que representa cada elemento o entidad puede seleccionarse en una librería que provee FlexSim, además de poder importar modelos 3D externos. Una vez que se ejecuta la simulación, se puede observar en tiempo real la acción de los diferentes elementos que componen la simulación así como el progreso de la misma. Durante la propia ejecución, es posible modificar las propiedades de los objetos.

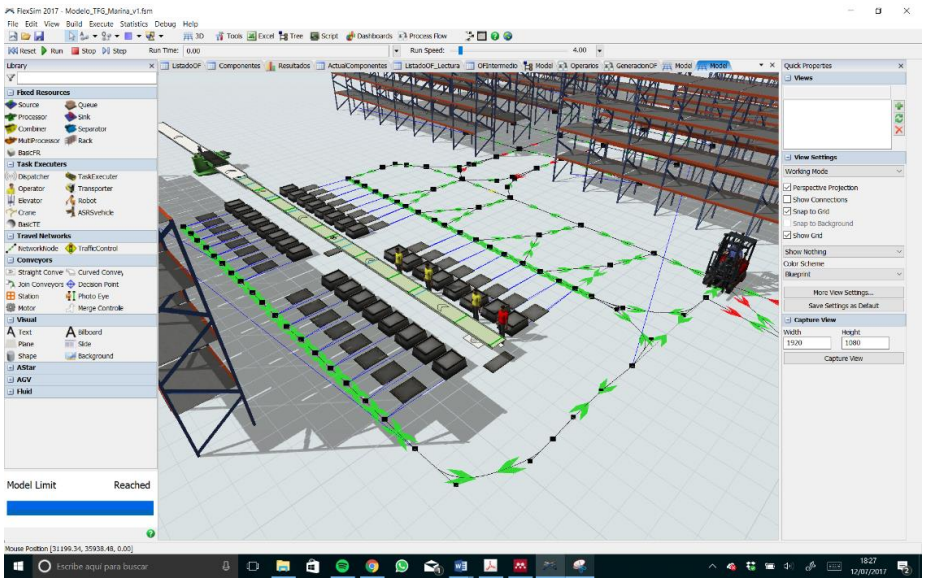

Figura 8. Interfaz gráfica de FlexSim. Fuente: elaboración propia.

3.2.4. Simio

El modelo conceptual es trasladado a Simio a través de diagramas en un entorno de 2 dimensiones. En él se van colocando los diferentes elementos que componen la simulación, los parámetros se ajustan en un cuadro de diálogo y, de la misma manera que AnyLogic, se asigna un modelo 3D para cada nodo del diagrama. No obstante, Simio posee una funcionalidad extra: Conexión directa con Google 3D Warehouse, un enorme almacén de modelos 3D, lo que hace mucho más fácil añadir modelos 3D personalizados.

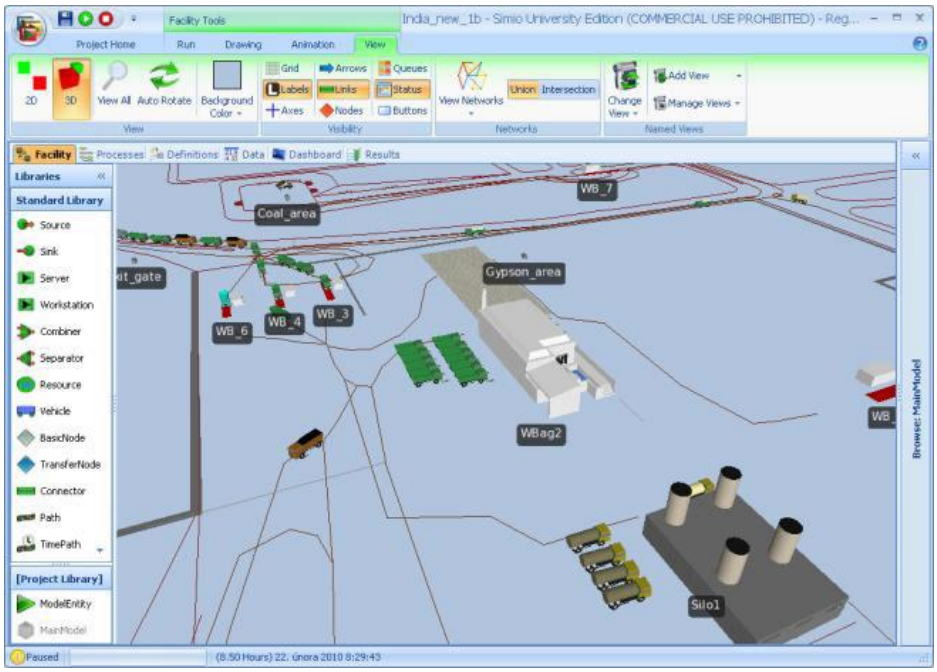

Figura 9. Interfaz gráfica de Simio. Fuente: www.simio.com.

3.2.5. Comparación

El mismo artículo realiza finalmente una comparación teniendo en cuenta 3 factores o capacidades principales de los *softwares*. Estos tres factores que se describen a continuación resultan fundamentales para analizar posteriormente las ventajas que nos puede aportar un *software* de videojuegos. En este sentido, tenemos:

- Paralelismo temporal. Hacer referencia a si la visualización se ejecuta al mismo tiempo que la simulación o bien se efectúa posteriormente. Es lo que denominamos anteriormente como simulación en tiempo real.
- Interactividad. Hace referencia a si, en el propio entorno 3D, es posible modificar el modelo, seleccionar objetos, modificar sus propiedades…, etc.
- Autonomía de la herramienta de visualización. Se refiere al grado de dependencia e integración de la herramienta de visualización en la simulación.

Con todo ello, es posible construir la *Tabla 1* donde se muestra cuáles de los *softwares* comentados posee o no las capacidades mencionadas.

Tabla 1. Comparación de *software* comercial de simulación

	AnyLogic	ED	FlexSim	Simio
Paralelismo	✓	✓	✓	✓
Interactividad	–	✓	✓	✓
Autonomía	–	–	✓	–

Fuente: (Bijl & Boer, 2011).

Asimismo, en lo que se refiere a elemento gráfico de las simulaciones, a la apariencia, prácticamente todos los motores son bastante parecidos, Sin embargo, algunos como FlexSim, Simio o Enterprise Dynamics permite añadir texto 3D a la visualización, de cara a mostrar algún dato importante o *output* del proceso.

3.3. Motores de videojuegos

Se han caracterizado hasta el momento algunos de los programas principales que podemos encontrar en el mercado. A estas alturas, debemos tener en cuenta que la visualización 3D no solo se aplica en áreas como la industria, sino que juega un papel importante en otros muchos sectores. Algunos como la medicina llevan años explotando esta herramienta, la cual permite realizar grandes progresos en campos como la prevención de accidentes, cirugía, investigación…, etc. Pero, con diferencia, uno de los ámbitos que lleva la voz cantante es el sector de los videojuegos. Estamos hablando de una industria multimillonaria[2] que se aprovecha del potencial de los ordenadores, consolas u otros dispositivos tecnológicos actuales de venta masiva y que, debido a su complejidad, suele requerir un conocimiento especializado en numerosas áreas para el desarrollo de cualquier videojuego. En consecuencia, los juegos suelen realizarse de un modo modular siendo el motor de videojuegos el corazón central de la aplicación. Este último consiste en un conjunto de módulos que se ocupan simultáneamente de los gráficos 3D, la inteligencia artificial, la gestión de la memoria, las conexiones con otras aplicaciones e Internet, el sonido y todas aquellas tareas que forman parte de un videojuego. Lo esencial del motor es que está diseñado de tal manera que pueda ser "reutilizado" para otros juegos. De hecho, una de las funcionalidades principales es que podemos aprovechar en futuros proyectos todo o parte de los elementos desarrollados en un proyecto.

En este sentido, como se ha mencionado anteriormente, este proyecto de investigación propone aprovechar el potencial de un sector que mueve enormes cantidades de dinero para aplicarlo a la simulación 3D de procesos industriales. Este concepto no resulta tan novedoso como pueda parecer. De hecho, el artículo (Friese, Herrlich, & Wolter, n.d.) ya nos habla del término "Serious Gaming", el cual procede originalmente de la aplicación de los videojuegos en áreas como la educación y el aprendizaje, pero que ha sido extrapolado a toda aplicación que emplee las posibilidades de estos motores en otros propósitos diferentes del entretenimiento. El mismo artículo presenta una serie de ejemplos entre los cuales encontramos la construcción de un museo virtual, la visualización de información geográfica en el contexto de orientación e investigación, o la creación de un entorno virtual que simula una cueva.

Otros como (Herwig & Paar, 2002), (Herrlich, 2007) o (Indraprastha & Shinozaki, 2009) aplican esta tecnología especialmente a la visualización de entornos exteriores, debido a la alta carga de polígonos y la exigencia de rendimiento que ello conlleva. Analizan algunas de las características que mencionaremos posteriormente, como la alta optimización en la arquitectura de estos *softwares*, la libertad de crear entornos interactivos, o la capacidad de aplicar modificaciones y alteraciones en el entorno en tiempo real, frente a los programas GIS y CAD cuyo uso está más limitado, exigiendo por tanto mucho más trabajo.

Por tanto, a pesar de las diferencias inherentes entre el desarrollo de los *serious games* y los juegos de entretenimiento (Cowan & Kapralos, 2014), existen ciertos aspectos importantes donde los motores nos proporcionan determinadas ventajas indudables.

[2] El mercado del videojuego sigue crecido a ritmo vertiginoso, con unas ventas de 91'5 mil millones de dólares en 2015 y un incremento del 11'84% sobre 2014. Se espera que esta cifra de ventas ascienda a 107 mil millones de dólares en 2017. Fuente: (Mishra & Shrawankar, 2016).

3.3.1. Caracterización

Uno de los puntos fundamentales para llevar a cabo este proyecto, es conocer las funcionalidades principales de la mayoría de los motores videojuegos. Algunos de los rasgos que a continuación se definen ya han sido mencionados previamente, si bien ahora se realiza un estudio más profundo recogido en (Cowan & Kapralos, 2014):

- Scripting. Una de las funcionalidades básicas de los motores es el conocido como scripting. Todos los objetos y eventos son controlados mediante simples fragmentos de código que recogen su comportamiento.
- Representación. Como *software* de visualización 2D y 3D, una parte fundamental es la representación de la escena[3]. Esto incluye la velocidad y la precisión con que esta última es generada así como la totalidad de los efectos visuales.
- Animación. Si bien se trata de un término empleado en muchas ocasiones como sinónimo de visualización o representación 3D, en este caso hace referencia al movimiento y deformación de todos los objetos de la escena.
- Inteligencia artificial. A modo de ejemplo representativo de esta funcionalidad, piénsese en el control de comportamientos como perseguir, esquivar o huir con la combinación de la búsqueda de caminos.
- Física. Engloba la interacción física entre los distintos elementos que componen escena. Los objetos responden con precisión debido a colisiones o en respuesta a fuerzas o presiones aplicadas sobre ellos.
- Sonido. La representación espacial del audio le permite a los sonidos tener una localización en el entorno desarrollado. Asimismo, es posible añadir variaciones o señales ambientales como la reverberación.
- Networking. Permite al usuario interactuar con otros usuarios dentro de la aplicación al compartir datos a través de una red.

Además de estas funcionalidades básicas, le *software* de desarrollo de videojuegos posee también una interfaz de usuario gráfica (GUI, por sus siglas en inglés) que generalmente cuenta con varios editores. Algunas de las herramientas de ediciones más comunes son las siguientes:

- Editor de niveles. También conocido como editor de mapas o escenarios, se encarga de la creación de los diferentes entornos virtuales o escenas. Incluiría los niveles del juego, mapas…, etc.
- Editor del proyecto. Gestiona todos los elementos que constituyen la escena así como las relaciones de parentesco (jerarquías) entre los mismos.
- Editor de scripts. Permite gestionar los scripts desarrollados y asociarlos al elemento correspondiente.
- Editor de materiales. Se encarga de los efectos visuales y todos los recursos o *assets* relacionados con las imágenes, materiales, texturas…, etc., de los elementos de la escena. En muchas ocasiones, el editor del proyecto, de scripts y de materiales suelen estar asociados en un único editor (véase *Figura 10*).
- Editor de sonido. El volumen, la atenuación y otras opciones pueden ser combinadas con efectos de filtro provistos por el motor de sonido.

[3] Cuando arrancamos el motor de videojuegos, este nos permite crear un proyecto donde definiremos cada uno de los eventos y objetos que constituyen dicho proyecto. La disposición en escena, la definición particular de sus características y la interacción entre dichos elementos, definirán una escena determinada. Un proyecto puede contener tantas escenas como se desee.

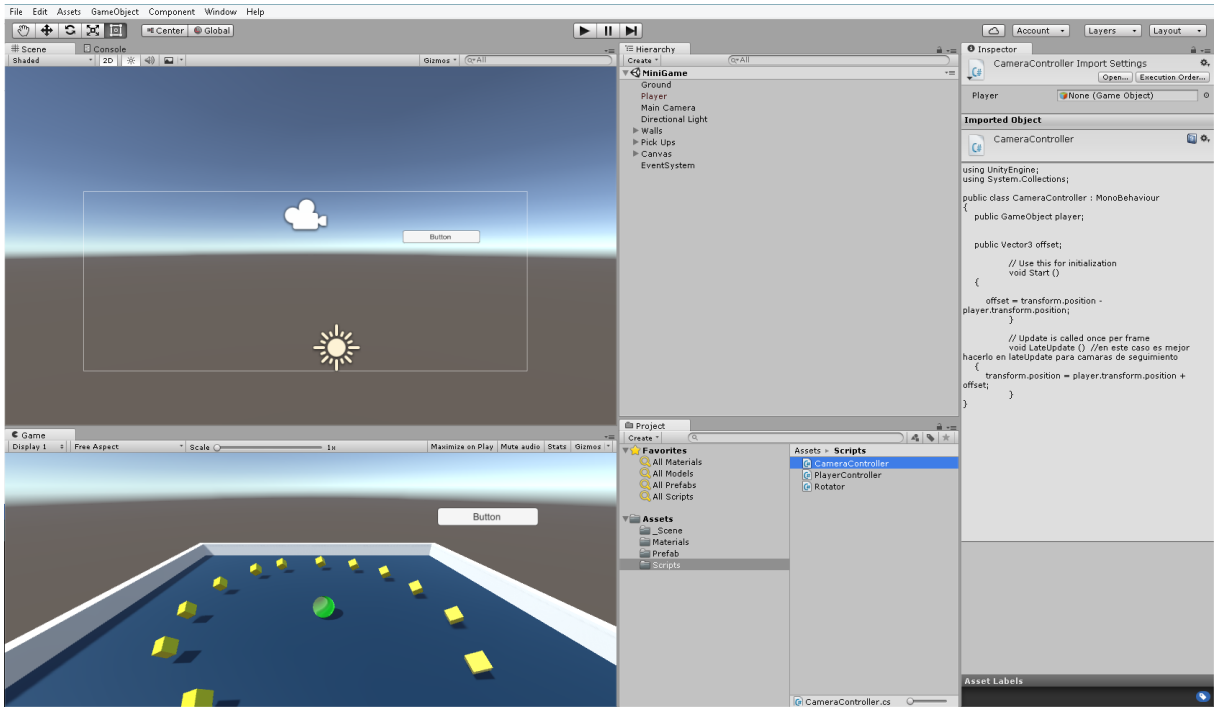

Figura 10. Interfaz de Unity3D. El editor del proyecto, el de scripts y el de materiales están contenidos en un editor general que controla todos los elementos. A la derecha, se muestra un ejemplo de script. Fuente: elaboración propia.

Finalmente, cabe comentar algunas ventajas que nos proporcionan los motores de simulación, especialmente en lo referente parte gráfica y el renderizado. Algunas de estas propiedades que a continuación se exponen, se encuentran en el estudio realizado en (Bijl & Boer, 2011), empero aquí se les proporciona un enfoque orientado a su utilidad para la simulación de procesos. En sentido, encontramos:

- Manejabilidad. Los motores de videojuegos más comunes como CryEngine, Unreal Engine o Unity3D (estos dos últimos de uso gratuito) poseen una interfaz intuitiva y amigable, que permite al usuario principiante manejar fácilmente con el motor. Como ejemplo, en muchos casos, con las librerías oportunas, el modelador puede evitar la programación directa: cuenta con una serie de editores flexibles donde el modelo 3D, la configuración de sus propiedades (posición, forma, visualización, color), y el o los scripts asociados al mismo que describen su comportamiento, pueden ser configurados de una manera ágil y sencilla. En suma, abierto el motor, podemos configurar el *layout* a nuestro gusto separando las diferentes ventanas que nos muestran las distintas partes de nuestro proyecto, incluida la visualización 3D.
- Alto rendimiento. Una de las ventajas más notables de este tipo de *software* es el rendimiento de procesamiento que proporciona. Desarrollar un entorno 3D con numerosos elementos puede ser muy costoso en términos de potencia de procesamiento. En este sentido, los desarrolladores de estos motores han identificado los cuellos de botella que se producen en la ejecución de los videojuegos, el uso de memoria o la transmisión de datos. Técnicas como la reducción de polígonos en función de la distancia a la cámara o el *batching,* que aligera el peso de los objetos estáticos, permiten optimizar enormemente tanto el desarrollo como la generación de la aplicación.
- Importación de modelos, escenarios o texturas. Si bien en el caso de los *softwares* comerciales se comentaba la posibilidad de importar modelos 3D de objetos, en el caso de los motores de videojuegos se da un paso más allá. Con aplicaciones como de modelado 3D es posible construir y obtener el modelo 3D equivalente al escenario real donde situamos el proceso. Podemos conseguir así un entorno mucho más realista y adaptado a nuestra simulación. En este sentido, se está investigando con

tecnologías como el escáner laser 3D terrestre (Lindskog, Vallhagen, & Johansson, 2017) para construir una representación virtual de un proceso real de producción. Todo ello se suma a las posibilidades gráficas que poseen (importación de texturas, modelos, objetos…, etc.), pero que son más alcanzables por los *softwares* comerciales de simulación.

- Calidad gráfica. Uno de los saltos más destacados que permiten los motores frente a los *softwares* de eventos discretos, es el nivel de calidad gráfica que proporcionan. Como se describe en (Bijl & Boer, 2011), se ha realizado un gran esfuerzo por parte del sector de los videojuegos en conseguir unos gráficos convincentes, realistas y, desde luego, el resultado ha sido extraordinario. El control de las sombras, texturas, iluminación, reflejos, efectos de tiempo, movimiento…, etc., constituyen la herramienta gráfica más potente del mercado. Si bien la mayoría de estas características no aportan valor a la validación y análisis del modelo, si tienen especial importancia en el aspecto del marketing, como bien apunta el mismo artículo. Es importante tener en cuenta todo lo mencionado sobre la comunicación de los datos a ese "personal no técnico" que posiblemente constituya la dirección del proyecto. Evidentemente, si se pretende vender las posibilidades de la simulación, el marketing será una parte fundamental.
- Usabilidad. Por último, el concepto de usabilidad hace referencia principalmente al sistema automático de control de la cámara, donde cada movimiento se decide en función de las acciones del usuario y el entorno 3D en el que se mueve. Por ejemplo, cuando existe algún elemento que bloquea la vista, el propio sistema ejecuta alguna opción para permitir la vista (ya sea mover la cámara o convertir dicho objeto en transparente). No obstante, también hace alusión al desarrollo del proceso donde, como se indicó, el modelador no tiene la necesidad de programar a un nivel bajo.

3.4. Realidad virtual

Este apartado pretende dar una idea básica del concepto de realidad virtual del cual existen multitud de definiciones. La razón de ello es su inclusión final en el presente trabajo, debido a las capacidades que nos proporcionan los motores de videojuegos. Además, interesa esclarecer el salto que se da de los mundos virtuales de tres dimensiones a los entornos VR (realidad virtual).

Como bien indica (Martínez, 2011), "la realidad virtual comprende la interfaz hombre-máquina que permite al usuario sumergirse en una simulación gráfica 3D generada por ordenador, y navegar e interactuar en ella en tiempo real desde una perspectiva centrada en el usuario" (p.5). Además, describe el denominado "triángulo de la realidad virtual", compuesto por tres características que permiten diferenciar este concepto de las animaciones 3D. Pese a que al igual que sucede con la propia definición, no hay un consenso total sobre este punto, podemos considerar estos tres rasgos como los siguientes:

- Interacción: posibilita la interacción del usuario con los elementos que conforman el mundo virtual a través de diversos dispositivos de entrada como joysticks, guantes de datos, gafas de VR…, etc.
- Tiempo real: permite que tal interacción con el mundo virtual sea instantánea. El movimiento de la cámara, la dirección hacia donde moverse en el escenario, o la acción a ejecutar se realiza en el momento en que el usuario decide llevarla a cabo.
- Inmersión: el objetivo de la VR es que el usuario pierda el contacto con la realidad exterior, percibiendo únicamente los estímulos del mundo virtual, y consiguiendo una experiencia lo más inmersiva posible.

Según esta definición, la distinción entre realidad virtual y la animación tradicional queda totalmente nítida. De hecho, ya fue indicada indirectamente en *2.4.4 Visualización de la simulación*. No obstante, (Martínez, 2011) abarca en la misma el mundo de los videojuegos, lo que no atiende al punto de vista de este trabajo.

De esta manera, se remarca aquí la inmersión parcial y la sensación de profundidad que deben proporcionar los dispositivos de realidad virtual, y que únicamente son ofrecidos por las gafas, y no por los ordenadores o *smartphones*. Este es el concepto de realidad virtual sobre el que se trabaja en el presente trabajo, dando el salto desde lo que definimos como el mundo en tres dimensiones, el paso previo.

Capítulo 4

Una vez contextualizado el estudio, se procederá a describir las librerías que permiten implementar un modelo de simulación genérico. Durante este proceso, se desarrollan varios proyectos en Unity, previos al modelo final, los cuales sirven para depurar y perfeccionar el código, así como para probar y completar los diferentes recursos gráficos desarrollados.

Al tiempo que se completan las librerías, se escoge el caso concreto de demostración, del cual, como se indicó en *1.3 Recursos del proyecto*, se ha de disponer de un modelo de simulación desarrollado en FlexSim. Cabe comentar que no se realiza en este trabajo un estudio del contexto que engloba al proceso, incluida la propia empresa. Tampoco es necesario que el modelo de FlexSim esté validado o sea totalmente fiel a la realidad. Como se mencionó previamente, todo esto es debido a que el caso elegido tiene interés únicamente desde el punto de vista de la comparativa de *softwares*. El requisito de la existencia del modelo de FlexSim radica exclusivamente en la obtención de unos tiempos de proceso y en dicha comparativa.

El desarrollo de los recursos gráficos se realiza una vez se elige el caso de demostración. Se destaca aquí la importancia clave de esta fase, pues marca la diferencia para con el usuario final, tal y como se señala en el *Capítulo 3*. En ella, se procede a generar el entorno virtual en el que se desarrolla el caso, y que proporciona una sensación de inmersión virtual al usuario, más aún cuando se implementa la realidad virtual. Dentro de este entorno se implantará el modelo, el cual poseerá una lógica parecida al proceso real que simula.

4.1. Modelización conceptual de una simulación genérica

El paso previo al desarrollo de las librerías es definir los elementos que conforman la modelización conceptual de un proceso de fabricación genérico. Ello obedece a una de las premisas del trabajo: generar un código lo más completo posible dentro del alcance del presente proyecto, y que permita modelar un proceso productivo cualquiera e independiente del caso de demostración aquí empleado. Dicho código podrá ser siempre ampliable en futuros trabajos, de cara a modelar procesos más complejos.

Sobre esta base, podemos decir que un hipotético proceso de fabricación simplificado podría presentar un diagrama de flujo similar al de la *Figura 11*:

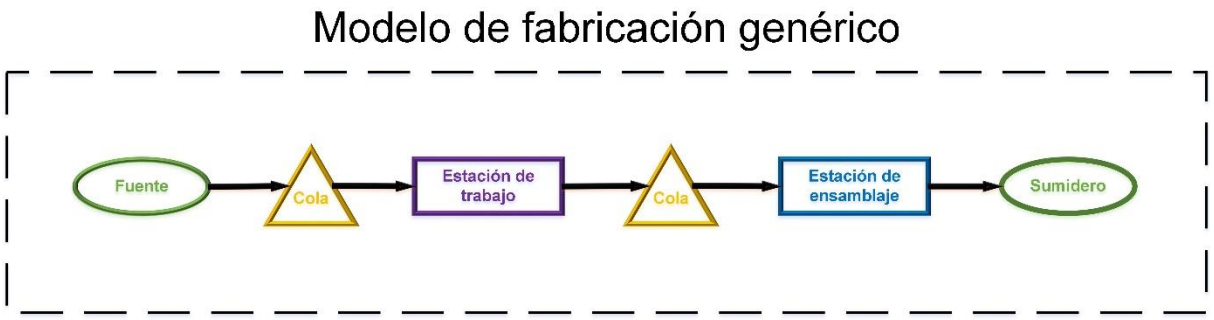

Figura 11. Ejemplo de diagrama de flujo de un hipotético proceso de fabricación. Fuente: elaboración propia.

A partir del diagrama, podemos definir los siguientes elementos y las principales directrices que modelan su comportamiento:

- Fuente: es el elemento encargado de crear y programar la llegada de los ítems al proceso. Esta programación puede realizarse de acuerdo a una distribución estadística que se ajuste al proceso real, o bien en función de una base de datos que recoja los tiempos de llegada de los diferentes ítems.
- Cola: no siendo un elemento existente en cualquier proceso manufacturero (véanse los sistemas *Lean Manufacturing*, donde representan una pérdida de tiempo y, por tanto, un despilfarro), sí suelen estar presente en la mayoría de una manera u otra. Su

función es la de almacenar los ítems si el envío al siguiente elemento se encuentra bloqueado.
- Estaciones de trabajo: representa cualquier subproceso, actividad o tarea que ha de atravesar un ítem y en la que forzosamente siempre ha de permanecer un tiempo variable. Es decir, cualquier modificación sobre un ítem a lo largo de una cadena representa conceptualmente una estación de trabajo. El tiempo de procesamiento, de la misma manera que en la fuente, podrá obedecer a una base de datos o a una distribución estadística.
- Estación de ensamblaje: el concepto es similar al de una estación de trabajo pero con una diferencia fundamental: en su paso, un número determinado de ítems serán ensamblados dando lugar a un nuevo ítem. Es decir, es un proceso que siempre tendrá como salida un único y nuevo ítem.
- Sumidero: es el elemento que pone fin al proceso y que conceptualmente, elimina los ítems del mismo.

No obstante, en la modelización conceptual de un proceso existen una serie de elementos fundamentales, en algunos casos abstractos, que gestionan la lógica del modelo y, por tanto, el comportamiento de los anteriores. Estos, que no vienen representados en el diagrama de flujo, se exponen a continuación:

- Reloj de la simulación: elemento central de la simulación, será el encargado de gestionar el tiempo de la simulación, sincronizándolo con el tiempo físico de Unity. Es el responsable del progreso *de forma discreta* del tiempo, avanzándolo en pequeños intervalos en los cuales se ejecutarán los eventos programados. Estos eventos programados se almacenan en una lista que maneja el reloj, y corresponden a las diferentes órdenes de llegada, procesado y salida enviadas por las estaciones, las fuentes y o los sumideros.
- Conectores: son los encargados de gestionar las diferentes conexiones entre los elementos que componen la cadena. Es decir, proporciona el lenguaje de comunicación entre los mismos, permitiendo que se entiendan entre ellos.
- Ítems: con este nombre se hace referencia a cualquier entidad única y diferenciada que atraviesa el proceso de producción, desde su fuente hasta el sumidero, siendo sometida en su paso a diferentes actividades o transformaciones que suponen un alto temporal en su camino.
- Servidores: con la premisa de general un código genérico, los servidores que procesan los ítems se administran con una entidad propia. Esto permite generar estaciones de trabajo con varios servidores.

4.2. Desarrollo del código

Con todo lo anterior, es posible abordar el desarrollo del código. Se ha de aclarar que se parte de unas librerías proporcionadas por el Grupo Integrado de Ingeniería de la Universidade da Coruña, las cuales se tradujeron de Java a C#, se completaron y, posteriormente, se emplearon en el caso de demostración.

En este sentido, se describirán brevemente las clases que representan a los elementos descritos en el punto anterior y cómo se gestiona el parentesco entre las mismas. Sin embargo, con el propósito de emplear una metodología adecuada de explicación, se definen primer lugar dos niveles de abstracción fundamentales:

- Nivel base o lógico. Este primer nivel hace referencia a la parte del código que define y modela conceptualmente los diferentes elementos de la simulación, su comportamiento, y las relaciones de herencia entre los mismos. Por tanto, es aquí donde se gestiona la lógica de la simulación de eventos discretos. Este nivel está compuesto por cuatro librerías diferenciadas:
    - Librería “SimElements”. La clase base de este nivel es la clase abstracta “Element”, la cual será heredada por los elementos de la librería a excepción

de la clase “Item” y la clase “ServerProcess”. Las clases definidas en esta librería simulan el comportamiento de los elementos mostrados en el diagrama de flujo de la *Figura 11*, incluidos los elementos ítems y servidores.
    - Librería “SimClock”. Contiene la clase “SimClock” que gobernará el comportamiento del reloj de la simulación. Además, contiene la clase “DoubleMinaryHeap” que permite gestionar listas de elementos de forma eficiente y la interfaz “Eventcs”, heredada por “Item” y “ServerProcess”.
    - Librería “SimLink”. Contiene las clases “SimpleLink” y “Multilink”, que se encargan de realizar las conexiones. Ambas heredan la interfaz “Link”.
    - Librería “SimValues”. Perteneciente al código inicial proporcionado por el Grupo Integrado de Ingeniería, contiene una serie de clases que cuya función es generar valores de tiempos aleatorios a partir de una semilla, y correspondientes a una distribución de Poisson. Como se indicará más adelante, existen versiones previas al código final donde las clases de las estaciones de trabajo obtienen los tiempos de procesado haciendo uso de esta librería.
- Nivel virtual (nivel de Unity). El segundo nivel de abstracción constituye la implementación del primer nivel en Unity. Las clases definidas en este nivel son los objetos virtuales que forman parte de la simulación y que, por tanto, componen la interfaz gráfica del modelo. Pero el comportamiento de estos objetos obedece a la lógica desarrollada en el nivel base. Dentro de él encontramos una única librería “SimSElements” que agrupa todas las clases, incluida “SElement” que conforma la clase abstracta de este nivel la cual hereda a su vez “MonoBehaviour”, la clase base de Unity de la que deriva cualquier Script. Únicamente “UnitySimClock” y “UnityMultiLink” no heredan de ella, sino que lo hacen directamente de “MonoBehaviour”. Además, todas las clases de Unity tienen el mismo nombre que la clase correspondiente en el nivel base la cual instancian, pero precedido de Unity[4].

Por tanto, cada objeto virtual de Unity que forme parte de la simulación es, a fin de cuentas, una clase de la librería “SimSElements” (a excepción del reloj y las conexiones), y esta clase instancia a su vez un objeto lógico del nivel base, el cual rige su comportamiento. Es decir, podemos hablar de una doble correspondencia que emplearemos para explicar el código. Para mayor claridad, se expone a continuación un diagrama explicativo simplificado de las correspondencias en el código. No se muestra la librería “SimValues” al no emplearse en el modelo del caso de demostración.

[4] Esto es, si la clase del nivel lógico es tiene el nombre de “ClaseLógica”, en el nivel virtual se llamará “UnityClaseLógica”.

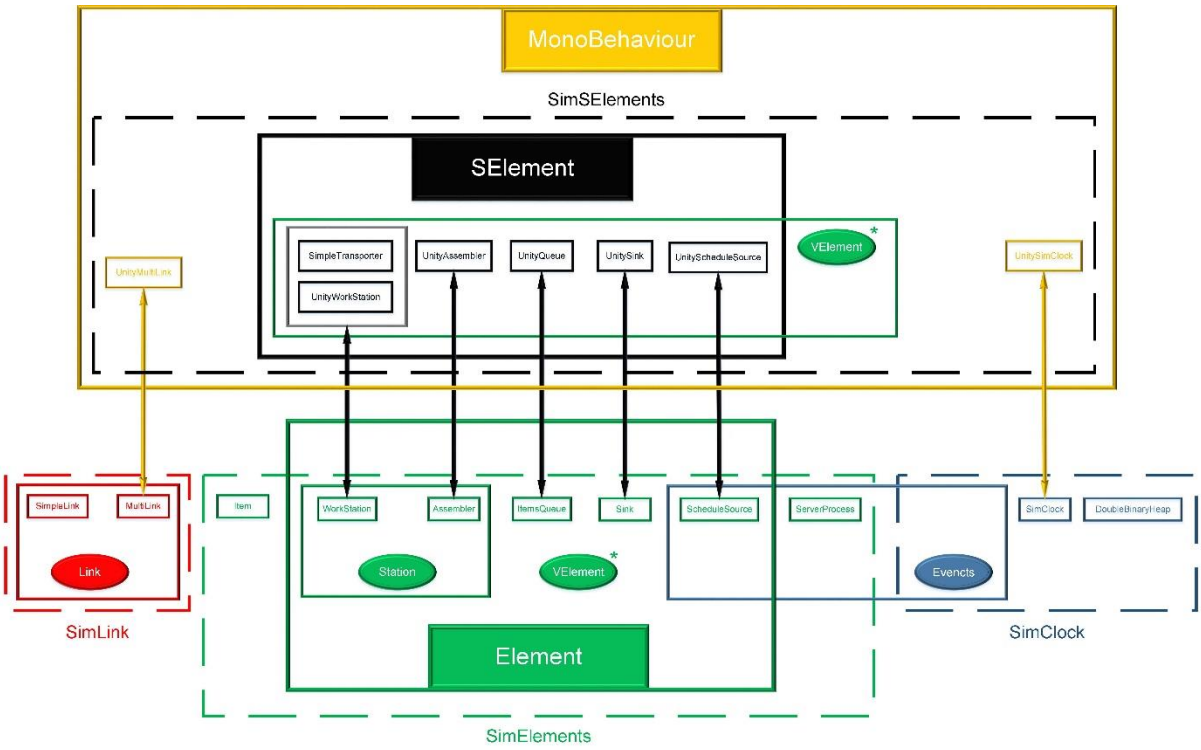

Figura 12. Diagrama explicativo de las librerías empleadas. Las flechas establecen las relaciones entre el nivel lógico y el virtual (la clase de Unity siempre instancia el elemento de la clase lógica). Los recuadros de línea continua identifican las herencias de clases base o interfaces. Los recuadros de línea discontinua se refieren a la agrupación de las librerías. Fuente: elaboración propia.

Por tanto, con este planteamiento, se abordará a continuación cada elemento mencionado en el punto anterior, identificando la clase de cada nivel que le corresponde[5]:

i. Reloj de la simulación:
   a. “SimClock”. Su principal método “advanceClock” sincroniza el tiempo de la simulación de eventos discretos con el tiempo de Unity, ejecutando los eventos que han sido programados para el tiempo discreto de avance. Estos son gestionados gracias a la clase “DoubleBinaryHeap”, que los ordena eficientemente según su tiempo de ejecución representado por el “delay”.
   b. “UnitySimClock”. El elemento virtual del reloj se encarga fundamentalmente de ejecutar los métodos de inicialización y conexión del resto de componentes de la simulación. Para ello almacena todos los elementos derivados de las clases “SElement” y los elementos “MultiLink” en sendas listas, lo cual le permite generar informes de los datos de producción y tiempo de cada estación. Junto a ello, efectúa la orden de avance de tiempo al elemento lógico “SimClock”, una vez el intervalo de tiempo actual ha concluido. Por otra parte, el método “restartSim” permite resetear todos los elementos de la simulación y comenzar de nuevo la misma.

ii. Conexiones:
   a. “SimpleLink”. Es la primera de las clases que sirven para ejecutar y gestionar las conexiones entre los diferentes elementos, siendo su papel crítico. En este caso, permite establecer una conexión únicamente entre dos elementos y tiene desarrollado un único modo de funcionamiento. Esta clase es implementada directamente por las estaciones de trabajo y las colas, definiendo en cada una los elementos previo y posterior. Por tanto, no es necesario realizar una clase correspondiente en el nivel de Unity.

[5] En la lista multinivel, el ítem de la letra “a” corresponde a la clase del nivel base y el “b” a la clase del nivel virtual, a excepción de las “Conexiones”, donde existen dos versiones.

b. “MultiLink”. Es una evolución de la clase “SimpleLink” para realizar conexiones entre múltiples elementos. Inicialmente se establecen esas conexiones en el mundo virtual a través de listas que recogen los elementos previos y posteriores, y cualquier envío de un ítem entre dos componentes de la simulación pasa a través de un elemento “MultiLink”. A la hora de buscar una salida, existen dos modos de funcionamiento, según se trate de una cola (*Queue*) o estación de trabajo (*WorkStation*), o una estación ensambladora (*Assembler)*. Cabe comentar que, en el momento en que se completa un evento de proceso en una estación, está ejecuta el método “notifyAvaliable” informando al “MultiLink” previo de que se haya libre para un envío. Como respuesta, se ejecutará el envío en caso de haber uno previamente bloqueado. Mención
c. “UnityMultilink”. La necesidad de la existencia de un objeto virtual para los *links* deriva exclusivamente de su única función: establecer las conexiones entre los objetos virtuales de Unity. Por tanto, carece de sentido darles una interfaz visible.

iii. Ítems:
a. “Ítem”. Como entidades propias, tienen su propia clase en el nivel lógico. Esto es debido a que, realmente, a nivel conceptual, no se trata más que de un objeto con una serie de propiedades y tiempos que definirán su camino a través del proceso. En caso de no emplear la librería “SimValues”, de él se extraerán los tiempos de cada estación así como otras propiedades pertinentes. Cabe decir que estas son asociadas al objeto en el momento de su creación.
b. No tienen una clase a nivel virtual, pues su instanciación en Unity se hace en la actividad en la que se haya (*Figura 13*).

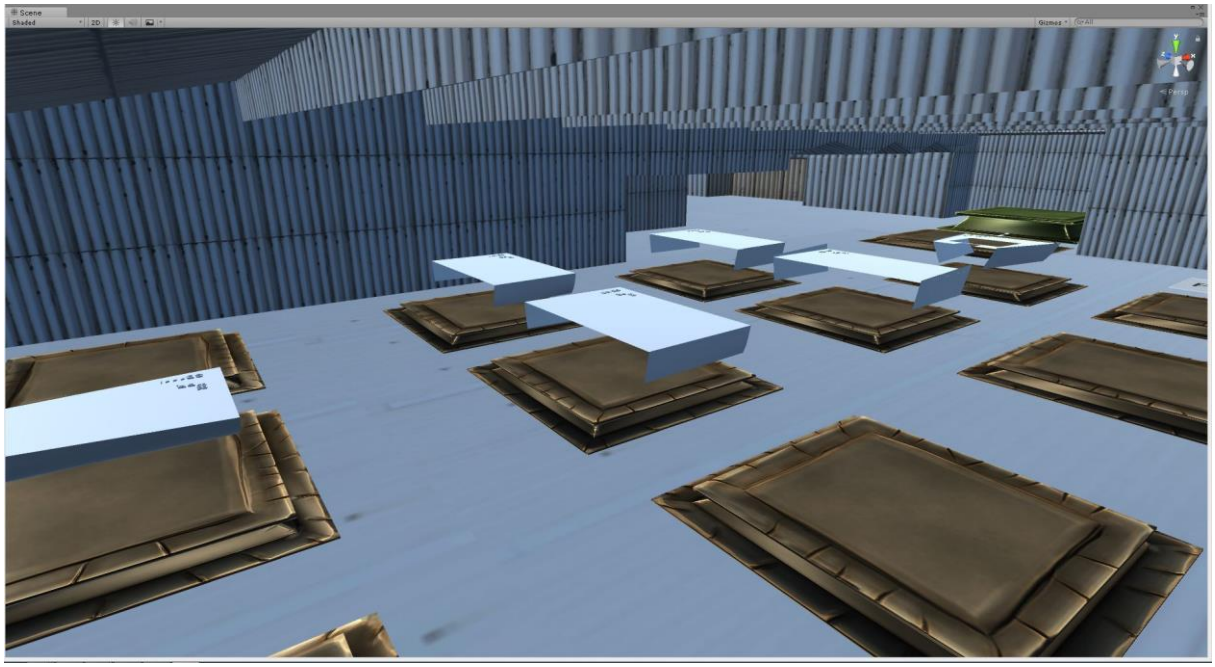

Figura 13. Ítems instanciados en las respectivas estaciones de trabajo “UnityWorkStation” en el caso de demostración. Fuente: elaboración propia.

iv. Servidores:
a. “ServerProcess”. Esta clase es el núcleo o núcleos de las estaciones ensambladoras y de trabajo. Es la encargada de programar el evento de procesado, así como de almacenar los ítems en caso de los procesos de ensamblaje, a la espera de que lleguen el resto de componentes. Como se ha mencionado, los tiempos de procesado son obtenidos de los ítems entrantes.

v. Fuente infinita:
    a. “InfinitySource”. Es la primera versión de la clase que modela el comportamiento de la fuente. En este caso, se crea un ítem siempre y cuando el siguiente elemento esté libre para recibirlo. En caso contrario, se quedará bloqueada hasta que se libere el elemento.
    b. “UnityInfinitySource”. La función principal de esta clase es la de generar el elemento virtual correspondiente del ítem que se ha creado, una vez que se ejecuta el evento de su creación. El método encargado es el ”generateItem”. A partir de ahí, dicho objeto virtual solo podrá ser modificado por una estación ensambladora (conformará el nuevo ítem como composición de sus subítems), o por el sumidero (que destruirá el elemento). En el resto de elementos, circulará a través de ellos a través del método “loadItem”, heredado de la interfaz “VElement”, que gestiona la posición del objeto en el mundo virtual.

vi. Fuente con programación de llegadas:
    a. “ScheduleSource”. Partiendo de la clase “InfinitySource” contemplada en las librerías iniciales, esta modificación permite leer de archivo las características de los ítems, y programar su creación según el día establecido en la base de datos correspondiente. No obstante, posteriormente se programa un modo alternativo de funcionamiento en el que se crea un ítem por orden de la clase de Unity, una vez que el usuario decide crearlo.
    b. “UnityScheduleSource”. Su funcionamiento es similar a la de la clase “UnityInfiniteSource”. La única diferencia reside en el modo alternativo en el cual el usuario puede realizar la planificación de la producción a partir de una interfaz. Es este por tanto el que genera los eventos de creación de ítems. Estos eventos son programados en la clase del nivel base a partir de una orden de esta clase, retornando posteriormente la ejecución del método “generate Item”, que general el bloque virtual. Este modo será explicado con mayor claridad más adelante.

vii. Cola:
    a. “ItemsQueue”. Las colas son modeladas a través de esta clase, la cual almacena el ítem en caso de no poder ejecutar el envío. Tienen una capacidad máxima, la cual establece el modelador desde Unity.
    b. “UnityQueue”. El ítem aparece en la cola en caso no poder ejecutar el envío (*Figura 14*).

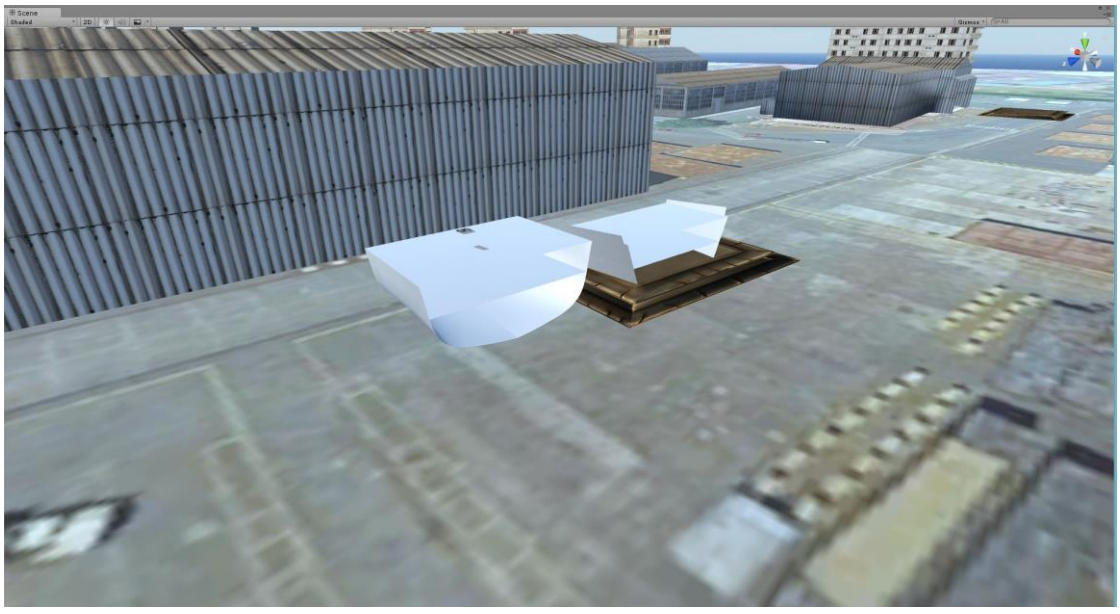

Figura 14. Ítems esperando en cola “ItemsQueue” en el caso de demostración. Fuente: elaboración propia.

viii. Estación de trabajo:
   a. “WorkStation”. Apoyándose en la clase “ServerProcess”, gestiona la recepción de los ítems, el bloqueo y desbloqueo de la estación, y la entrega al link del ítem al final del proceso.
   b. “UnityWorkStation”. Mientras el ítem se procesa, su objeto virtual se genera en la estación, a través del “loadItem” de esta clase.

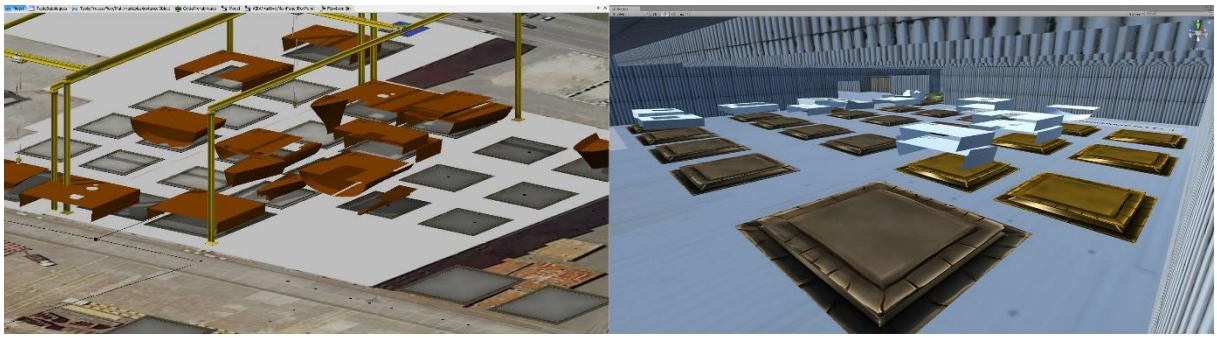

Figura 15. Comparación entre FlexSim y Unity: Ítems instanciados en sus respectivas estaciones de trabajo. Fuente: elaboración propia.

   c. “SimpleTransporter”. En este caso, es una clase generada específicamente a partir del caso de demostración, pero con intención de desarrollarla de la manera más genérica posible. Esto implica que la mayor parte de sus métodos son aplicables a otros casos, pero la declaración de algunos campos se realiza específicamente para el movimiento de una grúa. Su función es la de gobernar el transporte de un ítem por una grúa, lo cual se traduce en el nivel base en una estación de trabajo, pero a nivel virtual, requiere una clase diferente.

Figura 16. Ejemplo de clase “SimpleTransporter” en grúa para el caso de demostración. Fuente: elaboración propia.

ix. Estación de ensamblaje:
   a. “Assembler”. Las funcionalidades de esta clase son similares a la de “WorkStation”, pero su modo de funcionamiento difiere. En este caso, apoyándose en “ServerProcess”, se espera la llegada de todos los subítems que componen el ítem para ejecutar el evento de fabricación. Por tanto, en la recepción se comprueba el estado del servidor, así como su “tipo”: si ha recibido ya un subítem, tiene que comprobar que recibe la otra u otras partes

del ítem. El número de subítems es una variable que puede modificar el usuario modelador.

b. “UnityAssembler”. La única diferencia con “UnityWorkStation” es que, una vez recibe los dos subítems y termina el proceso de ensamblaje, destruye ambos objetos virtuales y carga la composición de los mismos, esto es, el ítem resultante (*Figura 17*).

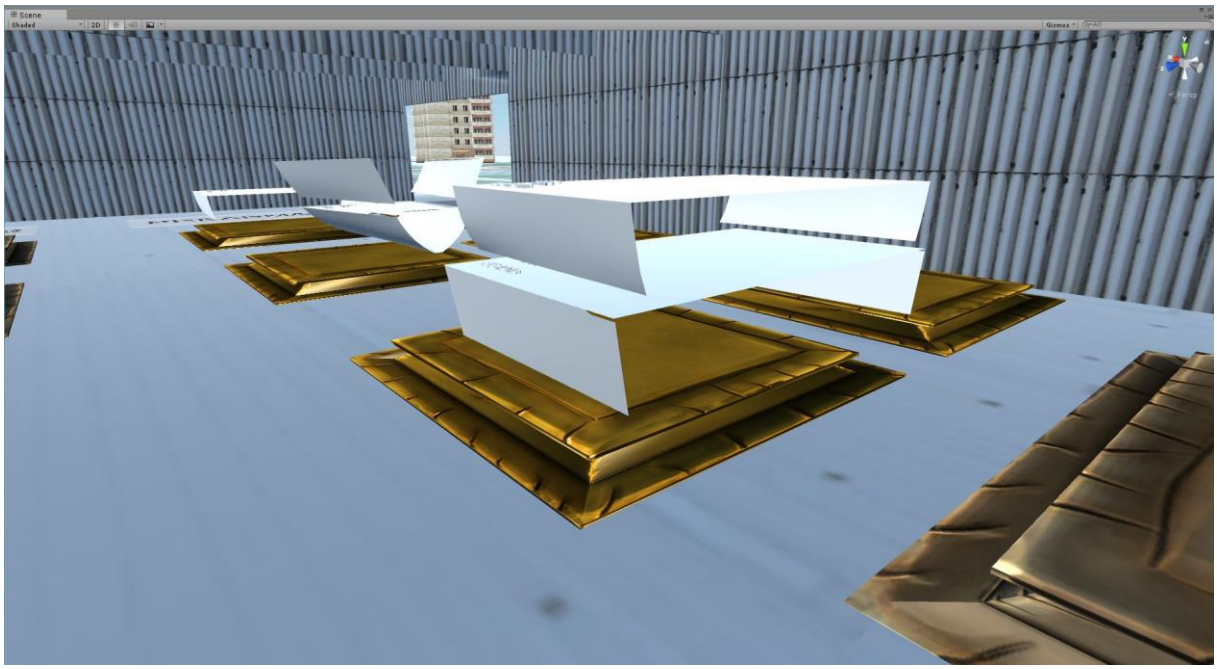

Figura 17. Funcionamiento de la clase “UnityAssembler” para el caso de demostración. Fuente: elaboración propia.

x. Sumidero:
   a. “Sink”. A efectos de la simulación, su única función es la de servir de sumidero del modelo, terminando el recorrido del ítem una vez llega a él. Sin embargo, se implementa a mayores una función de resultados, guardando en un archivo todos los tiempos de entrada y salida de cada ítem en cada proceso, los cuales fueron grabados en el objeto ítem en su paso por los mismos.
   b. “UnitySink”. Destruye el objeto virtual ítem una vez este llega al sumidero pudiendo activarlo en una cierta localización final.

Con todo lo anterior, se ha descrito brevemente el funcionamiento de la mayor parte de las librerías desarrolladas para este proyecto y que conciernen a la simulación. Para ilustrar dichos comportamientos, se han empleado imágenes del caso de demostración del presente trabajo. No obstante, como se ha mencionado al comienzo de este capítulo, se desarrollan simultáneamente varios proyectos en Unity donde se prueba y se depura el funcionamiento del código y se exploran ciertas posibilidades gráficas.

El primer modelo desarrollado en Unity contiene unas librerías iniciales que difieren a las empleadas en el caso de demostración. Emplea las clases “InfinitySource” y “SimpleLink”, con “WorkStation en su versión inicial. Es decir, los ítems se crean indefinidamente siempre y cuando la primera estación de trabajo quede libre. Solo existe en cada paso una única estación de trabajo, siendo el flujo de ítems unitario. Los tiempos de procesado son aleatorios y proporcionados por la librería “SimValues”, y no se emplea la clase “Assembler”, pues fue desarrollada más adelante. El resultado del modelo se muestra en la *Figura 18*.

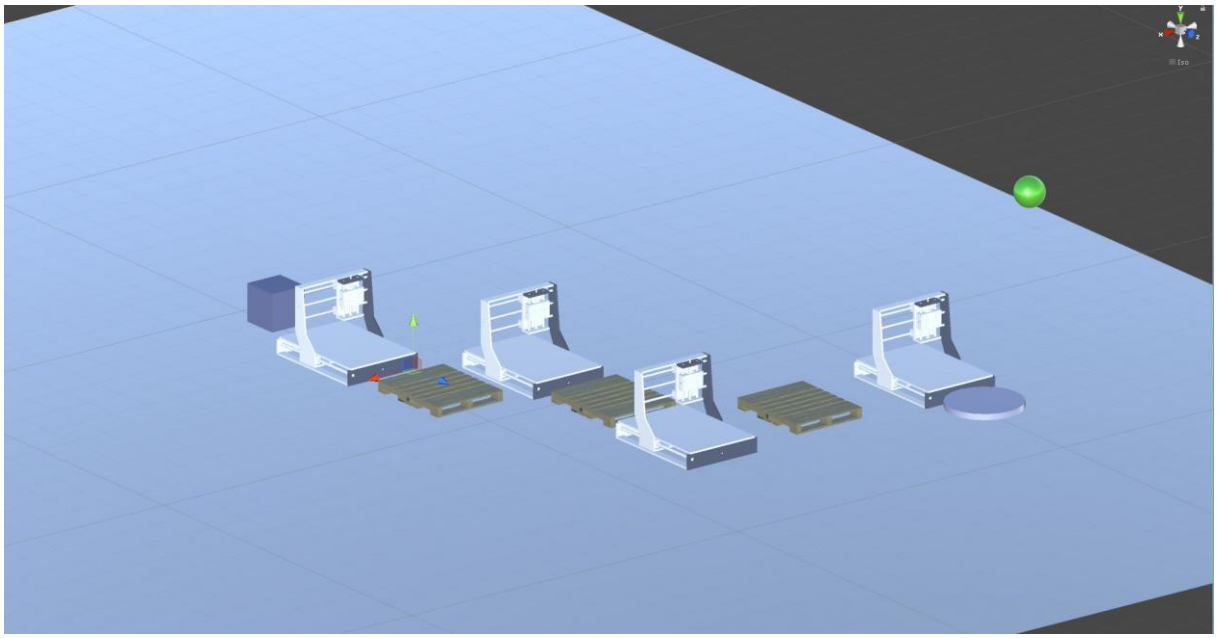

Figura 18. Primera versión de un modelo genérico desarrollado en Unity. Los palés representan las colas, el cubo la fuente y el cilindro el sumidero, siendo los elementos restantes las fuentes. Los ítems son representados a través de esferas verdes. Fuente: elaboración propia.

Posteriormente, tras otras pruebas, se realiza un segundo proyecto en el que ya se implementan unas versiones preliminares de las clases “MultiLink”, “UnityScheduleSource” y “Assembler”, aún en fase de pruebas. Además, las clases “ServerProcess” y “WorkStation” siguen siendo las originales, pues estas se simplifican más adelante. Por otra parte, esta versión se desarrolla tras escoger el caso de demostración. Por tanto, se empiezan a realizar las primeras pruebas de los recursos gráficos, tal y como se aprecia en la *Figura 19*.

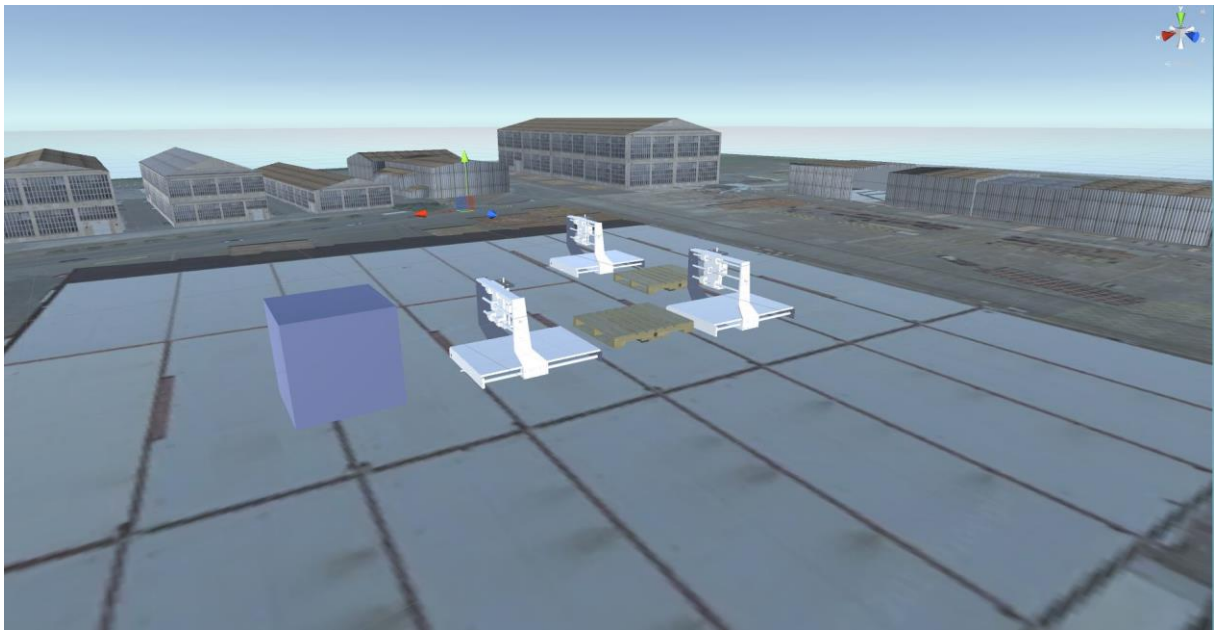

Figura 19. Una de los proyectos previos de Unity. El él ya se aprecian las versiones preliminares de algunos edificios y del terreno del astillero. Fuente: elaboración propia.

4.3. Desarrollo de la interfaz

Se explica a continuación el funcionamiento de la librería “Interface”, la cual contiene todos los *scripts* que gobiernan el comportamiento de los distintos elementos de la interfaz de usuario. Es importante remarcar que se continúa con la premisa de realizar una programación genérica, pero existen ciertas opciones que, como se indicará, son específicas del entorno del caso de demostración aquí empleado, aunque reutilizables en otros modelos con sendas características. De nuevo, para explicar esta librería, se emplea el caso de demostración.

El primer paso en la creación de interfaz es el de fijar el objetivo principal de la misma: proporcionar al usuario todas las herramientas gráficas para permitir un seguimiento exhaustivo y detallado de la simulación. En este sentido, se agrupan en grupos los aspectos relativos a la estructura como *serious game, a* los datos de la simulación y el punto de vista del usuario, y al control tanto de la propia interfaz como de las propiedades temporales de la simulación. Es decir, podemos dividir la interfaz gráfica desarrollada en tres partes fundamentales:

- La estructura de la aplicación como *serious game*. Una vez desarrollado el proyecto en Unity, se construye una aplicación que puede ser ejecutada desde cualquier ordenador. En este contexto, se considera necesario una serie de menús inicial y final que permitan al usuario ejercer cierto grado de control no sobre la simulación, sino sobre la aplicación en sí misma. Con este pretexto, se desarrolla:
    - una escena inicial (*Figura 20*) con un menú donde el usuario pueda iniciar la simulación a través de un botón ”Start Simulation”, salir de la aplicación (botón “Exit”), o bien obtener información sobre lo que se va a ejecutar (botón “Information”);

Figura 20. Ejemplo de menú inicial para el caso de demostración. Fuente: elaboración propia.

    - y una escena final (*Figura 21*) con otro menú donde pueda salir de la aplicación, reiniciar la simulación (“Restart Simulation”), y con una opción a mayores que permita generar un informe de la simulación (“Generate Report”) con una serie de datos finales que resulte de interés conocer.

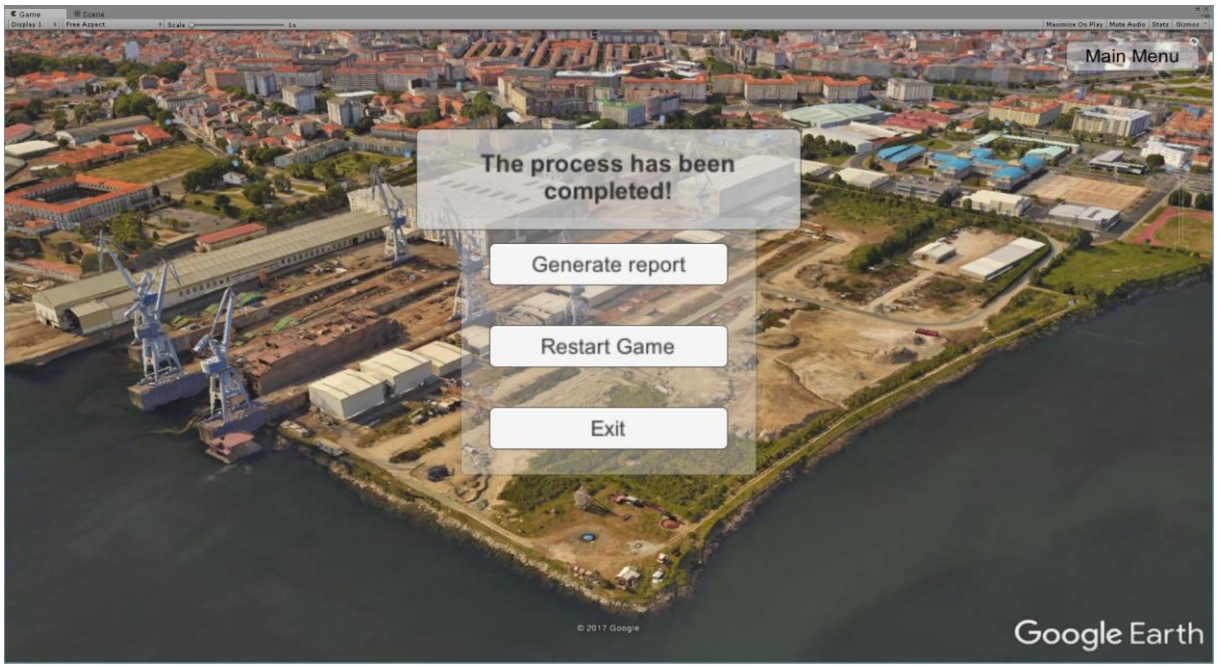

Figura 21. Ejemplo de menú final para el caso de demostración. Fuente: elaboración propia.

- La interacción del usuario con el mundo virtual. Constituye una de las partes fundamentales del proyecto, pues le permite al usuario explorar el entorno virtual y obtener información actualizada en tiempo real de la simulación. Dentro de ella podemos diferenciar a su vez dos aspectos:
    - El control de la cámara. Con el objetivo de diseñar un entorno lo más inmersivo posible, el usuario podrá controlar la vista de la cámara principal de la misma manera que lo haría en un videojuego. Con las teclas de dirección[6] combinado con el movimiento del ratón, podrá moverse por el escenario donde se desarrolla la simulación, entrar en los diferentes edificios, y observar cómo se completar el proceso.

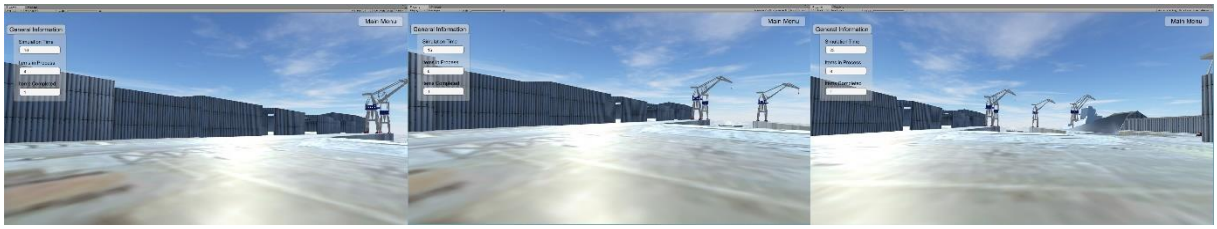

Figura 22. Secuencia de movimiento de la cámara en primera persona para el caso de demostración. Fuente: elaboración propia.

    - Obtención de información. En cualquier *software* de simulación comercial de eventos discretos, el principal interés del modelador es la obtención de datos a partir de los cuales extraer conclusiones. De esta manera, este aspecto se ha respetado como uno de los fines principales en la creación del proyecto: a medida que el usuario explora el entorno virtual, puede pulsar el botón izquierdo del ratón sobre cualquier elemento de la simulación para obtener información concerniente al mismo. En la *Figura 23* puede observarse un ejemplo de ello. Por otra parte, en la *Figura 24*, se muestran los datos que ofrece cada elemento de la simulación según la clase a la que pertenezca.

[6] Con “teclas de dirección” se hace referencia a las flechas del teclado o, en su defecto, las teclas “W” (hacia delante), “S” (hacia atrás), “A” (hacia la izquierda), “D” (hacia la derecha).

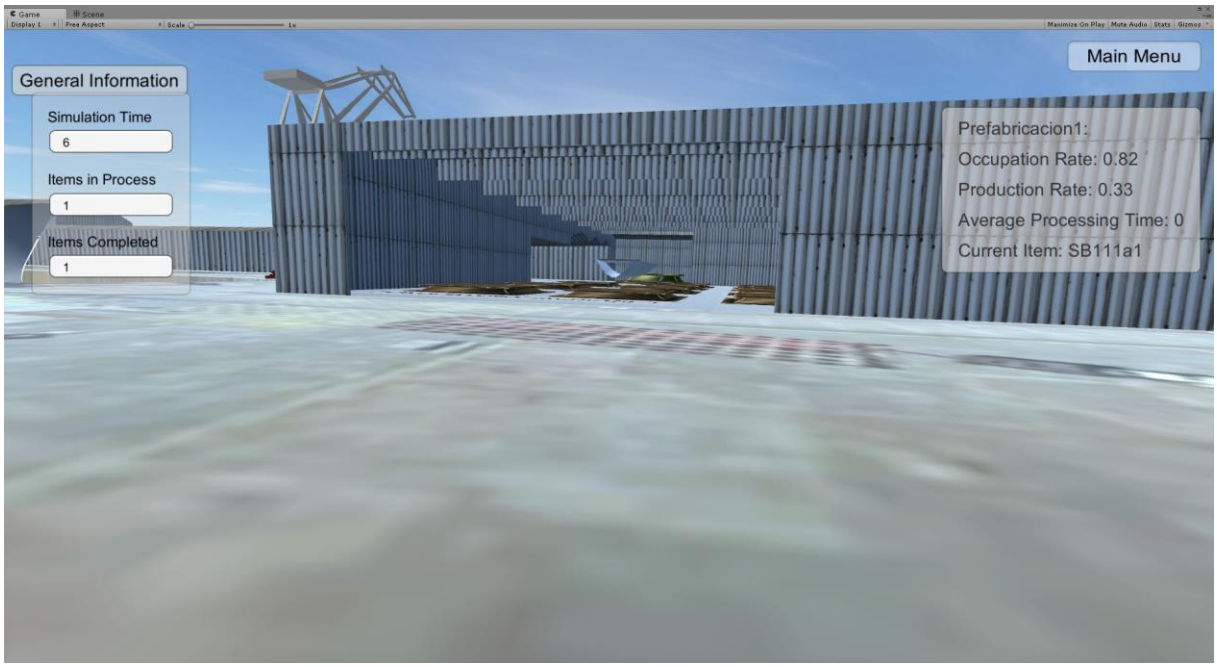

Figura 23. Detalle de paneles: a la derecha de la imagen, panel con datos de la estación de trabajo; a la izquierda, panel de información general del proceso. Fuente: elaboración propia.

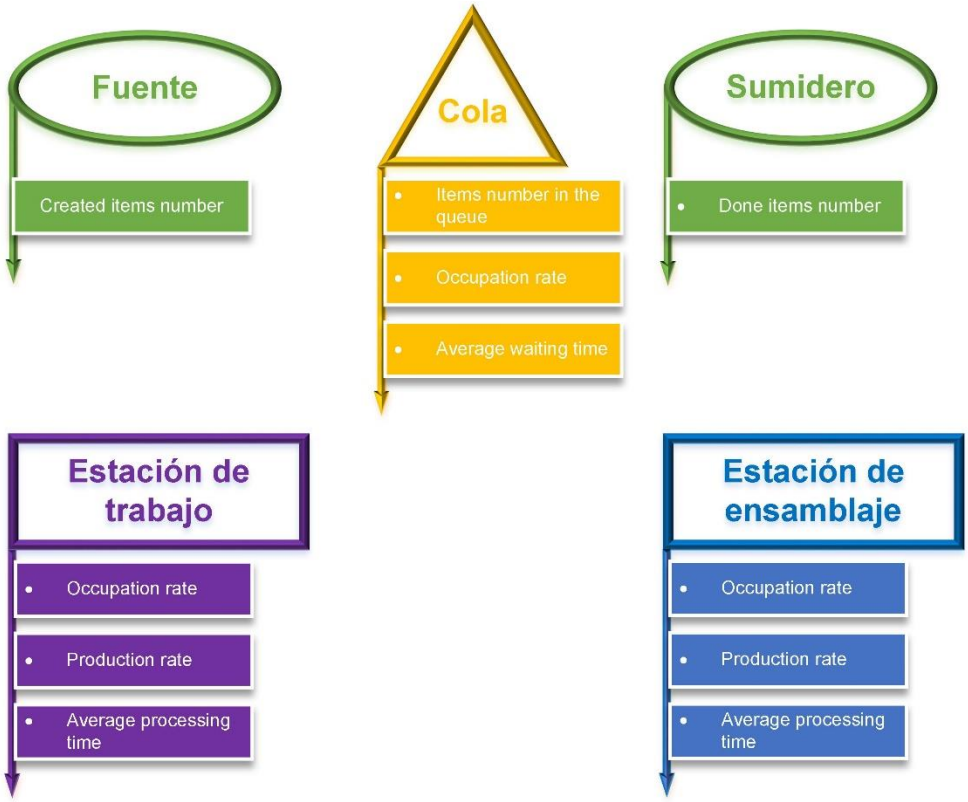

Figura 24. Diagrama explicativo de la información que se muestra en cada elemento durante la simulación. Fuente: elaboración propia

- La modificación de parámetros relativos a la experiencia virtual. La segunda parte de la interfaz engloba una serie de paneles y menús desde los cuales controlar los distintos parámetros tanto de la experiencia como del proceso simulado. Estos menús se pueden únicamente desplegar en la escena principal, y los elementos que los componen se mencionan a continuación:

- Panel de selección de modo (*Figura 25*). Este menú se despliega únicamente antes del inicio de la simulación, una vez el usuario pulsa el botón “Start Simulation” en la escena inicial, y cuando reinicia la simulación. En él, el usuario puede elegir entre dos modos de funcionamiento ya mencionados previamente, dando comienzo seguidamente a la simulación:
  - “Modo 0”. Funcionamiento normal de la simulación donde la fuente crea los ítems según unas fechas leídas de una base de datos.
  - “Modo 1”. Se despliega un segundo panel donde aparecen las imágenes de los diferentes ítems que se procesarán, con un código identificador. Una vez el usuario pulse en una imagen, esta desaparecerá programando instantáneamente la llegada de dicho ítem. Esta opción resulta especialmente interesante, al permitir la planificación de la producción por parte del usuario a través de un entorno virtual (*Figura 26*)

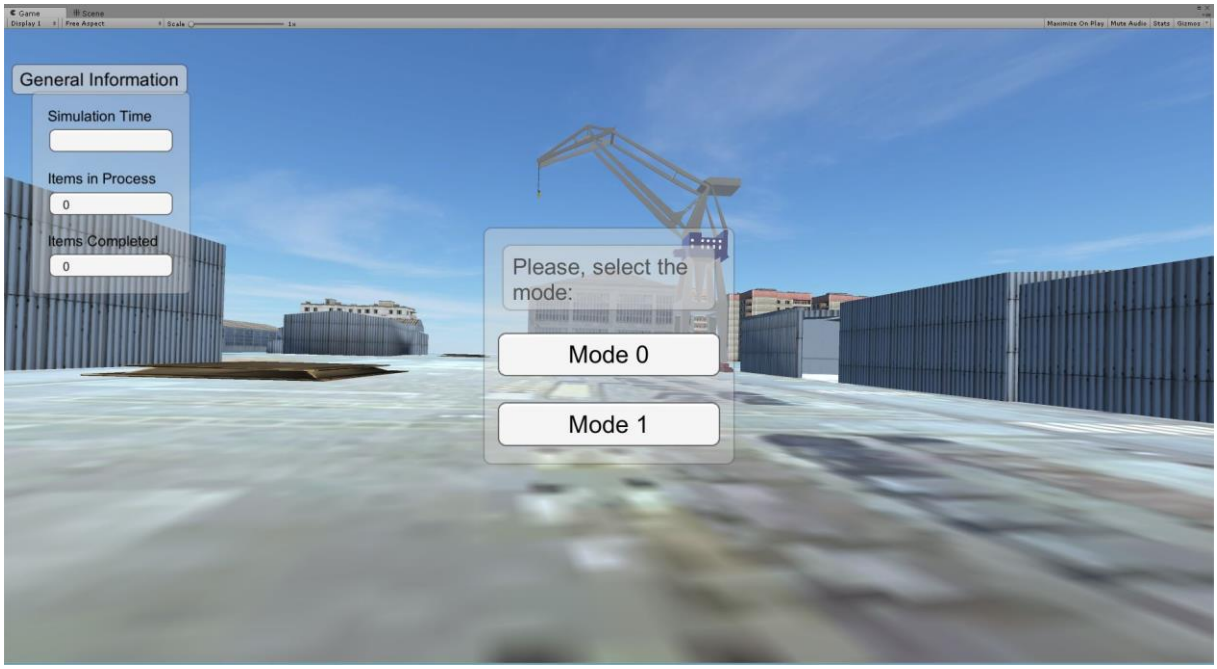

Figura 25. Detalle de panel de selección de modo en el caso de demostración. Fuente: elaboración propia.

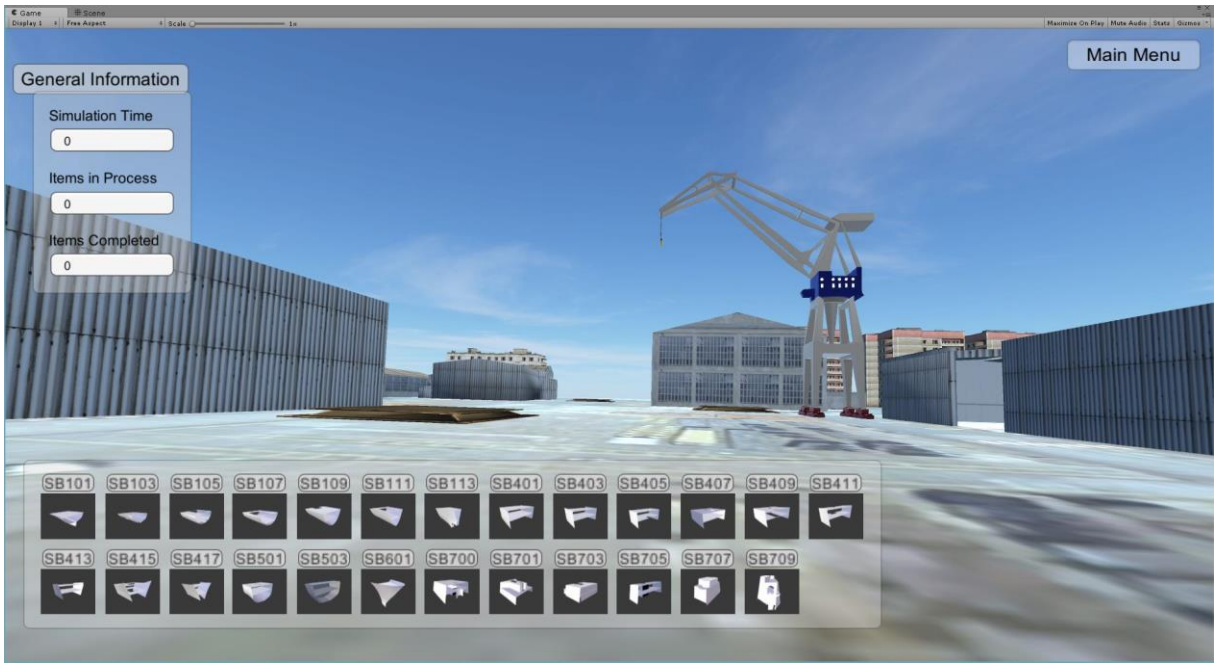

Figura 26. En la parte inferior de la imagen, panel de los ítems del caso de demostración; en la parte superior derecha, botón de menú principal. Fuente: elaboración propia.

- Panel de opciones generales (*Figura 23*). Con posibilidad de desactivarlo desde el menú de opciones, muestra los datos principales de la simulación:
  - Tiempo de simulación actual.
  - Ítems en proceso[7].
  - Ítems completados.
- Botón de menú principal (*Figura 26*). Podemos acceder al menú principal pulsando con el ratón directamente en el botón, o a través de la tecla “P”. Una vez esté abierto el menú, la simulación se detendrá. Solo continuará cuando se cierre de nuevo cualquier panel de menú.
- Panel de menú principal (*Figura 27*). Permite las siguientes funcionalidades:
  - “Resume”: cierra el menú y continúa la simulación.
  - “Options”: accede al menú de opciones.
  - “Restart Simulation”: detiene la simulación y despliega el menú de selección de modo.
  - “Exit”: sale de la simulación y carga el menú final.

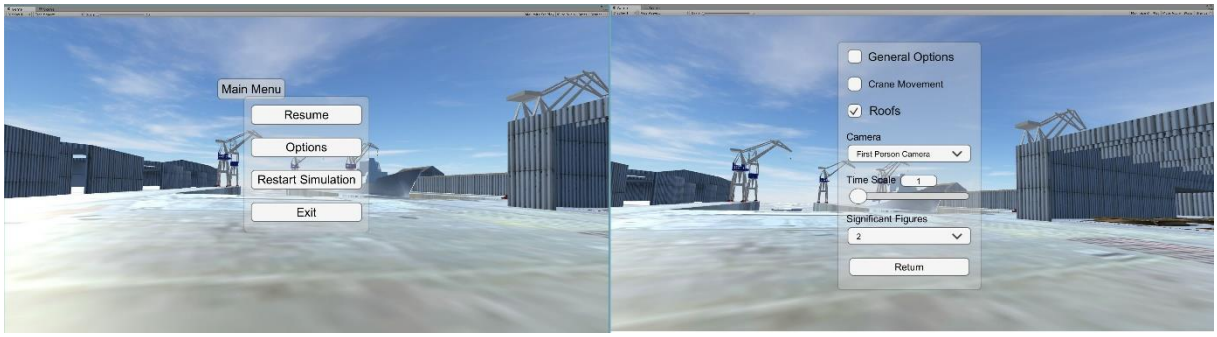

Figura 27. Menús principal y de opciones para el caso de demostración. Fuente: elaboración propia.

[7] Cabe comentar que, una vez se ensamblen dos subítems, pasarán a ser un único ítem que será contabilizado en el cómputo total como tal.

- Panel de menú de opciones (*Figura 27*). Permite las siguientes funcionalidades:
  - Botón “General Options”: activa y desactiva el panel en el que se muestran las opciones generales durante la simulación. Por defecto está activado.
  - Botón “Crane Movement”: esta opción se desarrolla específicamente para el caso de demostración, donde existe el movimiento de unos bloques por una grúa. Lo que permite es activar y desactivar la visualización del movimiento de la grúa. Sin embargo, con respecto al tiempo de procesamiento de los bloques, el tiempo de movimiento de la grúa es demasiado pequeño como para que tenga impacto alguno en el proceso. Por tanto, para poder visualizar tal movimiento, es necesario disminuir la escala de tiempo; esto es, una vez se active la opción, la escala de tiempo se reducirá permitiendo la visualización. Por defecto se encuentra desactivada (*Figura 27*).
  - Botón “Roofs”: es otra de las opciones que se desarrollan a partir del caso de demostración. En este, el proceso se desarrolla en varios talleres, y a través de este conmutador el usuario puede ocultar los tejados de los edificios en los que se desarrolla alguna parte de la simulación. Esta opción es especialmente interesante cuando el usuario tiene activada la vista general (“Overhead Camera”). Por defecto se encuentran activados (*Figura 28*).

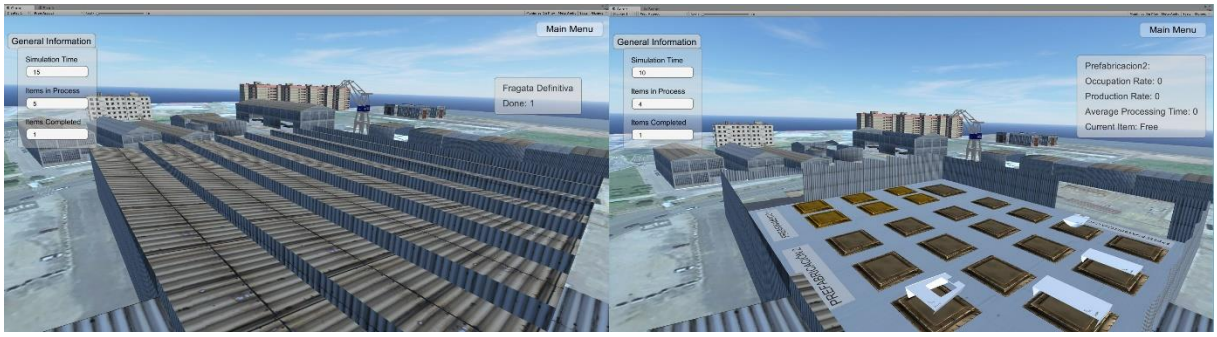

Figura 28. Comparación de imágenes con la opción tejados activados (izquierda) y desactivados (derecha). La vista en ambas imágenes es la satélite del caso de demostración: “Overhead Camera”. Fuente: elaboración propia.

  - Menú desplegable “Camera”: permite elegir la vista en primera persona “First Person Camera”, o una vista satélite de la escena “Overhead Camera”. En ambos casos, el usuario podrá desplazarse con normalidad. El valor predeterminado es la vista en primera persona (*Figura 28*).
  - Barra de desplazamiento “Time Scale”: permite ajustar la velocidad a la que transcurre la simulación. De esta manera, si el usuario se encuentra en un valor 10, la simulación estará avanzado 10 veces más rápido del valor predeterminado. El rango admitido es de 1 a 20 (*Figura 29*).
  - Menú desplegable “Significant Figures”: esta opción permite elegir el número de cifras significativas con las que se muestran los datos de la simulación de los diferentes elementos. El usuario podrá elegir mostrar 2, 3 o 4 cifras (*Figura 29*).
  - “Return”: al pulsar, regresa al panel del menú principal.

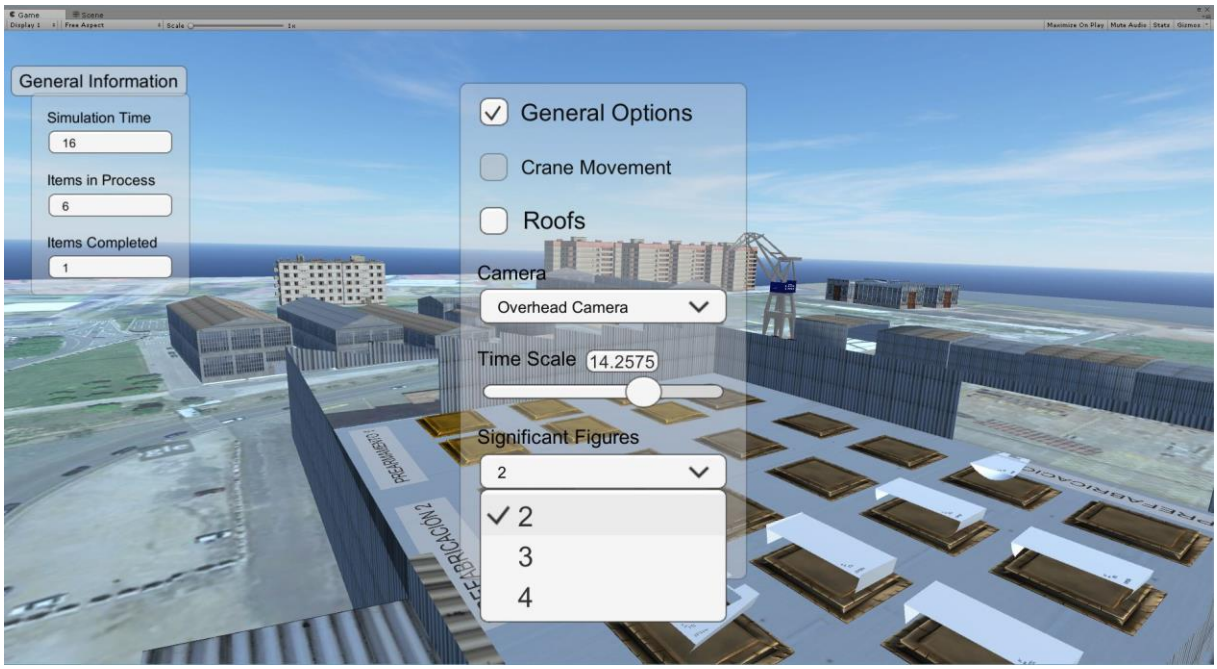

Figura 29. Vista del panel de opciones con la escala de tiempo modificada y el panel desplegable de cifras significativas a la vista. De fondo, vista satélite del caso de demostración. Fuente: elaboración propia.

Finalmente, se han añadido una serie funcionalidades a mayores con la premisa de permitir una experiencia más flexible y dinámica:

- Si en cualquier momento durante la simulación el usuario pulsa la tecla “G”, se desactivará o activará el panel de opciones generales para permitir una mejor visión del entorno.
- Si en cualquier momento durante la simulación el usuario pulsa la tecla “Escape”, saldrá al menú final.
- Si en cualquier momento durante la simulación el usuario pulsa la tecla “Control”, se bloqueará el movimiento de la cámara mientras lo mantenga pulsado. Esto permite acceder al menú sin necesidad de mover la vista actual.
- Si en cualquier momento durante la simulación el usuario pulsa la tecla “Shift”, se acelerará el desplazamiento de la cámara por la escena.

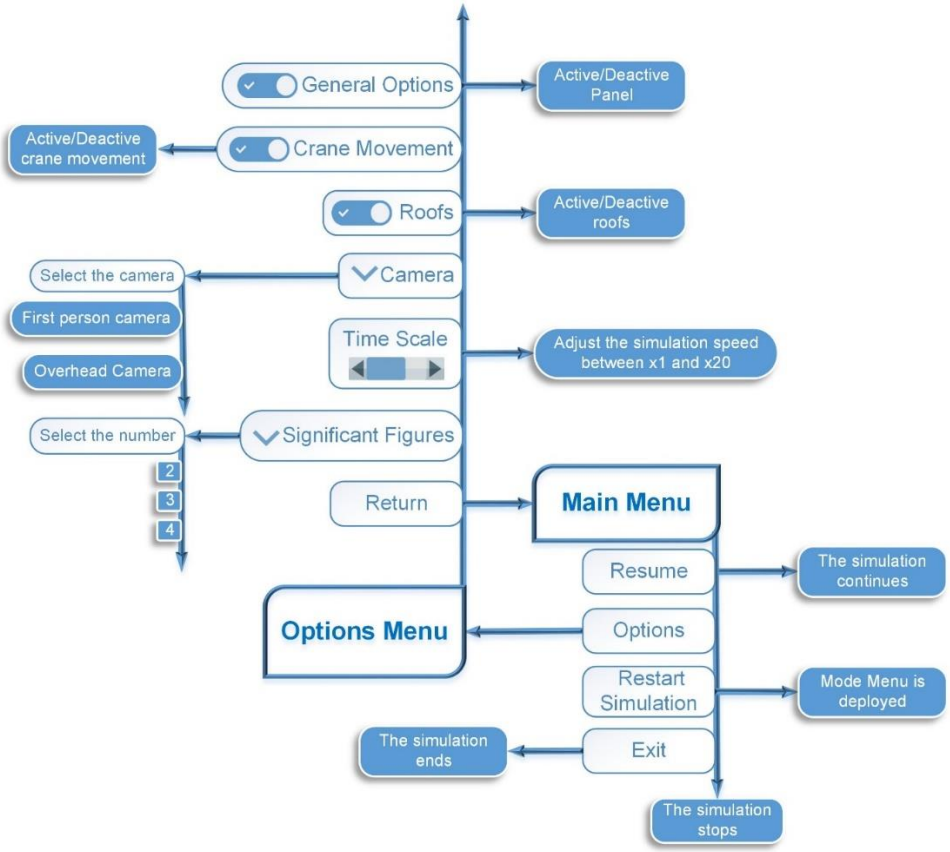

Figura 30. Esquema general del funcionamiento de los menús principal y de opciones de la interfaz gráfica de usuario. Fuente: elaboración propia.

4.4. Descripción del caso de demostración

El caso de demostración escogido el proceso de armado y montaje de los subbloques y bloques de una fragata. Concretamente, la zona en la que se desarrolla este proceso viene indicada en la *Figura 31*.

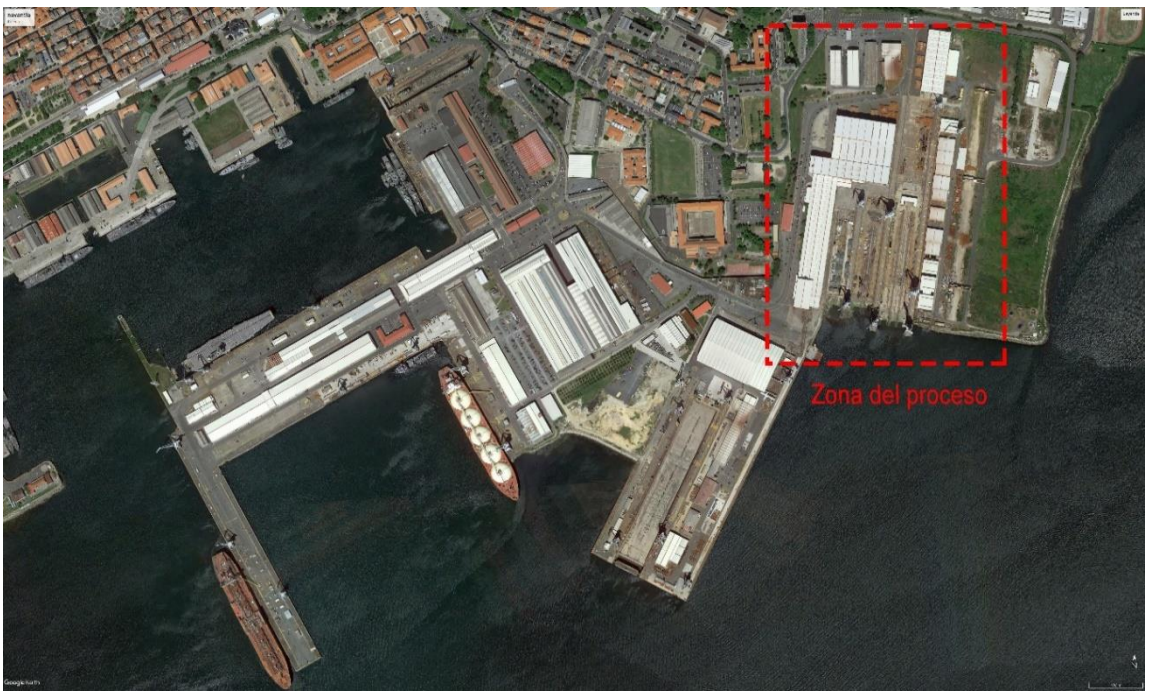

Figura 31. Imagen global del astillero. En rojo, la zona de interés. Fuente: elaboración propia.

Dentro de la zona de interés, el proceso se conforma por una serie de subprocesos que se distribuyen en los siguientes talleres o zonas:

- Taller de elaboración y previas. No está contemplado en el modelo de FlexSim.
- Taller de prefabricación 1.
- Taller de prefabricación 2.
- Taller de prearmamento 1.
- Taller de prearmamento 2.
- Cabinas de chorro y pintura.
- Colocación en grada.

La situación geográfica de estos talleres se indica en *Figura 32*:

Figura 32. Situación geográfica de los talleres de la zona de interés. Fuente: elaboración propia.

Antes de continuar con el estudio del proceso, cabe aclarar que, como ya se indicó previamente, si bien es importante comprender la situación y el flujo de los bloques por los diferentes talleres, no resulta de interés conocer en detalle las operaciones que se realizan en cada uno más que a efectos de número de estaciones y de tiempos.

Con respecto a la caracterización del proceso, el diagrama de flujo de la *Figura 33* representa el camino que recorren los diferentes bloques y subbloques:

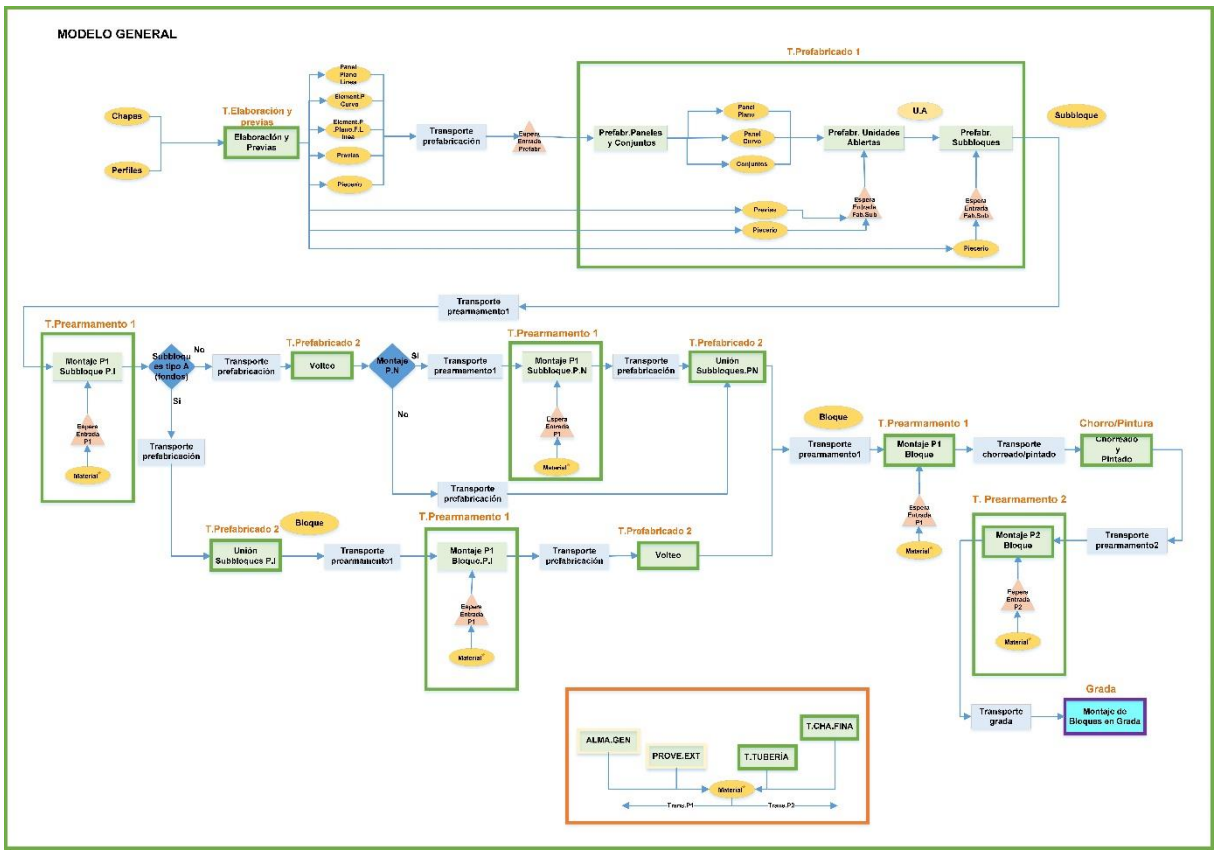

Figura 33. Diagrama de flujo del proceso de montaje de la fragata. Fuente: Unidad Mixta de Investigación de la Universidade da Coruña.

Obviando el taller de elaboración y previas por no contemplarse en el modelo de FlexSim, a partir del diagrama y con información proporcionada por la UMI, se realiza un pequeño resumen de los pasos fundamentales de los bloques por el proceso:

i. Taller de prefabricación 1: en este taller se produce el montaje de los subbloques a partir de los diferentes componentes fabricados en el taller de elaboración y previas.
ii. Taller de prearmamento 1: dentro de este taller se pueden diferenciar tres etapas:
    a. 1ª etapa: se corresponde con el 80% del total del prearmamento de cada subbloque. Siendo el primero a la derecha en el diagrama de la *Figura 33*, todos los subbloques pasan por él.
    b. 2ª etapa: se corresponde con el 10% del prearmamento. Como se aprecia en el diagrama, podrán pasar por él bloques o subbloques pues, dependiendo de sus características, tendrán el proceso de unión previamente.
    c. 3ª etapa: corresponde al otro 10% del prearmamento. A la vista del diagrama, es la última fase de prearmamento y se realza a nivel de bloque.
iii. Taller de prefabricación 2: La unión y volteo de los subbloques se realiza en este taller, con lo que, tras las diferentes etapas de prearmamento, el bloque o subbloque regresa al taller para acometerse dichas operaciones.
iv. Taller de pintura: taller en el que se procede al chorreado y pintado de cada bloque.
v. Taller de prearmamento 2: último subproceso de montaje a nivel de bloque.
vi. Colocación en grada: finalmente, se monta el bloque en grada.

4.5. Modelización conceptual del caso de demostración

Continuando con la misma idea, la modelización conceptual del caso de demostración se realiza de tal manera que queden patentes las potencialidades de Unity. Es decir, no se busca replicar con exactitud el modelo proporcionado en FlexSim sino que se plantea un nuevo modelo con ciertas simplificaciones. En concreto, en la versión final del modelo, se plantea la siguiente cadena de flujo:

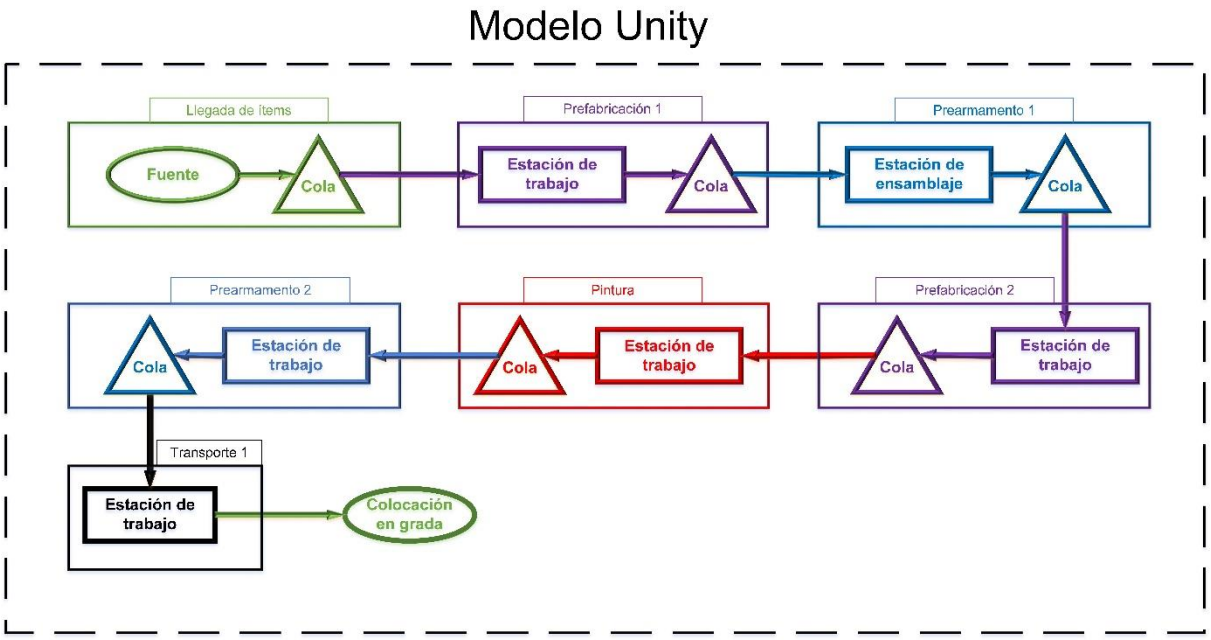

Figura 34. Diagrama de flujo del modelo de Unity del caso de demostración. Fuente: elaboración propia.

Como es observable, el modelo cuenta con 6 actividades diferenciadas, que tendrán una localización definida en planta, y separadas entre ellas por colas. El proceso comienza con la actividad “Fuente”, y termina con “Colocación en grada”, sin suponer esta última un tiempo añadido. Además, a diferencia del modelo de FlexSim, los bloques no pasan dos veces por el mismo taller, lo cual supone la principal diferencia entre ambos modelos.

Es importante tener en cuenta que la unidad de tiempo en el reloj de simulación será el día, por las características del proceso. Todos los tiempos proporcionados por UMI están en días.

A continuación, se identifican y los elementos definidos en el apartado *4.1 Modelización conceptual de una simulación genérica*:

- Fuente: “Llegada de ítems” conforma la única fuente del modelo, cuya función será la de programar la llegada de subbloques a la planta. Al suprimirse el taller de elaboración y previas, se toman los tiempos resultado de la simulación del modelo de FlexSim para la entrada de los ítems en el proceso.
- Cola: las actividades “cola” gestionarán el paso de los ítems por la cadena. Cabe destacar que la cola previa a la estación de ensamblaje deberá comprobar si tiene el subbloque correspondiente al que ya se halle en la misma.
- Estaciones de trabajo: las actividades “Prefabricación 1”, “Prearmamento 1”, “Pintura”, “Prearmamento 2” y “Transporte 1” son, a nivel conceptual, estaciones de trabajo. Existirán varias estaciones de cada actividad, las cuales serán agrupadas por talleres según la situación geográfica de los mismos. Cada estación, de capacidad unitaria, procesará los ítems según tiempos obtenidos de la simulación de FlexSim. Cabe comentar el caso particular de la actividad “Transporte 1”, que se corresponde con el transporte por grúa. Por tanto, esta última será instanciada por una clase “SimpleTransporter” de Unity.

- Estación de ensamblaje: la actividad “Prefabricación 2” constituye el único elemento ensamblador de la cadena, al agrupar los subbloques correspondientes dos a dos para formar el bloque, según un identificador. De nuevo, el tiempo de procesamiento es obtenido de la misma forma que en las estaciones de trabajo.
- Sumidero: la actividad “Colocación en grada” es el sumidero o punto final del modelo. Una vez los bloques lleguen a ella, saldrán de proceso de fabricación, y se activarán en el barco final, que se irá montando a medida que los bloques finalicen el proceso. Como se ha explicado, no constituye un tiempo más en el modelo, sino que el montaje se realiza de manera instantánea (*Figura 35*).

Figura 35. Actividad “Colocación en grada”, donde el barco se va construyendo según llegan los bloques. Fuente: elaboración propia.

- Reloj de la simulación: a pesar de necesitar un objeto virtual, este no se hace visible al no proporcionar valor añadido alguno. No obstante, exige realizar dos puntualizaciones muy importantes. Por un lado, tanto por las características del proceso como por los datos proporcionados por la UMI, la unidad de tiempo es el día. Por otro, y derivado de la primera, la sincronización con el tiempo físico se hace de tal manera que un segundo en Unity corresponde a un día en la simulación de eventos discretos.
- Ítems: tanto los bloques como los subbloques son conceptualmente ítems. Estos ítems tendrán una serie de propiedades que los identificarán como únicos. De hecho, tal y como se explicó en apartados anteriores, los ítems son los encargados de portar con todos los tiempos de cada actividad. Estas propiedades y tiempos constituyen la base de datos de la que leen las diferentes estaciones de trabajo y ensamblaje, y son establecidas en el momento de creación en la fuente. Se trata de un archivo de texto que, en este caso, contiene ordenados según el día de creación las propiedades de cada ítem con la configuración mostrada en la *Figura 36* y *Figura 37*.

Tipo	Identificador de subbloque	Prioridad	Identificado de bloque	Días de creación respecto al anterior	Taller	Días de procesado	Taller	Días de procesado	A partir de aquí sigue la misma configuración taller día

Figura 36. Configuración de la base de datos para el caso de demostración. Las líneas negras intermedias representan espacios de tabulador. Fuente: elaboración propia.

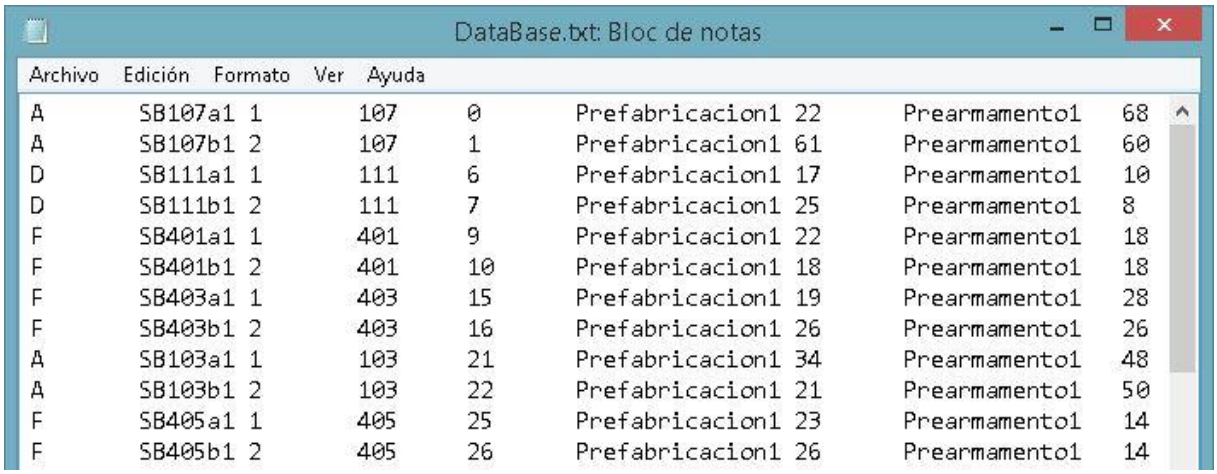

Figura 37. Ejemplo de base de datos con las propiedades almacenadas en los ítems. Fuente: elaboración propia.

- Conexiones: al igual que el reloj, se necesita un objeto virtual que porte el script, pero este es invisible para el usuario.
- Servidores: cabe comentar únicamente que esta clase, instanciada por las estaciones de ensamblaje, es la encargada de comunicarle al conector previo si el subbloque entrante es la parte complementaria al subbloque existente en ese momento en la estación.

4.6. Recursos gráficos

El objetivo de este apartado es informar de todo el trabajo realizado en lo referente a la parte gráfica del proyecto. Precisamente este aspecto acaba convirtiéndose en uno de los más destacados, suponiendo una fracción importante del tiempo total. Esto mismo se reflejará en la comparativa llevada a cabo en *5.1 Comparativa*, donde constituirá uno de los campos de comparación.

Por otra parte, es importante aclarar que con *recursos gráficos* se hace referencia al diseño, modelado y texturizado de todos los elementos visibles para el usuario. Estoy incluye el escenario en el que se desenvuelve la acción, los edificios (en total, 8 edificios), y los diferentes elementos de simulación.

Para elaborar esta parte del trabajo, se recurre a tres programas, dos de ellos gratuitos: Blender, un programa informático multiplataforma de *software* libre; SketchUp, un programa de diseño gráfico y modelado en tres dimensiones basado en caras; y Photoshop, un editor de gráficos rasterizados (véase *Figura 38*). Además, se parte de una serie de recursos proporcionados tanto por el Grupo Integrado de Ingeniería como por la Unidad Mixta de Investigación (en adelante, UMI) de la Universidade da Coruña. En concreto:

- Se parte de una modelización previa de los edificios en SketchUp, si bien se modificarán todos ellos profundamente. El empleo de estos recursos evitar tener que calcular proporciones entre sus diferentes partes. Cabe decir que el diseño es precario y en ningún caso tiene texturas aplicadas.
- Los bloques que conforman el barco son proporcionados en archivos CAD por la UMI. Como se explicará, también fue necesario modelarlos uno por uno.
- La grúa cigüeña del astillero Navantia – Ferrol es proporcionada por la UMI ya texturizada.

Figura 38. De izquierda a derecha: SketchUp, Blender y Phostoshop: programas empleados para el desarrollo de los recursos gráficos. Fuente: elaboración propia.

En cuanto a las tareas acometidas, se puede dividir el trabajo en cinco fases:

i. Creación del terreno del escenario. Para la confección del suelo por el que el usuario se mueve en primera persona, se emplea SketchUp por su herramienta de geolocalización. Lo que se realiza es una composición de imágenes obtenidas a través del mismo programa. Estas son exportadas y, a través de una herramienta de edición gráfica, se compone la imagen mostrada en la *Figura 39*. Posteriormente se importa de nuevo a SketchUp, y se modela la forma básica del terreno, que más tarde se empleará en Unity.

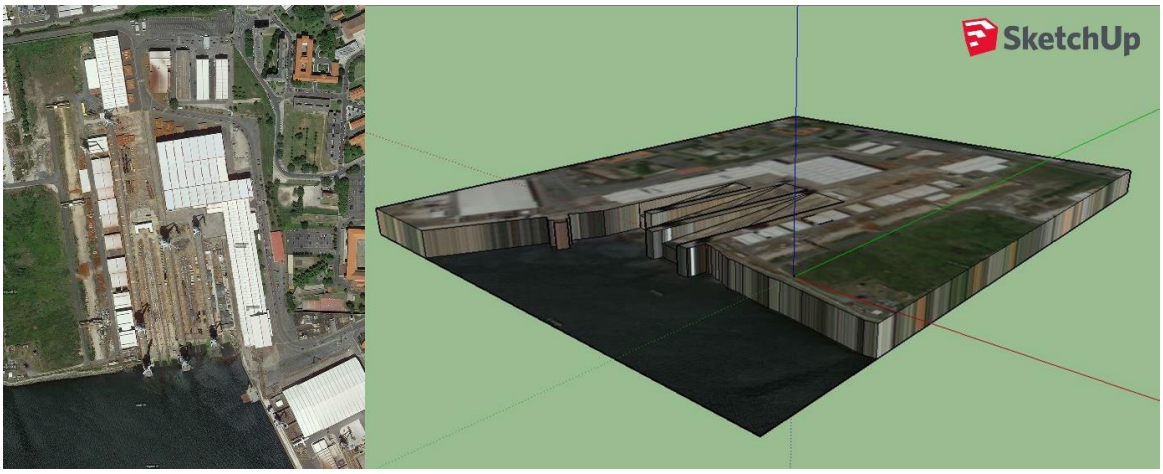

Figura 39. Proceso de confección el fondo: a la izquierda, la composición de las imágenes extraídas de SketchUp; a la derecha, la modelización del terreno en SketchUp. Fuente: elaboración propia.

ii. Modelado de los edificos. Esta tarea es desarollada inicialmente a través de SketchUp y completada con Blender. Inicialmente, el diseño de los edificios es corregido a través de SketchUp, Se aísla el edificio del archivo proporcionado, se corrigen las aristas y las caras, se separan ciertas partes del edifico para facilitar el modelado, y se le proporciona un suelo. Posteriormente, a través de Blender, se eliminan los vértices y aristas duplicados, se corrigen todas las caras, se orientan las normales, y se marcan las aristas que dividen las superficies. Por último, una vez está el edifico depurado, únicamente en aquellos donde se modela el interior por contener alguna parte del proceso simulado, se le da grosor al edificio (lo que a nivel gráfico conlleva una duplicación de las caras). Este último viene representado en la *Figura 40*.

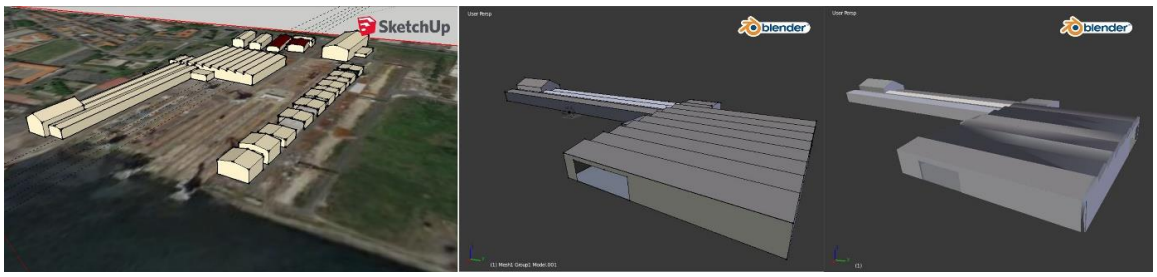

Figura 40. Ejemplo de modelado de los talleres de prefabricación 1 y prearmamento 1. De izquierda a derecha: archivo original de SketchUp; corrección de errores y aplicación de suelo en SketchUp y Blender; y aplicación de la resticción “Solidify” en Blender, que da grosor a las paredes. Fuente: elaboración propia.

iii. Asignación y configuración de los materiales para renderizado en tiempo real. Una vez se elaboran los modelos tridimensionales de los 8 edificios que componen el escenario, se asignan las texturas que darán lugar a los materiales renderizados en Unity en tiempo real. Al no poseer las texturas de los edificios reales, se extraen una serie de texturas de uso gratuito de Internet. Con estas texturas genéricas se configurarán todos los edificios. Para esta parte del proceso se emplea exclusivamente Blender.

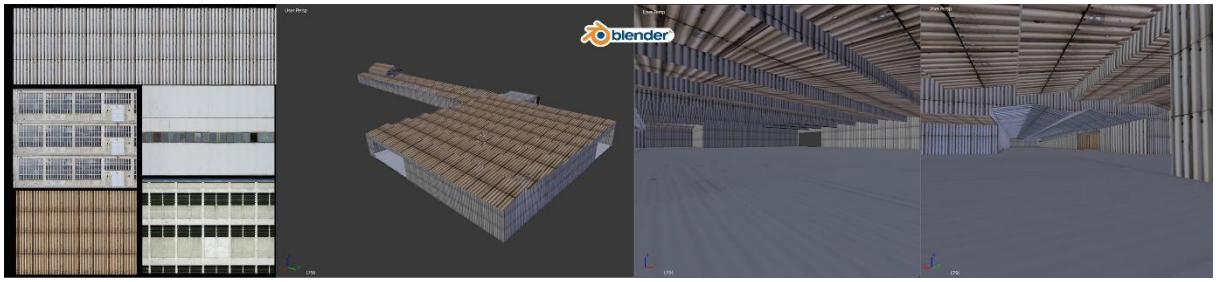

Figura 41. Asignación y configuración de los materiales para renderizado en tiempo real. De izquierda a derecha: texturas empleadas; vista exterior de los talleres de prefabricación 1 y prearmamento 1; y vistas interiores de los mismos. Fuente: elaboración propia.

iv. Modelado de los bloques del barco. Uno de los problemas encontrados en los archivos CAD proporcionados por la UMI fue la de la ausencia de caras duplicadas con aristas en direcciones contrarias. Esto implicaba inicialmente que en Unity, en función del ángulo de observación, ciertas caras estuviesen ocultas a la vista por la dirección de la normal. Por tanto, fue necesario procesar todos los bloques y subbloques con Blender, trabajo que se automatizó en la medida de lo posible. Tras varias pruebas, se corrigieron las normales de todos los bloques y de duplicaron las caras con las normales invertidas. N obstante, no se contempló la asignación de materiales en estos elementos, pues el tiempo de procesado hubiese sido demasiado costoso para la aportación que suponía al trabajo desarrollado. Una vez procesados todos los bloques, con la ayuda de personal de la UMI, se montó la fragata completa.

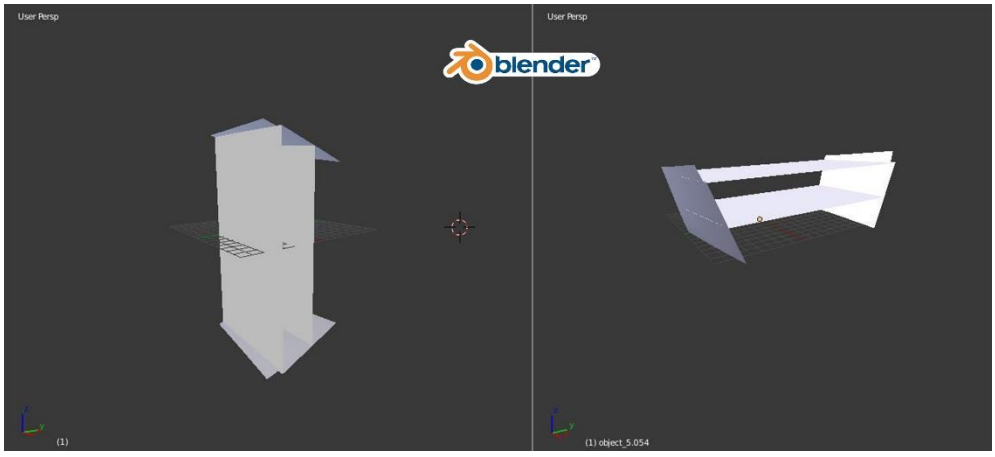

Figura 42. Ejemplo de modelado del bloque SB413. A la izquierda el archivo de origen, girado y con caras desagregadas. A la derecha, bloque modelado, en posición horizontal, con caras duplicadas y agregadas, y normales corregidas. Fuente: elaboración propia.

v. Configuración del resto de elementos de la escena. El resto de elementos que conforman la escena son la grúa cigüeña y los bloques que representan las estaciones de trabajo, ensamblaje y las colas. La primera se importa directamente en Unity pues el modelado es completo. Para los bloques de las estaciones, intentando asemejarse al modelo de FlexSim, se emplean unos recursos gratuitos descargados del “Asset Store” de Unity modificados para conseguir tal similitud.

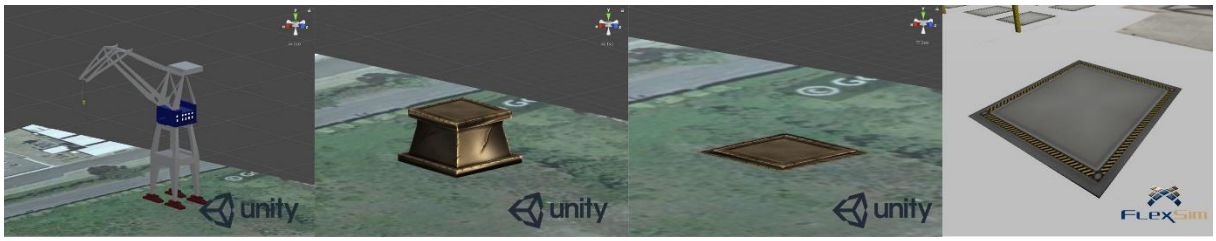

Figura 43. Resto de elementos de la escena. De izquierda a derecha: grúa cigüeña de Navantia –Ferrol importada en Unity; bloque descargado del "Asset Store"; bloque modificado; y comparación con estación de trabajo en FlexSim. Fuente: elaboración propia.

Cabe concluir que el trabajo gráfico realizado constituye una pequeña aproximación de las posibilidades que nos ofrecen estas herramientas. Si bien hará hincapié en el *Capítulo 5*, el alcance del trabajo desarrollado en esta parte se ha considerado válido teniendo en cuenta el tiempo que ocupa conseguir una mejora significativa en la calidad del resultado.

4.7. Generación de informes finales

Por último, como se ha mencionado previamente, una de las funcionalidades más interesantes de la simulación es la obtención de datos de la misma. En este sentido, es de sumo interés conocer el estado de las diferentes estaciones de trabajo, ensamblaje y colas al final de la ejecución. Junto con ello, sería interesante saber en qué momento los ítems entran o salen de cada estación, o cuantos días han estado a la espera en una cola en concreto. Estos datos nos ayudan a localizar cuellos de botella o funcionamientos incorrectos en nuestro modelo, y por tanto tomar decisiones a partir de estas observaciones. Por tanto, con respecto a todo ello, se implementan dos funcionalidades:

- Por un lado, se programan los ítems para que, una vez creados, vayan guardando los tiempos de entrada y salida en cada estación. De esta manera, a medida que avanza la simulación, cada vez que un subbloque llega al sumidero, se registra automáticamente en un archivo todos los tiempos de entrada y salida en cada elemento de la simulación. Este archivo está disponible para el usuario en la misma ruta que contiene el fichero ejecutable.

Id Subbloque	Source OUT	Queue1 OUT	Prefabricacion1 IN	Prefabricacion1 OUT	Prearmamento1 IN	Prearmamento1 OUT	Queue3 OUT	Prefabricacion2 IN	Prefabricacion2 OUT
SB107a1	0	0	0	0	0	2	2	2	2
SB107b1	2	2	2	2	2	2	2	2	2
SB111a1	9	9	9	35	35	60	60	60	60
SB111b1	11	11	11	48	48	60	60	60	60
SB401a1	14	14	14	47	47	74	74	74	74
SB401b1	15	15	15	42	42	74	74	74	74
SB403a1	23	23	23	51	51	102	102	102	102
SB403b1	24	24	24	63	63	102	102	102	102
SB405a1	38	42	42	77	77	107	107	107	107
SB405b1	39	47	47	86	86	107	107	107	107
SB103a1	32	32	32	83	83	155	155	155	158
SB103b1	33	35	35	66	66	155	155	155	158
SB113a1	120	120	120	198	198	203	203	203	203
SB113b1	122	122	122	155	155	203	203	203	203

Figura 44. Ejemplo de informe de los días de entrada y salida de cada subbloque en las diferentes estaciones. La figura muestra el contenido tal y como se muestra una vez copiado en el archivo Excel. Fuente: elaboración propia.

- Por otro, la opción "Generate Report" del menú final permite generar un archivo donde se registran todas las tasas de producción, ocupación y tiempos generales de procesamiento de las diferentes estaciones de trabajo de la simulación completada.

On7/6/2017 Prearmamento1:
Occupation Rate: 0.041
Production Rate: 0
Average Processing Time: 0
Transporte1: Nothing to show
Prearmamento2:
Occupation Rate: 0.09
Production Rate: 0.036
Average Processing Time: 1.333
Fragata Definitiva
Done: 7
Prearmamento2:
Occupation Rate: 0.036
Production Rate: 0.006
Average Processing Time: 4
Prefabricacion1:
Occupation Rate: 0.813
Production Rate: 0.042
Average Processing Time: 18.429
Prefabricacion1:
Occupation Rate: 0.94
Production Rate: 0.03
Average Processing Time: 31
Prefabricacion1:

Figura 45. Ejemplo de informe final de las diferentes estaciones. La figura muestra el contenido tal y como se muestra una vez copiado en el archivo Excel. Fuente: elaboración propia.

Como comentario, los informes generados son archivos de texto *.txt*. Solamente copiando el contenido en un archivo Excel, se mostrará de forma más clara para el usuario.

4.8. Modelo en realidad virtual

Una vez se valida el modelo de simulación de eventos discretos en 3 dimensiones del caso de demostración, se propone un nuevo objetivo a mayores: utilizar ese mismo entorno 3D para generar una experiencia mucho más inmersiva a través de la realidad virtual. Una de las motivaciones de dicho proyecto son las relativas facilidades que proporcionan los motores de videojuegos a la hora de generar *serious games* con estas características. Además, al no disponerse de unas gafas de realidad virtual, se decide emplear un visor y un teléfono móvil. Por otra parte, se ha tener en cuenta la ventaja que ofrece el poder construir una aplicación que sea ejecutable en un dispositivo móvil, independientemente de la experimentación con entornos de realidad virtual.

No obstante, este objetivo lleva asociado una serie de cambios que afectan a las diferentes partes del proyecto y que se exponen a continuación:

- El primer trabajo más inmediato fue el cambio del controlador de la cámara del usuario, así como la propia cámara. Para conseguir la realidad virtual, es necesario que exista un doble renderizado simultáneo en cada media parte de la pantalla del móvil *Figura 46*, de tal manera que sea la lente del visor la encargada de generar esa sensación de profundidad. En suma, el movimiento del usuario en el escenario no puede hacerse a través de un ratón o una pantalla, por lo que se implementa una función de “auto-caminar” que permite que, al inclinar ligeramente el móvil hacia delante, este se mueva en esa misma dirección en el mundo virtual. De esta manera se complementa la parte que concierne al control de la cámara en esta extensión del proyecto.

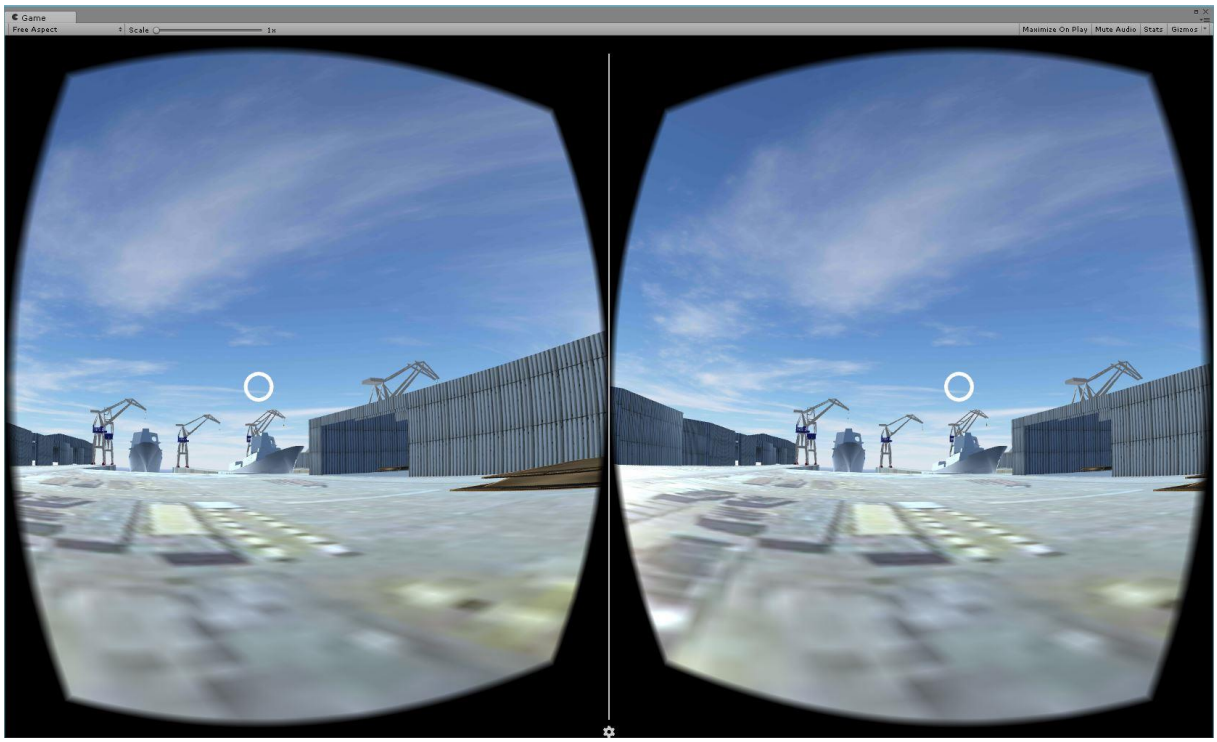

Figura 46. Vista general del escenario con doble renderizado. Fuente: elaboración propia

- Otro de los problemas que se tuvo que afrontar es la interacción con la interfaz. Como se ha indicado, no existe botón o pantalla a través del cual ejecutar los botones, sino que debe hacerlo el propio usuario con la vista. Esto, unido a las dificultades que suponían alguna de las funcionalidades de la interfaz desarrollada, obliga en primer lugar a simplificar esta, quedando tal y como se muestra en la *Figura 48*. Por otra parte, para resolver los problemas de interacción, se convierten todos los botones, menús desplegables y deslizadores en barras de progreso. Es decir, se le añade un puntero a la cámara del usuario situado justo en el centro de la pantalla de forma que, cuando el usuario posiciona dicho puntero en la barra de progreso, esta se empieza a completar de tal manera que, una vez cargada, se ejecuta dicha acción. Cabe comentar que la interfaz en este caso es un elemento más de la escena, lo que implica que su movimiento no va ligado al movimiento de la cámara del usuario. Esto es, es un panel situado en un punto central de la escena y permanece estático durante toda la experiencia. Cabe comentar que se eliminan todos los comandos rápidos que habían sido implementados.

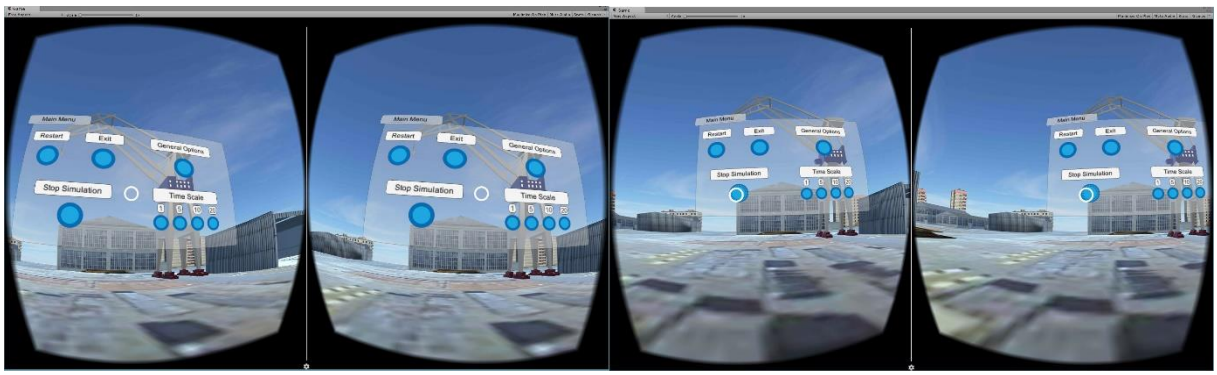

Figura 47. Composición de imágenes con doble renderizado de la cámara: a la derecha, vista general del menú; a la izquierda, interacción con el menú a través del cursor. Fuente: elaboración propia

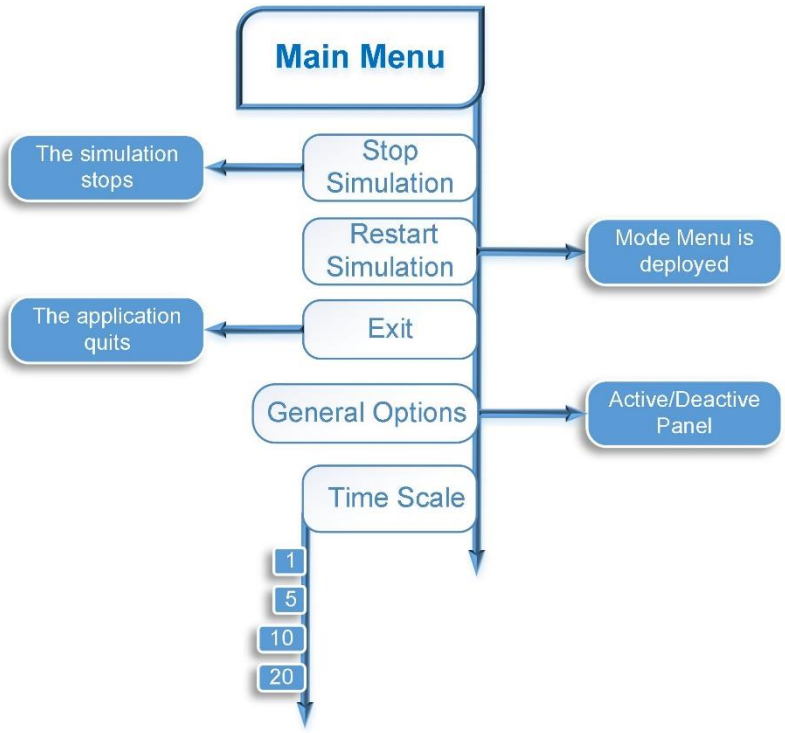

Figura 48. Interfaz desarrollada para realidad virtual. Fuente: elaboración propia

- Por último, unido al punto anterior, fue necesario modificar el código y depurarlo para eliminar todas aquellas funcionalidades que dependían en una u otra medida de la interfaz, lo que supuso un trabajo de “reprogramación”.

Capítulo 5

En este último capítulo se procede a realizar la comparativa entre las potencialidades que proporciona Unity como motor de una simulación de eventos discretos, empleando las librerías desarrolladas en este trabajo, frente al *software* de simulación de eventos discretos FlexSim, a través del caso de demostración expuesto.

Por último, se plantean asimismo una serie de futuros trabajos a acometer a partir del proyecto actual, describiendo en qué dirección existe una mayor capacidad de mejora y, por tanto, en la que se debe progresar.

5.1. Comparativa

Para realizar la comparativa entre ambas plataformas, se analizarán algunos de los puntos que ya se habían adelantado en *3.3.1 Caracterización* concretando para el caso de este trabajo. Es importante aclarar que nuestro objetivo va más allá de la parte gráfica que en dicho apartado se analizaba, acotando todas las posibilidades que nos permiten estos motores en lo referente a la desarrollo de un proyecto de estas características, desde el planteamiento del modelo conceptual hasta el análisis de los resultados.

- Alto rendimiento. Una de las primeras cuestiones que se comentaba es la alta capacidad de procesamiento que ofrecían los motores de videojuegos, cuya arquitectura está sumamente organizada para aligerar el desarrollo y la generación de aplicaciones. En este sentido, con respecto al caso de demostración, se ha verificado que no hacen falta grandes medios computacionales para conseguir proyectos con cierto nivel de detalle gráfico concluyendo que, con estos motores, podemos conseguir proyectos gráficamente impresionantes con los mismos medios que necesitaríamos para un software tipo FlexSim. Por otra parte, se llevó el motor sus límites límite a través de un modelo con un número de 10000 estaciones de trabajo en serie. El resultado de la prueba fue concluyente, demostrando que es capaz de ejecutar listas con miles de eventos y de renderizar, si bien con gran dependencia de la potencia del soporte empleado, una cantidad muy grande de recursos gráficos.
- Estructura de la aplicación. Quizás una de las diferencias más notables entre ambos softwares es el tipo de enfoque que poseen. FlexSim está orientado al desarrollo del proyecto dentro del propio programa. Sin embargo, la finalidad original de Unity es construir una aplicación, con lo que ello nos proporciona dos modos de funcionamiento cuando se emplea para la simulación: es posible, una vez volcado el modelo, validarlo y experimentar en la propia escena; o bien, construir la aplicación y experimentar con la escena en la medida en que se le permita al usuario según la interfaz desarrollada[8] Junto con ello, cuando lo que se hace es construir la aplicación, resulta lógico conformar una estructura de *serious game*. En el caso de demostración, se añadieron dos escenas a mayores, adecuándose a esa estructura y proporcionándole asimismo al usuario un mayor control sobre la propia aplicación.
- Versatilidad del proyecto desarrollado. Derivado de lo anterior, a diferencia de FlexSim, Unity permite crear un proyecto que puede ser ejecutado en la mayor parte de los dispositivos, ya sea un ordenador, *tablet*, *smartphone,* televisión o incluso red social, incluyendo diferentes sistemas operativos como IOS o Windows. Es decir, si se conforma el *serious game* con una interfaz lo suficientemente amplia como para experimentar con la simulación, se obtiene una aplicación realmente versátil y potente tanto para su uso en la optimización del proceso, como para aspectos relativos al *marketing*. En el caso de FlexSim, siempre es necesario tener un medio que soporte tal plataforma para poder ejecutar el proyecto.

[8] Por ejemplo, en el caso de demostración de este proyecto se implementan dos modos de funcionamiento. En uno de ellos, se le permite a este realizar una planificación diferente lo que, en un caso real, puede proporcionar resultados válidos en cuanto a optimización.

- Recursos gráficos. Es quizás el aspecto que los motores de videojuegos dominan con clara superioridad (*Figura 49*). Por una parte está la gran personalización que ofrecen, con los editores de materiales y de *shaders*, que permiten controlar aspectos hasta aspectos relativos al ángulo de reflexión de la luz o comportamientos de un material según la vista de una cámara determinada. Por otra, el renderizado de los motores está altamente optimizado, y la calidad del mundo virtual es muy alta. Es importante tener en cuenta que para conseguir grandes resultados es necesario manejar programas de modelado 3D, como los que se han empleado para este trabajo, conllevando un gran esfuerzo añadido. Pero, especialmente al principio, con pocos avances es posible dar un salto de calidad muy significativo. Por su parte, FlexSim provee una serie de elementos gráficos para las máquinas, permitiendo importar modelos 3D para adecuar la visualización a la realidad. La importación de imágenes también posibilita aportar la geolocalización de la escena, para una mayor correspondencia. No obstante, no está realmente enfocado a ello, por lo que las posibilidades no son mucho más amplias.

Figura 49. Comparativa de las vistas generales de los proyectos de FlexSim (izquierda) y Unity3D (derecha) del caso de demostración. Fuente: elaboración propia.

- Usabilidad. Otro aspecto que se adelantaba era el término usabilidad, que hacía referencia al sistema de control de la cámara. Sin lugar a dudas, es algo que coloca a Unity por encima de FlexSim en lo concerniente a la experiencia del usuario. Para el caso de demostración, se configuraron en concreto dos cámaras con las que el usuario puede experimentar, obteniendo distintos puntos de vista de la simulación y, en todo caso, configurando un entorno donde él mismo es el protagonista que se mueve por la escena. A diferencia de ello, en FlexSim no es posible personalizar el punto de vista, y conseguir experiencias como las desarrolladas en este proyecto. Es un software enfocado únicamente a la simulación, y ello implica que el movimiento de la cámara únicamente permite cambiar la rotación y aumentar o disminuir el zoom.
- Manejabilidad. Junto al punto anterior, se situaba la manejabilidad, esto es, la facilidad por parte del modelador a la hora desarrollar un proyecto. Lo que se ha comprobado es que, sin lugar a dudas, con las librerías necesarias, es relativamente sencillo implementar un modelo de simulación en un software como Unity. Junto a ello, muchas de las características que atañen al escenario y al proceso de desarrollo, como la posición y el tipo de luz, las propiedades de los materiales, la posición de los objetos, el propio *layout* de trabajo, o la calidad del renderizado en la fase de trabajo, son fácilmente modificables sin necesidad de programar absolutamente nada, contrariamente a lo que en un principio podría parecer. Sin embargo, aunque en el sitio web hay disponible una amplia documentación de la aplicación, es fundamental tener claro que el desarrollo de dichas librerías exige un gran esfuerzo, puesto que Unity no provee ninguna de las funcionalidades que ya vienen contempladas de serie en FlexSim. En contrapartida, a medida que el proceso simulado se hace más complejo, FlexSim también le exige al usuario cierto nivel de programación, nivelando en cierto sentido la balanza.

- Modelado. Lo que se analiza en este aspecto es la facilidad y rapidez con la que se implementa un modelo en Unity y FlexSim, pero desde un punto de vista cuantitativo; esto es, de tiempos. Para ello, se midió el tiempo que tarda un modelador de FlexSim en simular un proceso compuesto por 3 estaciones de trabajo y tres colas con tiempos aleatorios - además de la fuente y el sumidero - y el que se tarda en Unity. Tratándose de un proceso muy pequeño, en Unity se tardó un 40% más de tiempo en realizar el mismo modelo. Sin embargo, no es extrapolable a modelos más complejos donde sería necesario programar en ambas plataformas. Los resultados gráficos de los modelos se exponen en la *Figura 50*.

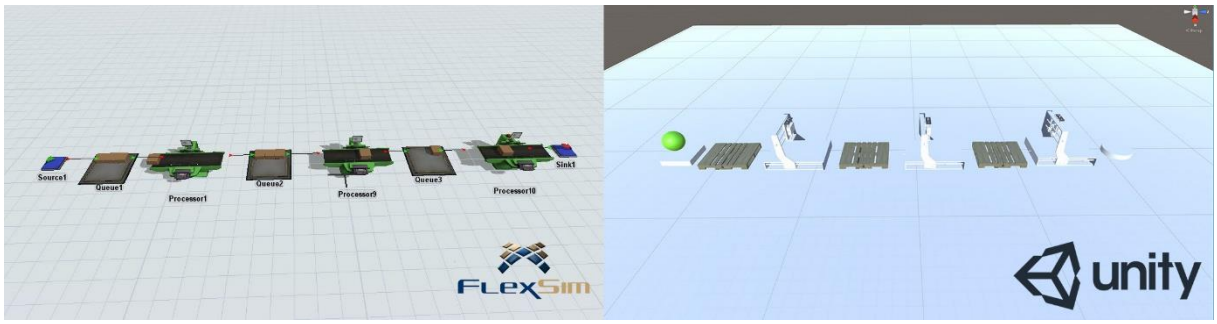

Figura 50. Resultados de la comparativa de modelado en FlexSim (izquierda) y Unity3D (derecha). Fuente: elaboración propia.

- Motor físico. A diferencia de FlexSim, Unity es un motor físico que se emplea en este caso para crear nuestro propio motor de eventos discretos. Esto implica que disponemos de un mundo mucho más amplio para crear ciertas acciones o eventos que posiblemente no serían ejecutables en FlexSim. Un ejemplo claro en este proyecto es el movimiento de la grúa, programado para realizar una serie de movimientos que componen el transporte del bloque. Podríamos incluso programar una serie de movimientos variables en función del ítem entrante. En suma Unity proporciona otra particularidad como es la de crear una visualización, que en ese caso si sería específica para el modelo en cuestión. Por otra parte, otro ejemplo interesante sería el de cierto evento de un proceso industrial que se ejecuta según una acción física, como el desplazamiento de paquetes sobre unos rodillos por acción de la gravedad. En un software convencional de simulación de eventos discretos no conocemos los tiempos de desplazamiento a menos que los midamos en planta. Pero en un motor de videojuegos si podemos llegar a modelar tal acción, al poseer un motor físico. Es decir, con todo lo anterior, es observable las posibilidades a mayores que nos brindan estos motores físicos frente a un motor de avance discreto.
- Reciclaje. Sin lugar a dudas, se ha comprobado que es una de las funcionalidades más ventajosas e interesantes de los motores de videojuegos. En el desarrollo del caso de demostración, alguno de los elementos, en especial parte de la interfaz y el entorno, fueron desarrollados previamente en otros proyectos, como se indicó en *4.2 Desarrollo del código*. Estos componentes fueron posteriormente reimportados al proyecto final, con los mismos scripts que gobernaban su comportamiento, lo que permitió avanzar parte del trabajo cuando aún no se había escogido el caso de demostración. Por otra parte, esta versatilidad permite intercambiar a través de Internet componentes desarrollados y modificar posteriormente el código que lo maneja para adaptarlo al proyecto, como fue el caso del mar implementado en el astillero, o el control de la cámara de primera persona. Por su parte, en FlexSim si existe la funcionalidad de crear objetos personalizados con un comportamiento determinado y reutilizarlos en otros proyectos. Sin embargo, Unity está orientado al desarrollo de *software*, por lo que esta característica está mucho más perfeccionada que en FlexSim, aportando más facilidades a la hora de desarrollar un modelo nuevo.
- Realidad virtual y aumentada. Fue uno de los objetivos añadidos en el tramo final del trabajo y, sin duda, da muestra de las posibilidades que están aún por investigar. En

ningún caso FlexSim permite desarrollar algo similar. Por su parte, tras la experiencia de este trabajo, se han comprobado las facilidades que proporcionan los motores como Unity para convertir un proyecto desarrollado en 3 dimensiones en un proyecto de realidad virtual, lo que significa crear un entorno realmente inmersivo que, combinado con la planificación de la producción, puede dar lugar a herramientas muy dinámicas.

- Accesibilidad económica. Finalmente, el aspecto económico supone un importante inconveniente en la adquisición de los *softwares* comerciales de simulación. Realmente, una de las ventajas más destacadas de los motores es indudablemente su precio. Una licencia completa de Unity3D no cuesta más de 2.000€, mientras que los *softwares* comerciales como FlexSim pueden costar del orden de 15.000€ por copia.

5.2. Conclusiones

En primer lugar, en este trabajo se han desarrollado las librerías necesarias para la implementación de un modelo de simulación genérico en Unity3D con el alcance definido previamente. Estas librerías permiten simular el comportamiento de una serie de elementos como la fuente, las colas, las estaciones de trabajo y ensamblaje, y el sumidero, pudiendo mostrar datos del modelo en tiempo real y generar una serie de informes finales de cada elemento. Asimismo, se incluyen los *scripts* necesarios para el desarrollo de una interfaz gráfica de usuario en Unity que le permita al mismo tener control de la simulación y la aplicación.

A continuación, se implementó el caso de demostración que consistió en el montaje de una fragata en grada, adaptando el código previamente desarrollado al mismo. Esto incluye además la generación de una interfaz usuario y la programación de un segundo modo de funcionamiento, en el cual el usuario puede planificar de forma interactiva la llegada de ítems al proceso dentro de la propia aplicación. Paralelamente a lo anterior, se desarrolló el entorno virtual haciendo uso de Blender, SketchUp y Photoshop, donde se modelaron y texturizaron todos los componentes del entorno. Una vez obtenido el modelo, se experimentó con entornos de realidad virtual.

Finalmente, se llevó a cabo una comparativa entre Unity3D y FlexSim, teniendo en cuenta una serie de aspectos que engloban la parte gráfica, el modelado, el rendimiento y la versatilidad de ambas plataformas. A partir de esta comparativa, fue posible obtener una serie de conclusiones generales que se exponen a continuación:

- Con respecto al rendimiento, queda contrastada la alta capacidad del código y Unity para manejar listas de cientos de eventos y renderizar múltiples texturas y elementos en la vista de la cámara sin necesidad de disponer de un equipo de gran potencia, o por lo menos, superior al necesario para FlexSim.
- A efectos de modelado y de soporte en otras plataformas, Unity es una herramienta mucho más dinámica que FlexSim, tanto por la posibilidad de crear aplicaciones y exportarlas a otros dispositivos, como por la manera en que el modelador lo utilice para la simulación y optimización de procesos.
- Gráficamente es indudable la superioridad de Unity. No obstante, el esfuerzo que supone crear la parte gráfica, una vez se alcanza cierto nivel, no añade valor para la simulación, aunque sí como herramienta de marketing. Por tanto, la solución es buscar una solución de compromiso en función del fin deseado, teniendo en cuenta la facilidad para el reciclaje que nos permite Unity.
- Al tratarse de un motor de videojuegos, Unity no tiene integradas herramientas específicas de desarrollo de modelos de simulación de eventos discretos como en el caso de FlexSim. Esto implica que es necesario programar la totalidad de las librerías, superponiendo el motor de eventos discretos al motor físico de Unity. Por tanto, si no se dispone de estas librerías, la viabilidad de Unity como herramienta modeladora de

procesos industriales se complica, al exigir demasiado esfuerzo previo para completar un proyecto.
- En caso de disponerse de tales librerías, si podría resultar realmente competente desarrollar un modelo en una plataforma como Unity, a efectos de tiempo de modelado frente a resultado.
- FlexSim no posee un motor físico, con lo que las posibilidades que a mayores permite Unity podrían marcar la diferencia si se diese el caso de tener que desarrollar un proyecto con suficiente detalle como para necesitarlo.
- Por el momento, es posible concluir que la realidad virtual es únicamente una herramienta de marketing, pues no aporta ventajas desde un punto de vista analítico.

Por último, cabe comentar que no necesariamente en todos los usos de la simulación se requiere un entorno inmersivo. Sin embargo, bien es cierto que puede acelerar la optimización del proceso así como la explicación de un modelo a personal no técnico, mejorando el flujo de la comunicación en una empresa. De hecho, el propio FlexSim sigue esta tendencia pudiendo ya implementar entornos de realidad virtual. En suma, con modos de funcionamiento que permitan planificar interactivamente, puede ayudar a tener un mayor control del proceso o a la emisión de órdenes de trabajo, donde si existe ya una conexión directa con la realidad aumentada.

Sin lugar a dudas, el trabajo realizado en este proyecto se engloba enteramente dentro del marco de la Industria 4.0, que no es más que el futuro de la industrial actual. Por tanto, es posible afirmar que la simulación y optimización de procesos puede aprovechar los avances de la industria del videojuego para proporcionar nuevas funcionalidades y mejorar las actuales.

5.3. Trabajos futuros

En último lugar, a la vista del trabajo realizado, se plantean una serie de ideas que definen la dirección que debería tener una ampliación del mismo.

- Un primer trabajo futuro sería el de ampliar las librerías desarrolladas de forma que alcancen una funcionalidad equivalente al *software* comercial de eventos discretos.
- Por otra parte, un aspecto no desarrollado en este trabajo que podría resultar de interés ampliar en el futuro sería el de desarrollar un módulo específico de gestión de escenarios para facilitar la experimentación en el modelo, de forma análoga al *software* de simulación de eventos discretos.
- En cuanto a la parte gráfica, se considera que no es necesario una mayor calidad en lo concerniente a la simulación. Sin embargo, en caso de que sí exista interés de mejorar el proyecto actual, el siguiente paso es obtener las texturas de los edificios y adecuar las proporciones de los mismos. Además, una opción relativamente sencilla en este sentido es mejorar el entorno no interactivo.
- Por último, en este caso de demostración, podría ser de interés programar los movimientos de los transportes intermedios entre talleres, lo cual resultaría relativamente asequible, al tener desarrollado el movimiento de la grúa.

Bibliografía

Banks, J., II, J. C., & Barry, L. (2005). *Discrete-event system simulationfourth edition*. Retrieved from http://www.academia.edu/download/35744346/discrete-event_system_simulation_by_jerry_banks.pdf

Bijl, J. L., & Boer, C. A. (2011). Advanced 3D visualization for simulation using game technology. *Proceedings of the Winter Simulation Conference*. Winter Simulation Conference.

Brettel, M., Friederichsen, N., & Keller, M. (2014). How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective. *Of Mechanical, Industrial …*. Retrieved from http://www.waset.org/publications/9997144

Castrilllón, J. V. (2008). LA SIMULACIÓN DE PROCESOS, CLAVE EN LA TOMA DE DECISIONES. *DYNA*, *83*(4), 221–227.

Cowan, B., & Kapralos, B. (2014). A Survey of Frameworks and Game Engines for Serious Game Development. In *2014 IEEE 14th International Conference on Advanced Learning Technologies* (pp. 662–664). IEEE. https://doi.org/10.1109/ICALT.2014.194

Crespo Pereira, D. (2013). Modelos de series temporales para simulación de procesos industriales : aplicación al dimensionamiento y control de sistemas altamente variables. Retrieved from http://ruc.udc.es/dspace/handle/2183/11511

Friese, K.-I., Herrlich, M., & Wolter, F.-E. (n.d.). Using Game Engines for Visualization in Scientific Applications. In *New Frontiers for Entertainment Computing* (pp. 11–22). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-09701-5_2

Guasch, A., Piera, M. A., Casanovas, J. (Casanovas G., & Figueras, J. (Figueras J. (2003). *Modelado y simulación aplicación a procesos logísticos de fabricación y servicios*. Edicions UPC. Retrieved from https://books.google.es/books?hl=es&lr=&id=5kJpBgAAQBAJ&oi=fnd&pg=PP1&dq=simulación+de+procesos+eventos+discretos&ots=dG-k6wcbSK&sig=V8hKNBpXd2lydd8dQk57SIc7k8w#v=onepage&q&f=false

Herrlich, M. (2007). A Tool for Landscape Architecture Based on Computer Game Technology. In *17th International Conference on Artificial Reality and Telexistence (ICAT 2007)* (pp. 264–268). IEEE. https://doi.org/10.1109/ICAT.2007.25

Herwig, A., & Paar, P. (2002). Game engines: tools for landscape visualization and planning. *Trends in GIS and Virtualization in*. Retrieved from https://www.researchgate.net/profile/Philip_Paar/publication/268212905_Game_Engines_Tools_for_Landscape_Visualization_and_Planning/links/5464ab2b0cf2cb7e9dab8bc5.pdf

Indraprastha, A., & Shinozaki, M. (2009). The Investigation on Using Unity3D Game Engine in Urban Design Study. *ITB Journal of Information and Communication Technology*, *3*(1), 1–18. https://doi.org/10.5614/itbj.ict.2009.3.1.1

Lasi Hans-Georg Kemper, H., Peter Fettke, P., Thomas Feld, D.-I., & Michael Hoffmann, D.-H. (2014). Industry 4.0. https://doi.org/10.1007/

Lindskog, E., Vallhagen, J., & Johansson, B. (2017). Production system redesign using realistic visualisation. *International Journal of Production Research*, *55*(3), 858–869. https://doi.org/10.1080/00207543.2016.1218085

Martínez, F. (2011). Presente y Futuro de la Tecnología de la Realidad Virtual. *Creatividad Y Sociedad*. Retrieved from http://www.creatividadysociedad.com/articulos/16/4-Realidad Virtual.pdf

Mishra, P., & Shrawankar, U. (2016). Comparison between Famous Game Engines and Eminent Games. *International Journal of Interactive Multimedia and Artificial Intelligence*, *4*(1), 69. https://doi.org/10.9781/ijimai.2016.4113

Peralta Abarca, J. del C. (2011). Simulación de procesos a través de eventos discretos. *Inventio, La Génesis de La Cultura Universitaria En Morelos, Nº. 15, 2011-2012, Págs. 71-74*, (15), 71–74.

Rohrer, M. (2000). Seeing is believing: the importance of visualization in manufacturing simulation. *Simulation Conference, 2000. Proceedings.* Retrieved from http://ieeexplore.ieee.org/abstract/document/899087/

Rüßmann, M., Lorenz, M., Gerbert, P., & Waldner, M. (2015). Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston Consulting*. Retrieved from http://www.inovasyon.org/pdf/bcg.perspectives_Industry.4.0_2015.pdf

Wenzel, S., & Jessen, U. (2001). The Integration of 3-D Visualization into the Simulation-based Planning Process of Logistics Systems. Retrieved from http://journals.sagepub.com/doi/pdf/10.1177/003754970107700304