

A decorative header at the top of the page featuring a complex, repeating geometric pattern of interconnected lines forming various polygons in shades of gray.

Producción de videojuegos serios

VÍCTOR MANUEL LARIOS ROSILLO (coord.)

Producción de videojuegos serios



Centro
Universitario
de Ciencias Económico
Administrativas

Universidad
de Guadalajara



Itzcóatl Tonatiuh Bravo Padilla
Rectoría General

Miguel Ángel Navarro Navarro
Vicerrectoría Ejecutiva

José Alfredo Peña Ramos
Secretaría General

José Alberto Castellanos Gutiérrez
Rectoría del Centro Universitario
de Ciencias Económico Administrativas

José Antonio Ibarra Cervantes
Corporativo de Empresas Universitarias

Sayri Karp Mitastein
Dirección de la Editorial Universitaria

Primera edición, 2015

Coordinador
Victor Manuel Larios Rosillo

Textos

© Víctor Manuel Larios Rosillo, Óscar Fernando
González Delgadillo, Martha Elena Zavala Villa,
César Alejandro García García, Laura Torres
López, Martha Patricia Martínez Vargas

Diagramación y cuidado tipográfico

Virginia Ramírez Moreno
Gustavo Alonso Ortega López

Corrección

Sandra Elizabeth Hernández Zamora



Este trabajo está autorizado bajo la licencia Creative Commons
Atribución-NoComercialSinDerivadas 4.0 Internacional (CC
BY-NC-ND) lo que significa que el texto puede ser compartido
y redistribuido, siempre que el crédito sea otorgado al autor,
pero no puede ser mezclado, transformado, construir sobre él ni
utilizado con propósitos comerciales. Para más detalles
consúltese [https://creativecommons.org/licenses/by-nc-nd/4.0/
deed.es](https://creativecommons.org/licenses/by-nc-nd/4.0/deed.es)

Todos los derechos de autor y conexos de este libro, así como de cualquiera de sus contenidos, se encuentran reservados y pertenecen a la Universidad de Guadalajara. Por lo que se prohíbe la reproducción, el registro o la transmisión parcial o total de esta obra por cualquier sistema de recuperación de información, sea mecánico, fotográfico, electrónico, magnético, electroóptico, por fotocopia o cualquier otro, existente o por existir, sin el permiso por escrito del titular de los derechos correspondientes.

Queda prohibido cualquier uso, reproducción, extracción, recopilación, procesamiento, transformación y/o explotación, sea total o parcial, sea en el pasado, en el presente o en el futuro, con fines de entrenamiento de cualquier clase de inteligencia artificial, minería de datos y texto y, en general, cualquier fin de desarrollo o comercialización de sistemas, herramientas o tecnologías de inteligencia artificial, incluyendo pero no limitando a la generación de obras derivadas o contenidos basados total o parcialmente en este libro y/o en alguna de sus partes. Cualquier acto de los aquí descritos o cualquier otro similar, está sujeto a la celebración de una licencia. Realizar alguna de esas conductas sin autorización puede resultar en el ejercicio de acciones jurídicas.

Producción de videojuegos serios / Víctor
Manuel Larios Rosillo, coord. ; Óscar
Fernando González Delgadillo... [et al.] ;
prólogo María del Rocío Maciel Arellano,
prólogo -- 1a ed. -- Guadalajara, Jalisco
: Editorial Universitaria : Universidad de
Guadalajara, Centro Universitario de Ciencias
Económico Administrativas, 2015.
184 p. : il. ; 23 cm. -- (Colección
Monografías de la Academia).
Bibliografía: p.171-177.

ISBN 978 607 742 191 7

1. Juegos de computadora-Diseño 2. Juegos de
computadora-Programación 3. Videojuegos-
Diseños Videojuegos-Industria I. Larios
Rosillo, Víctor Manuel, coord. II. González
Delgadillo, Óscar Fernando III. Maciel
Arellano, María del Rocío IV. Serie

794.81 .P96 CDD
QA76.76 .C672 .P96 LC

D.R. © 2015, Universidad de Guadalajara



Editorial Universitaria
José Bonifacio Andrada 2679
Colonia Lomas de Guevara
44657 Guadalajara, Jalisco

01 800 834 54276
www.editorial.udg.mx

ISBN 978 607 742 191 7

Mayo de 2015

Impreso y hecho en México
Printed and made in Mexico

Índice

7	Agradecimientos	
9	Prólogo	MARÍA DEL ROCÍO MACIEL ARELLANO
11	Introducción	VÍCTOR MANUEL LARIOS ROSILLO
15	Capítulo 1. Producción de videojuegos serios	VÍCTOR MANUEL LARIOS ROSILLO
40	Capítulo 2. Elementos de la cadena de producción	VÍCTOR MANUEL LARIOS ROSILLO ÓSCAR FERNANDO GONZÁLEZ DELGADILLO
94	Capítulo 3. Integración de multitudes de personajes artificiales a los videojuegos	MARTHA ELENA ZAVALA VILLA VÍCTOR MANUEL LARIOS ROSILLO
110	Capítulo 4. Motor de inteligencia artificial	CÉSAR A. GARCÍA GARCÍA
120	Capítulo 5. Integración de bases de datos en videojuegos	LAURA TORRES LÓPEZ
135	Capítulo 6. Estrategias de comunicación en red para videojuegos	MARTHA PATRICIA MARTÍNEZ VARGAS VÍCTOR MANUEL LARIOS ROSILLO
168	Conclusiones generales	
171	Bibliografía	

Agradecimientos

Agradecemos a todos los que hicieron posible la realización de esta obra; por sus inquietudes, actitudes y valiosas aportaciones. Lo dedicamos especialmente a aquellos que fueron pacientes en el largo proceso de desarrollo y creyeron en nuestro equipo, y también a los que conocen el tema, participaron en él y desean que este libro perdure, en un fascinante viaje de conocimiento entre tecnología y creatividad, con la posibilidad de aplicarlo para el bien de la sociedad y la productividad empresarial tanto a nivel regional, nacional e internacional.

Es importante señalar que este trabajo no sería posible sin el apoyo de los fondos establecidos para proyectos de investigación COETCYTJAL-UDG 2009, que permitieron generar un laboratorio de producción de videojuegos serios e impartir una escuela de invierno en 2010 con personalidades de talla mundial. Esta obra es el colofón de la experiencia adquirida en dicho proyecto.

Asimismo, damos nuestros agradecimientos al Programa Nacional de Posgrados de Calidad (PNPC) del Consejo Nacional de Ciencia y Tecnología (Conacyt), ya que fue posible con el reconocimiento y acreditación que este programa federal proporcionó al Doctorado en Tecnologías de Información de la Universidad de Guadalajara, mediante las becas de tiempo completo, que el equipo de estudiantes becarios de doctorado pudieran participar con varios capítulos para que esta obra tuviera un componente de visión vanguardista, proporcionada por temas de investigación que representan verdaderos retos y oportunidades con contribución a nivel internacional.

Agradecemos además a la industria, particularmente a la empresa Intel de México, que desde el inicio respaldó este laboratorio dando acceso a su clúster de procesamiento paralelo para los autores de este libro, y al maestro Raymundo Vázquez Lugo, quien ha promocionado incansablemente la vinculación de la academia e industria con sus opiniones sobre las mejores prácticas a realizar.

Resta decir que esta obra no tendría el alcance esperado a nivel internacional, ni toda la retroalimentación de experiencia, sin el apoyo, siempre continuo y durante más de una década, de la Universidad de Toulouse y el equipo Vortex del centro de investigaciones IRIT (Institut de la Recherche Informatique de Toulouse). Esta institución nos ha guiado y respaldado en la codirección de tesis doctorales, así como en la organización de talleres y seminarios orientados a los

videojuegos serios y las problemáticas de investigación asociadas. Damos un reconocimiento muy especial en esta obra a Yves Duttein y Hervé Luga, quienes año con año nos brindaron consejos, observaciones y retroalimentaciones que de alguna forma resultaron en experiencias invaluable que se materializan en el contenido de este ejemplar.

Prólogo

MARÍA DEL ROCÍO MACIEL ARELLANO

Las tecnologías de información están omnipresentes en nuestra vida diaria. Gracias a sus plataformas, nuestras actividades laborales, personales y hasta las tareas educativas, se vuelven cada vez más eficientes.

Particularmente, el caso de los videojuegos es una muestra de ello, pues inicialmente se desarrollaron como un área lúdica; sin embargo, actualmente tienen muchos otros usos que pueden mejorar nuestra calidad de vida y productividad. Como ejemplo, a nivel de la industria, los juegos serios ofrecen enormes posibilidades para capacitación mediante la simulación de situaciones que, de otra manera, serían muy costosas de reproducir. En nuestra vida personal nos pueden ayudar a cambiar hábitos o a mejorar nuestra salud mediante rutinas, además de que nos pueden respaldar en procesos educativos.

Es por ello que, al tener los videojuegos un gran impacto social para la mejora de productividad y calidad de vida, se volvió de gran importancia la realización de un trabajo como el que tiene entre sus manos; un trabajo que facilitara entender su desarrollo, así como conocer cuáles son las opciones para hacerlo con herramientas de *software* libre. El interés radica en que si bien esta industria de videojuegos siempre estuvo tachada de ser muy costosa en las inversiones de equipamiento, instalaciones y *software* para su producción, hoy en día dicha producción se ha vuelto accesible a través del uso de *software* libre y con equipo de cómputo convencional.

De esta manera, lo que se pretende con esta obra es alentar a emprendedores y desarrolladores a aplicar su creatividad para la realización de videojuegos serios. Cabe resaltar que su creación es resultado de cuatro años de trabajo desarrollando videojuegos serios por parte del doctor Víctor M. Larios y su equipo de investigadores.

Desde la perspectiva del Doctorado en Tecnologías de Información, el presente trabajo muestra, inicialmente, la organización de la cadena de producción y, posteriormente, refleja cuáles son las áreas de oportunidad para desarrollo de videojuegos con innovación y en la frontera con lo más avanzado de las tecnologías de información. Cuatro direcciones de investigación son mostradas,

mismas que corresponden a tesis doctorales, las cuales han sido ya sustentadas y cuentan con resultados de impacto a nivel internacional. La primera se da con la generación de miles de personajes para poblar un videojuego con una serie de rasgos similares mediante cómputo evolutivo y algoritmos genéticos. La segunda corresponde a conectar a todas las entidades a generar con un cerebro único que permita simular multitudes; en videojuegos, esto permite generar simulaciones de alto impacto social en temas de seguridad y protección civil. En la tercera línea se presenta la importancia de las bases de datos para los videojuegos y cómo estas pueden evolucionar para generar videojuegos que se adapten a la integración de nuevas fuentes de información y de gestión, tales como las redes sociales. Finalmente, en cuarto lugar se encuentra el manejo de las telecomunicaciones y una presentación de estrategias para optimizar los procesos de comunicación en la red, con juegos, donde una gran cantidad de usuarios comparten la simulación.

Por todo lo anterior, para el Doctorado en Tecnologías de Información, en mi calidad de coordinadora, se volvió una empresa importante el poder divulgar estos trabajos de manera accesible al público en general, y así poder motivar a otras comunidades de emprendedores, investigadores, artistas, sociólogos y de otras áreas a acceder a estas tecnologías y encaminar proyectos en esta dirección.

Asimismo, se vuelve relevante que al ser Guadalajara la sede nacional para el desarrollo de la industria creativa digital y su ambicioso proyecto, sea esta publicación una contribución de la Universidad de Guadalajara para que esta industria se pueda detonar y que la limitación de recursos y acceso a la tecnología no sea un factor restrictivo para emprendedores; que esto promueva proyectos no solamente lúdicos, sino también en el sentido de los videojuegos serios con aplicaciones de alto impacto social, de mejora de productividad y competitividad en la región, y un medio más para alcanzar una mayor calidad de vida en la zona metropolitana.

En la actualidad, el doctor Víctor M. Larios y su equipo de investigadores dirigen sus esfuerzos en el tema de *Smart Cities* a través del Centro de Innovación en Ciudades Inteligentes del CUCEA UDG en apoyo con redes de investigación internacionales con respaldo de la triple hélice de industria, academia y gobierno.

Es indudable que el planteamiento de estas cadenas de producción de videojuegos avalan proyectos de este centro de innovación y que varias tesis de doctorado siguen apoyando y dando continuidad al macroproyecto con una dimensión de alto impacto social como es el aplicar los videojuegos mediante sus simulaciones para resolver los problemas de las ciudades.

Introducción

VÍCTOR MANUEL LARIOS ROSILLO

Como un esfuerzo de la Universidad de Guadalajara para contribuir en el desarrollo de una economía basada en el conocimiento, este libro promueve opciones para desarrollar el sector creativo digital en el nicho estratégico de los videojuegos serios. Se busca, por lo tanto, que sin grandes recursos e infraestructura, cualquier persona interesada en el tema pueda identificar mediante esta obra los elementos para producción de videojuegos serios, estimar mercados con los que pueda generar sus planes de negocio y determinar qué tecnologías, principalmente abiertas o de muy bajo costo, se pueden integrar para emprender. En cuarenta años de evolución de los videojuegos, los denominados “videojuegos serios” ofrecen la posibilidad de impactar en respaldar la generación de una industria local más competitiva, así como contribuir fuertemente en la mejora de procesos educativos. En definitiva, la relevancia de esta obra consiste en propulsar y dotar de las mejores prácticas al talento y contribuir en consolidar el ecosistema de la citada Ciudad Creativa Digital en la zona metropolitana de Guadalajara.

Si bien gran parte de los videojuegos se desarrollan en ambientes de dos dimensiones (2D), en esta obra se considera que la visión de los problemas y soluciones en ambientes 3D dan mayor cobertura (2D es un subconjunto de la solución) al concepto que se plantea, que son los videojuegos serios. De tal manera, el presente libro busca integrar una estructura amplia en relación con las diferentes tecnologías de esta disciplina, explicando paso a paso los procesos y herramientas que se estiman accesibles y en su mayoría bajo licencia de *software* libre. Además de lo anterior, también se incluyen trabajos clasificados como “de frontera” para mantener esta obra vigente, es decir, una investigación sobre algo desconocido que aporta conocimiento nuevo.

El objetivo principal es, entonces, dar a conocer los componentes y elementos más importantes en la producción y desarrollo de los videojuegos en general, pero con un énfasis especial en la integración y la programación de la interacción en ambientes tridimensionales (3D), primordialmente de los videojuegos serios, los cuales, más que tener una intención lúdica, tienen propósitos educativos.

Cabe mencionar que el desarrollo de un videojuego, sea cual sea su clasificación, significa un enorme reto, ya que implica una gran cantidad de áreas de conocimiento que se requieren integrar para constituir un producto, por lo que es indispensable desarrollar fuertes habilidades de trabajo multidisciplinario en equipo.

Ahora bien, entre los integrantes o equipos que deben tenerse en cuenta para que un videojuego pueda ser exitoso, se encuentran los escritores, los artistas y los programadores. Los primeros se encargan de crear un guión de historia coherente. Por su parte, los segundos crean a los personajes con escenarios, las animaciones y los efectos especiales. Finalmente, los programadores son aquellos que, mediante sus códigos, hacen que los personajes y sus animaciones cobren vida en función del guión dentro de las diferentes plataformas tecnológicas en que tenga lugar su ejecución. En este sentido, lo anterior significa que los videojuegos tienen como reto la necesidad de sumar varios talentos reunidos en un equipo multidisciplinario y, en consecuencia, con una alta coordinación e integración.

Aunado a esto, hay que tener presente que la producción generalmente tiene presupuestos y tiempos restringidos, por lo que la gestión particular de este tipo de proyectos requiere de una alta capacidad de creatividad y adaptación tecnológica. Es por ello que las grandes universidades con esta vocación de todo el mundo han encontrado que las carreras dedicadas al desarrollo de videojuegos forman profesionistas muy completos, con habilidades deseables para todo el ecosistema de las áreas de las tecnologías de la información, la informática, la computación y las telecomunicaciones. En pocas palabras, forman estudiantes que asimilan los conocimientos e integran soluciones a los problemas y retos que plantean los videojuegos, gracias a la ayuda que ofrecen las diferentes tecnologías.¹

En lo que respecta a la estructura de este libro, su división consta de una introducción, seis capítulos de desarrollo y conclusiones generales. A continuación se detalla cada uno de ellos.

El capítulo uno presenta una introducción sobre el contexto tecnológico de los videojuegos serios y establece definiciones de conceptos y características propias; se hace énfasis en las áreas de oportunidad y mercados potenciales donde estos tienen lugar; y por último, dado que el producto final entra en las economías basadas en el conocimiento, se señalan algunos detalles importantes en términos de propiedad intelectual.

En el siguiente capítulo se plantean los elementos necesarios para generar una cadena de producción; se expone la organización requerida, el flujo de trabajo y las herramientas indispensables para ambientes tridimensionales; y se

¹ Si se quiere información al respecto se puede consultar en la IGDA (Asociación Internacional de Desarrolladores de Videojuegos), donde un grupo de empresas y universidades se asociaron para definir los elementos que debe contener un plan de estudios en esta área y que puede explicar el por qué se ha vuelto tan atractivo este tipo de formación profesional (IGDA, 2010).

presenta una forma de construir, partiendo del guión, las escenas, los personajes, las animaciones y los efectos especiales audiovisuales. Por la complejidad de las herramientas requeridas, se podría pensar que el proceso de producción puede ser costoso, de manera que aquí se incorpora una selección de *software* con licencia libre y sin costo, con todas las funciones necesarias en la cadena de producción. Asimismo se abordan los aspectos artísticos para la creación de los escenarios, los personajes, sus animaciones y su integración con el motor 3D (componente necesario que crea el ambiente y ejecuta el videojuego) elegido. En las diferentes secciones se sugieren recomendaciones para optimizar los resultados con las herramientas seleccionadas, y así tener una cadena de producción eficiente. Finalmente se discuten algunos problemas que podrían encontrarse en este aspecto, y se detallan las herramientas con costo que los resolverían.

A partir del capítulo tres se incluye una colección de trabajos de investigación de frontera, con la finalidad de dar una mayor visión de algunos procesos de la cadena de producción en donde hay áreas de oportunidad. En este caso se expone el problema de crear multitudes de personajes para videojuegos serios, representando mundos complejos con cientos o miles de personajes autónomos. Igualmente se muestra una solución para esto, resultado de varios años de investigación y desarrollo, que no solo puede ser útil para videojuegos serios, sino también para producciones cinematográficas con animaciones digitales que requieran de estas técnicas de inclusión de multitudes de personajes.

El capítulo cuatro se centra en lo que se denomina el “motor de inteligencia artificial” y en las técnicas necesarias para que este anime a los personajes del juego. Se ilustran los diferentes procedimientos empleados para que los programas que gobiernan a los personajes de un juego tengan comportamientos más inteligentes; se relaciona con el de la generación de multitudes y ayuda a procesar su comportamiento con la discusión de algunas técnicas de programación paralela y distribuida. Cabe mencionar que este capítulo es el resultado de varios años de trabajo desarrollado en conjunto con un grupo de investigadores de la Universidad de Toulouse en Francia.

Enseguida, en el capítulo cinco se plantea la demanda de almacenamiento de información y muestra una forma de solucionarla apoyada en las bases de datos. Para una casa de producción de videojuegos serios, este es todo un tema que permite integrar funciones para almacenar los eventos y ayudar a que cuando se tienen varios participantes (sean autónomos o gobernados por la computadora con el motor de inteligencia artificial) el ambiente presente una coherencia.

Por último, el capítulo seis trata de cómo lograr las comunicaciones para que se pueda contar con un ambiente con múltiples usuarios y toma en cuenta los extremos de ambientes donde participan miles simultáneamente. De la misma manera, se presentan las técnicas más comunes y una propuesta propia de cómo conseguir esto con base en la experiencia. Las redes, los protocolos de comunica-

ción, las estrategias y trabajos de optimización son un tema de vigencia con retos y áreas de oportunidad que complementa el desarrollo de videojuegos serios.

Finalmente, el cierre del libro corresponde a las conclusiones y perspectivas que se tienen en esta área, de la cual hay muchas oportunidades de desarrollo y que abre un mundo de posibilidades muy interesantes para la realización de proyectos asociados a videojuegos serios. Permitiendo que quienes apuesten a este tipo de desarrollos se mantengan en una economía basada en el conocimiento a través de la tecnología y una preparación para el desarrollo de trabajo altamente multidisciplinario.

CAPÍTULO 1

Producción de videojuegos serios

VÍCTOR MANUEL LARIOS ROSILLO

Introducción

Aun con los avances tecnológicos que existen actualmente, la industria de los videojuegos sigue teniendo cambios vertiginosos, además de que implica costos significativos en su producción. En el caso de los juegos con sentido lúdico, se puede decir que aquellos que cuentan con mayores niveles en efectos, trabajo artístico e interactividad, tienen costos de realización similares o mayores a los de las producciones cinematográficas. De la misma manera, para llevarse a cabo su creación son necesarios equipos de 200 a 300 personas trabajando contra el reloj, ya que disponen de tiempos de entrega muy cortos. No obstante, si bien el resultado final es de gran calidad, al igual que las grandes películas, los videojuegos guardan una corta vida una vez que los consumidores los tienen en sus manos. Es por ello que, para mantenerse en el mercado, las casas de producción requieren estar sacando constantemente productos.

Un claro ejemplo que ilustra lo que puede llegar a costar un videojuego es el de *Halo 3*, estrenado en 2007 (García, 2012). Logrado bajo un equipo de producción compuesto por 120 personas a tiempo completo durante 16 meses (italo (series), s.f.; Huntemann y Aslinger, 2012; Zackariasson y Wilson, 2012), en la campaña de lanzamiento se mencionó que el título había alcanzado un presupuesto de producción similar a las películas cinematográficas más caras de la época y, en consecuencia, que se estimaban ganancias significativas. El costo aproximado fue de 60 millones de dólares (mdd), pero tan solo en el primer día de su estreno generó ganancias por 170 mdd (Montes, 2007). Posteriormente, en 2008 Microsoft anunció que había vendido cerca de 8.1 millones de copias del juego (Sinclair, 2008), lo que se refleja en unos 486 mdd, considerando que cada licencia tenía un costo de 60 dólares (Schiesel, 2009). Asimismo, este videojuego fue parte de la campaña de lanzamiento de la nueva consola de videojuegos Xbox 360, que en su

momento ofrecía capacidades mejoradas de gráficos en tiempo real con relación a su antecesor (Rubio, 2012). Cabe decir que la combinación del juego con la consola tuvo como resultado un fuerte impacto, así como una gran aceptación, derivado de grandes inversiones en innovación y tecnología de los grandes corporativos que respaldaron estos desarrollos.

Ahora bien, como ya se mencionó previamente, existe el mercado lúdico de los videojuegos, el cual se vuelve cada vez más complejo y despierta mayores exigencias entre los usuarios de los mismos; sin embargo, también hay aplicaciones con otros fines, como los educativos y los de simulación, conocidos como *videojuegos serios*. En esta clasificación se abre un mundo de oportunidades todavía con equipos de producción pequeños, pero apoyados con los avances actuales. Es importante aclarar que para este tipo de aplicaciones de los videojuegos se requiere principalmente de creatividad; no es necesario tanto realismo. Incluso algunos de estos no requieren más que una interfaz con animaciones en 2D, en razón de que se pondera más un buen guión y desarrollo de la narrativa, además de dinámica y lógica en ellos (Michael y Chens, 2005). Los detalles de las simulaciones en estas aplicaciones, que se definirán enseguida, no siempre dependen completamente de los gráficos, sino de la información que presentan y la interacción que se da.

La necesidad de equipos multidisciplinarios de producción

En todas las áreas de aplicación en que la informática está presente, se requiere de todo un equipo multidisciplinario de producción, sobre todo en la que incluye a los videojuegos. La figura 1.1 muestra, a grosso modo, las diferentes disciplinas que hay que considerar para formar los equipos de producción de videojuegos.

Como se puede ver en dicha figura, para comenzar el proceso lo primero por hacer es determinar qué es lo que se busca con el juego y cuál es el objetivo, todo ello sustentado por una buena historia. El equipo del área de ciencias sociales tiene que interactuar de inmediato con un equipo de artistas y diseñadores para generar la escena (lo más apegada al guión e historia definida), así como los personajes, los objetos que serán parte de las escenas, las animaciones y las formas de interacción, junto con la música y efectos sonoros que ambientan y dan vida a la historia. Por su parte, hablar de la interacción y la representación de los objetos de la escena, particularmente de los personajes y su control, implica que todo este equipo está en trabajo conjunto con programadores, e incluso en algunos casos puede haber matemáticos y físicos para definir las dinámicas de los objetos mediante modelos que permitan generar un realismo al ser programados. En ese sentido, este equipo de programación realiza la integración de todo y cuida que la

aplicación resultante del videojuego cumpla técnicamente con lo necesario para una ejecución segura, no pesada y altamente interactiva en la plataforma de juego final, que puede ser una computadora o una consola especializada.

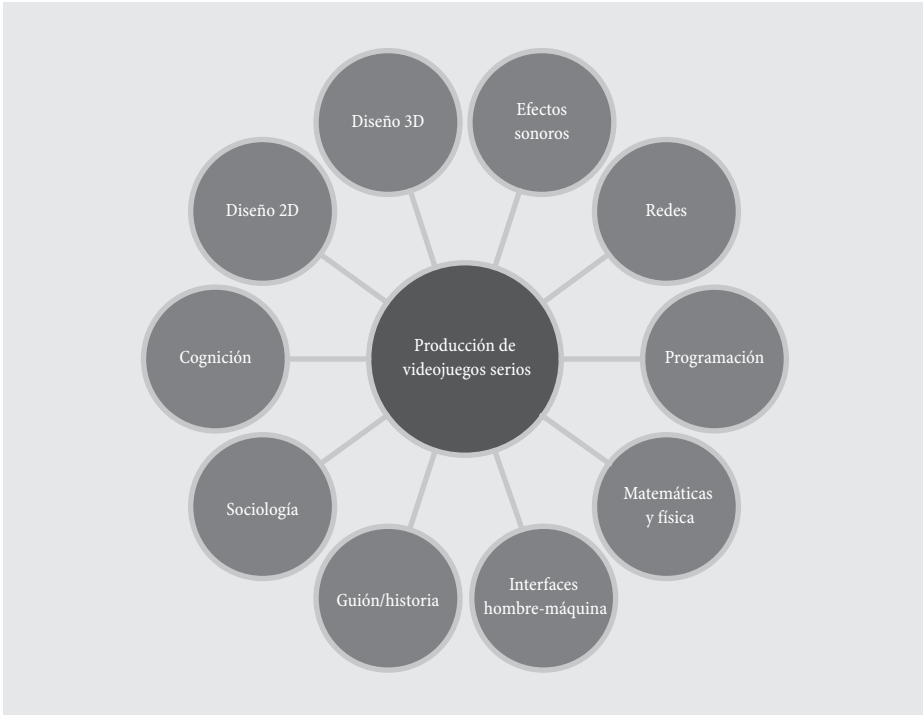


Figura 1.1. Las diferentes grandes áreas de conocimiento comprendidas en el desarrollo de un videojuego

Técnicamente, los juegos pueden ejecutarse en una sola consola o computadora, aunque a partir de la última década aumentó notablemente la inclinación por contar con una conectividad de red que permitiera a múltiples jugadores competir entre ellos en el videojuego, por lo que el área de redes también está presente en el desarrollo. Esto puede subsanar la carencia de programas de inteligencia artificial que a nivel de los adversarios no representen mayor reto.

Finalmente, la parte correspondiente a la disciplina de la interfaz hombre-máquina es la que se encarga de generar una evaluación para revisar que los objetivos, los requerimientos y especificaciones iniciales se cumplan. Es hasta ese momento que se puede liberar el videojuego en el mercado.

Mención aparte merece el tema del gasto de campaña mercadológica, el cual es casi comparable o mayor al de la producción del mismo videojuego. Este no se incluye en la figura anterior por ser, más que parte de la producción, parte de la estrategia de ventas.

Los equipos de producción se conformarán de acuerdo con la complejidad del videojuego. Así, para proyectos sencillos podrá haber equipos desde una a tres personas, y para aquellos superproyectos, se contará con departamentos especializados por área y subáreas, sumando de decenas a cientos de personas, como en las grandes casas de producción de videojuegos.

Las tecnologías y tendencias actuales

Hoy en día, para poder desarrollar una cadena de producción de videojuegos no es necesario tener un equipo como el de las grandes casas productoras. Gracias a los avances tecnológicos en procesamiento gráfico, actualmente existen mejores programas de *software* que simplifican tareas gráficas y facilitan la depuración de código.

Como resultado de 30 años de evolución de los videojuegos (Kaplan, 2013; Wolf, 2008), con equipos pequeños ya se pueden hacer cosas muy interesantes. A continuación se especifica cómo se logra esto.

Los avances tecnológicos cada año permiten hacer más con menos. Esto se relaciona con la ley de Moore (Cheang Wong, 2005), que menciona que cada 18 meses se doblan las capacidades de procesamiento de los equipos y se alcanzan resultados similares en otras áreas, como en almacenamiento y resolución de las pantallas, entre otros aspectos. Dichos avances facilitan que los proyectos que antes se realizaban con grandes equipos y costosos laboratorios, ahora se produzcan con resultados similares o mejores empleando pocos recursos, e incluso con miembros del equipo que pueden trabajar desde sitios geográficos remotos operando a través de una red. Lo anterior se debe, por una parte, a los desarrollos que siguen teniendo las computadoras personales en cuanto a capacidades de procesamiento, de mayor memoria y de dispositivos especializados, como el caso de las tarjetas dedicadas a la representación de los gráficos cada vez más eficientes y rápidas. Por otra parte, también está el desarrollo de las tecnologías de la información, que permiten, tanto en instituciones como a nivel personal, disponer de enlaces de red de datos más veloces, con mayor capacidad y omnipresentes —gracias a los sistemas inalámbricos—, lo que facilita el trabajo en equipo sin barreras geográficas.

La estandarización de librerías de desarrollo y portabilidad de código a diferentes plataformas. Con más de 40 años de evolución de los gráficos por computadora (inició en los setenta con Sutherland) (Ryan, 2011), la industria y la academia han consolidado algunos estándares para la programación, facilitando el desarrollo de aplicaciones cada vez más sofisticadas, y reutilizable en las librerías de su código.

Grandes redes de distribución de videojuegos y aplicaciones en línea. En la década pasada —2000 a 2010—, como una consecuencia de la conectividad de los usuarios y plataformas a la red de internet, un elemento que potenció el desarrollo de los videojuegos fue el hecho de que los fabricantes más importantes de plataformas de *hardware* (para consolas dedicadas, computadoras personales y hasta dispositivos móviles) crearon sistemas de distribución y venta de juegos sobre internet, accesibles a todos los propietarios de los dispositivos de dicho *hardware*. Este gran paso permitió que los desarrolladores pudieran ofertar sus juegos y alcanzar mercados que rentabilizaran sus costos de desarrollo, liberándolos de gastos de distribución y venta en línea, debido a que todo se realiza sobre la red, además de que permite que las aplicaciones de juegos desarrolladas por equipos pequeños y con poco presupuesto, pero con gran creatividad y originalidad, puedan ser conocidas y tener oportunidad de crecer. Igualmente, esto conlleva un beneficio para los clientes, pues cuentan con una gran oferta de aplicaciones disponibles para sus equipos, con plataformas más atractivas y funcionales.

Impacto de las tecnologías multinúcleo o *multicore*

Como ya se mencionó, los videojuegos serios como una disciplina de trabajo e investigación son el producto de 30 años de desarrollo de los gráficos por computadora y de los mismos videojuegos, tal como se muestra en la figura 1.2. Cabe mencionar que durante este trayecto ha habido periodos donde las ganancias generadas por el mercado de los videojuegos han sido muy significativas; por ejemplo, en el año 2005 la empresa Nintendo¹ reportó como ganancia a nivel mundial el equivalente a lo que el gobierno de Estados Unidos destinó de su producto interno bruto para la educación de ese año (Zackariasson y Wilson, 2012; Vogel, 2010; O'Donell, 2011).

En suma, esto ha creado un sector muy especializado de la industria, el cual se involucra tanto en el desarrollo de *hardware* más especializado y robusto para la generación de gráficos como en el manejo de los procesos paralelos para la ejecución de videojuegos altamente interactivos y con gran cantidad de detalles de gran realismo en los ambientes que generan.

¹ Empresa fabricante de consolas para videojuegos. Uno de sus más recientes desarrollos es la consola Nintendo Wii, que fue una revolución en interacción. Dicha consola reemplaza los controles comunes de videojuegos por bastones que miden la posición en el espacio real en 3D, permitiendo que los jugadores experimenten físicamente formas más realistas de interacción con sus propios movimientos y gestos corporales. El lanzamiento de esta consola en 2005 fue la causa del reposicionamiento de esta empresa como líder en el mercado mundial de consolas de videojuegos a la fecha, a pesar de que no tiene gráficos tan potentes como sus competidores. El componente de interacción corporal que no existía a ese nivel de desarrollo abrió mercados nuevos, incluyendo a la gente adulta.



Figura 1.2. Treinta años de evolución de los videojuegos serios y diferentes elementos que han permitido su desarrollo

A decir verdad, por el precio que se paga por una consola de videojuegos (alrededor de \$500 dólares) se obtiene un aparato con una capacidad de procesamiento de cómputo que supera por mucho cualquier computadora personal que se adquiera por ese costo. En algunos casos, como se dio con el lanzamiento de la consola de Sony, la PlayStation 3, la arquitectura resultó más avanzada que cualquier computadora personal que se tuviera en el momento, debido a que cuando el nivel de computadoras personales se iniciaba con los sistemas a dos núcleos de procesamiento,² esta consola ya contaba con ocho núcleos³ más los

² La mayoría de las computadoras personales basadas en arquitecturas Intel o AMD contaban con problemas de consumo de energía y disipación de calor al aumentar las frecuencias de operación de los procesadores, pero los fabricantes descubrieron que al poner dos procesadores o más en un solo chip lograban mayor rendimiento en procesamiento por menos energía, lo que generó una revolución y un nuevo camino en esta industria, marcando el inicio de los procesadores *core duo* o con dos núcleos y previendo que poco a poco las computadoras personales incluirían mayor número de núcleos de procesamiento. En este sentido, lo anterior ha comprometido a los programadores de aplicaciones a cambiar el paradigma de programación secuencial por el de programación paralela, ya que de lo contrario no se aprovechan los múltiples núcleos de que se dispone.

³ El procesador Power6 de IBM —conocido como Cell BE— cuenta con ocho procesadores de propósito particular, unidos por un bus de alta velocidad compartiendo una memoria común.

recursos de procesamiento de la tarjeta gráfica. Hay que referir que este procesador es el mismo que la empresa IBM emplea montado en tarjetas de servidores para construir máquinas de supercómputo especializadas para procesar grandes cantidades de operaciones de punto flotante por segundo (Buttari, Dongarra y Kurzak, 2007; Playstation 3 cluster, s.f.). Entre las características de versatilidad de esta consola destaca que en ella es posible generar una segunda partición en el disco duro para poder implantar un sistema operativo Linux y así poder cargarle programas y usarla como nodo de procesamiento, aunque hay que señalar que la cantidad de memoria que se dispone para los procesadores es limitada, ya que la mayor parte de esta está destinada a la tarjeta gráfica, y solo es posible tener acceso a ella como consola de videojuego. Un dato significativo es que por un costo muy económico (\$2,000 dólares aproximadamente) se puede armar un arreglo de tres o cuatro PlayStations 3 en clúster,⁴ equivalente a 48 procesadores interconectados.

Desde años atrás, finales e inicios del año 2000, ya se empleaba la tendencia de usar múltiples núcleos de programación en las tarjetas gráficas para mejorar el desempeño de las capacidades de procesamiento. Dichas tarjetas con procesadores especializados, ante la gran necesidad de poder generar escenas gráficas, de forma independiente, de gran complejidad y realismo, traducido en cálculos acelerados de operaciones entre matrices en una, dos o tres dimensiones, fueron uno de los orígenes con mayor relevancia a nivel de las computadoras personales en esta evolución hacia las arquitecturas *multicore* o multinúcleo.

Asimismo, con esas tarjetas se buscó desarrollar en la unidad de procesamiento gráfico (GPU) librerías de *software* para poner ese poder de procesamiento disponible para otro tipo de cálculo diferente de las operaciones gráficas. Fue así como nació la librería de CUDA propuesta por el fabricante de tarjetas NVIDIA. A la fecha, esta tecnología ha avanzado tanto que en una sola tarjeta gráfica se pueden tener hasta 250 núcleos de procesamiento (cifra que va en aumento), y una computadora personal podría tener hasta cuatro de estas tarjetas cargadas, lo que permite realizar procesamiento de datos de gran complejidad, como es el cómputo de aplicaciones científicas en una computadora personal, compitiendo directamente con las soluciones de supercómputo.

En esta última década, con relación al avance tecnológico en el *hardware* y con las empresas especializadas en la producción y desarrollo de los videojuegos que han fabricado componentes de *software* que aprovechan al máximo las capacidades de gráficas y de procesamiento disponibles para videojuegos más realistas, se ha creado un gran mercado de consumo y una cultura global vinculados a las

Aunado a esto, la consola tiene la tarjeta gráfica, un disco duro de varios cientos de gigabytes y una tarjeta de red con velocidad de Gigabit Ethernet.

⁴ Un clúster es un arreglo de computadoras interconectadas que se configuran como si fueran una sola máquina por medio de *software* para poder sacar provecho de todos los procesadores para aplicaciones de supercómputo.

consolas de videojuegos, los cuales ya son parte de la vida cotidiana de muchos hogares en el mundo. Vale la pena señalar que esta cultura global es generadora de grupos que están a la expectativa de nuevas consolas con mejoras, así como de nuevos videojuegos que planteen retos, formando un mercado mundial de millones de clientes.

Por lo tanto, se puede concluir que en los videojuegos serios, tanto el estado de la tecnología actual y sus capacidades de procesamiento, como el de los usuarios, se encuentran en un estado de madurez muy avanzado, dando lugar a la necesidad y aceptación de aplicaciones de videojuegos no lúdicas. Ahora bien, más allá de estos elementos tecnológicos, a continuación se presentan los factores que están influyendo en el desarrollo de los videojuegos serios.

¿Qué son los videojuegos serios?

Los videojuegos serios, a diferencia de los lúdicos, en general están relacionados con una experiencia de aprendizaje y entre sus características se pueden identificar tres aspectos, que son: transferencia de conocimiento (educar), transferencia de habilidades (entrenar) y transferencia de actitud (informar).

A saber, los primeros videojuegos serios más conocidos son los simuladores de vuelo, donde los pilotos se entrenan y aprenden cómo controlar los aviones ante diferentes circunstancias. Algo muy valioso de esto es que los errores cometidos en el simulador no tienen el costo de una situación en la vida real. Si se reflexiona un poco sobre los requisitos de este tipo de videojuegos, se considera que en el caso anterior se cumplen todos. Un simulador de vuelo debe transferir conocimiento para *educar* a los pilotos acerca de cómo manipular el avión en circunstancias simuladas de fallas o mal tiempo. Por su parte, también debe *entrenarlos* para que ante circunstancias repetidas desarrollen habilidades. Finalmente, les *informa* cómo están evaluados según sus acciones, con el propósito de que puedan mejorar o saber que están actuando correctamente.

Una definición de lo que es un videojuego serio es esta:

Videojuego serio = diversión + una meta (aprender, comunicar, simular, estudiar) + interactividad + una serie de reglas + un ambiente (2D o 3D) + pretensión de una situación.

Así pues, en relación con el primer elemento, un juego tiene que ser divertido en mayor o menor cantidad para que resulte interesante como actividad. Si bien en los videojuegos serios la diversión no es completamente el objetivo, como ocurre con los lúdicos, sí es un elemento que debe estar ahí. Las metas o

los objetivos, por su parte, no siempre están presentes; existen videojuegos que buscan el análisis de un problema mediante la creación de diferentes escenarios de simulación en función de distintas tomas de decisiones. A este respecto, una referencia ilustrativa es la del videojuego *Peace Maker*,⁵ que plantea el problema entre dos naciones en guerra, y donde, de acuerdo con las decisiones que se van tomando, se puede lograr la paz o empeorar las cosas.

Interrelación de disciplinas

El tema de la interrelación de las disciplinas para la creación de un videojuego serio y su influencia durante su desarrollo es algo complejo; sin embargo, se puede decir que lo anterior procede de varias vertientes complementarias que tienen principalmente cuatro grandes terrenos: la investigación, el mercado o la audiencia para la cual se crean, la industria y sus necesidades y, finalmente, la Web 2.0, que se vuelve semántica y que está orientada a darle un mayor valor a la información y a las redes sociales. Estos cuatro elementos (véase figura 1.3) se corresponden fuertemente, y todos en conjunto aportan al desarrollo de las tecnologías y teorías para videojuegos serios.

Como se muestra en la figura 1.3, la academia es la que propicia la investigación y la que ha dado lugar a la nueva modalidad de la Web 2.0, pero también, en un sentido más pragmático, tanto los usuarios finales como la industria se han respaldado en las nuevas interfaces hombre-máquina que proponen una interacción más efectiva en los ambientes. Esto significa que no solo a nivel de interfaces de los usuarios, sino también con estudios sobre cómo los usuarios perciben e interactúan con los videojuegos serios, es que se pueden mejorar para que sean más efectivos. En definitiva, los videojuegos como tales, en un sentido lúdico, han aportado bastante para el desarrollo actual de los videojuegos serios y toda la experiencia que se tiene es de gran valor.

⁵ Una traducción del título sería *El pacificador*. Para más información, véase <http://www.peacemakergame.com>

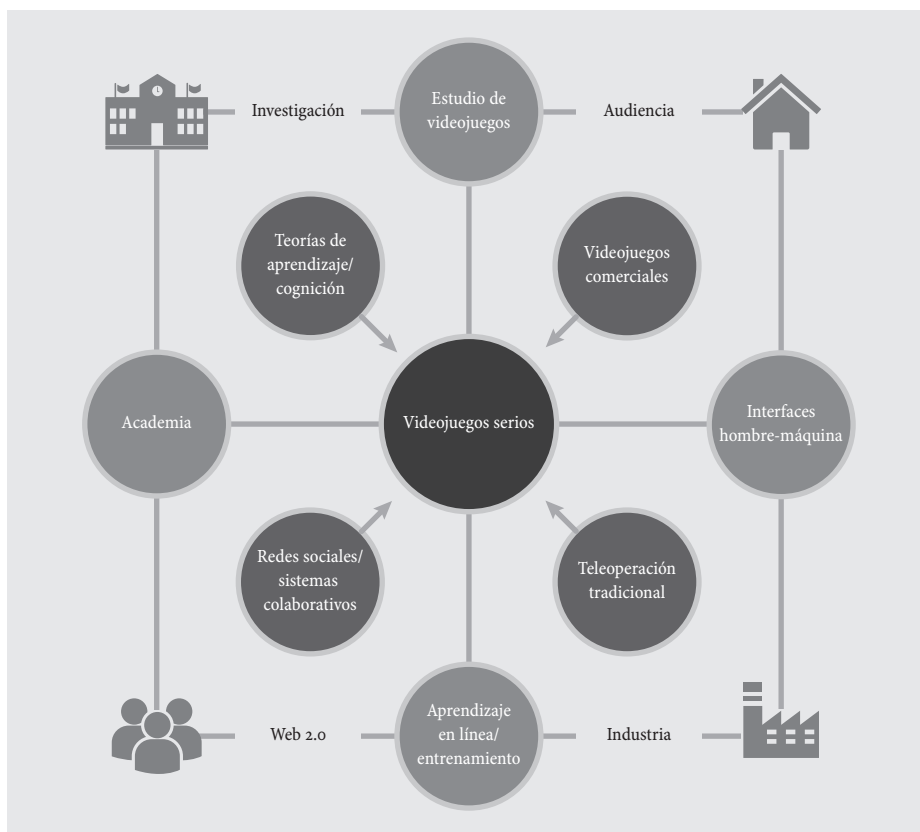


Figura 1.3. Factores y actores que influyen e impulsan el desarrollo de los videojuegos serios

Características y ventajas de los videojuegos serios

En general, los videojuegos ofrecen muchas ventajas como medio de aprendizaje, de entrenamiento y de información, ya que están soportados por varias disciplinas. Entre las principales características que debe tener un videojuego se encuentran las siguientes:

- *Actividad voluntaria.* Los videojuegos, en principio, son elegidos por el jugador, y es este quien decide cómo y de qué manera interactuar con el mismo.
- *Pretención de una situación.* Dependiendo del objetivo del videojuego, se puede tener una narrativa que envuelva al jugador en una situación separada de la real, especialmente creada para que llegue a su meta. Esto permite mezclar escenarios reales y con acciones irreales en nuestro mundo, tales como volar o mover objetos pesados.

- *Inmersión.* Esta característica no necesariamente depende de la perspectiva de una realidad virtual con sensores y dispositivos interactivos sofisticados que engañen los sentidos. En gran parte de los videojuegos, es la narrativa de este la que logra crear un estado de abstracción tal en el jugador, que, para fines prácticos, se dirá que está en un estado de inmersión. De aquí la importancia de que un videojuego cuente con un buen guión.
- *Límites de tiempo y lugar.* En todos los videojuegos, sean en 2D o 3D, se define un ambiente o lugar donde el jugador debe iniciar, además de las zonas o secciones dentro de las cuales el jugador puede moverse. De esta forma, así como se delimita el espacio físico, también se delimita un momento para cada acción y un tiempo finito de duración en dicha escena, que puede estar regido por eventos resultantes de la interacción generada en el lugar.
- *Basado en reglas.* Todo videojuego tiene una serie de reglas con las que funciona y mediante las cuales el jugador debe interactuar para lograr su meta u objetivo en el ambiente creado. Las reglas deben ser claras para el jugador y se trabajan desde la concepción del mismo. Lo anterior es fundamental, puesto que con base en estas reglas se retroalimenta al jugador su progreso y desempeño.
- *Crea un grupo social.* Los videojuegos generan grupos de personas que se identifican con los mismos y que, generalmente, en comunidad comunican sus experiencias sobre cómo lograr un mejor desempeño en el juego o llegar a una meta determinada en este.

Una vez detalladas las características, el siguiente paso corresponde a la reflexión de las ventajas que implican los videojuegos. Al generar una situación donde se puede representar y estudiar un problema, se ayuda a producir pensamiento crítico y analítico. Esto lleva a que los jugadores asuman roles, enfrenten diferentes problemáticas, planteen estrategias para la solución del problema, tomen decisiones y tengan una retroalimentación de estas sin el costo que tendrían en la vida real.

En este proceso, además, los jugadores aprenden una serie de reglas para jugar, las cuales se pueden desarrollar para cualquier profesión que se plantee o actividad de utilidad en una sociedad. Adicionalmente, se desarrollan y mejoran habilidades motrices de coordinación entre ojos y manos al reaccionar a los ambientes e interactuar en ellos. Al momento de navegar en los espacios de los videojuegos, se despliega una percepción de los espacios en 2D y 3D que permite ubicarse en mapas y asimilar nociones de distancia. Ya en los ambientes, se desarrolla la memoria sobre formas u objetos del juego asociadas con las metas.

Por último, resta decir que los videojuegos son potenciales precursores de la curiosidad, es decir, los jugadores aprenden probando, dado que el costo implica solamente repetir el punto donde se está jugando para llegar a una meta; los ju-

gadores no dudan en buscar nueva información que los pueda llevar más rápido a cubrir su objetivo y alcanzar dicha meta propuesta.

Mercados y sectores de oportunidad aplicativa

Entre las opciones de mercado con que cuentan los videojuegos serios se puede encontrar un área emergente, donde actualmente estos poseen una gran demanda con soluciones probadas. Estos sectores son los siguientes:

- Aplicaciones militares
- Gobierno
- Educación
- Salud
- Aplicaciones corporativas
- Político (intervención)
- Religioso
- Artístico
- Mercadeo

El primer punto, es decir el de las aplicaciones militares, tiene que ver con que parte de los videojuegos se ha concentrado en el desarrollo de simuladores de guerra, estrategias y manipulación de equipos y dispositivos. Esta ha sido una forma efectiva, sin costo de vidas humanas, de reclutar, enseñar y entrenar a los efectivos de las fuerzas armadas. Un videojuego con este contenido es el de *America's Army*.⁶

Para el caso del gobierno, los videojuegos se emplean para buscar mejoras en el manejo de varias circunstancias, como son el entrenamiento en situaciones de gran estrés, como el combate de incendios o ataques de terroristas; el manejo de situaciones de crisis, como negociar con terroristas; y finalmente, para enfrentar desastres naturales o infecciones contagiosas en una población.

Por su parte, los videojuegos enfocados a la educación mantienen un equilibrio entre lo divertido y lo didáctico. Asimismo, mediante ellos se miden los resultados y progreso de los estudiantes. Un ejemplo muy interesante de este tipo es el de *At-risk*,⁷ ya que en él se monitorea la motivación de estudiantes adolescentes y se prevee que no caigan en una depresión o frustración que pueda tener consecuencias negativas.

⁶ Véase <http://www.americasarmy.com>

⁷ Véase <http://www.kognito.com/atrisk/trailer.html>

Los videojuegos desarrollados para el sector salud tienen como características principales buscar la distracción de los pacientes para que no sientan dolor ante intervenciones dolorosas; ofrecer simulaciones para mejorar la rehabilitación del paciente; utilizarse para intervenciones terapéuticas, como la cura de fobias; y mejorar habilidades psicomotrices, entre otras.

Respecto a las aplicaciones corporativas, hay que señalar que las empresas tienen grandes ahorros gracias a los videojuegos, debido a que a través de cursos a distancia, al no requerir espacios o equipo dedicado para trabajar en la empresa, se puede conseguir el mismo resultado. Lo único que se necesita es el videojuego y el equipo de cómputo del trabajador que va a entrenar. Este tipo de práctica también ha probado ser efectiva en el desarrollo de habilidades multitarea o de aprendizaje y entrenamiento para la operación de equipos industriales complejos.

El resto de los mercados —político, religioso, artístico y de mercadeo— busca, más que entrenar o educar, informar para lograr un cambio de actitud o desarrollar una habilidad; en el caso de las artes, por ejemplo, se aprende por la repetición y experimentación sucesiva.

En definitiva, se puede concluir entonces que este tipo de juegos han roto las barreras étnicas y culturales, en donde hombres, mujeres y niños de todas las edades pueden hacer uso de estos. Con la omnipresencia, cada vez más intensa, de las computadoras en nuestra vida cotidiana, los videojuegos son una parte inseparable de nuestras culturas. Es el momento de sacar provecho de ellos en nuestro contexto local.

Propiedad intelectual y derechos de autor

La gran cantidad de contenido digital que está a nuestro alcance es un aliado para facilitar el trabajo creativo en un videojuego; sin embargo, es muy importante tener cuidado con no infringir los derechos de autor de un contenido. Estos derechos son respaldados por la ley de cada país, donde se reconoce al autor de una obra o idea, ya sea literaria o musical, entre otras, como dueño de esta, por lo que todo uso de su idea requiere que se paguen regalías. En este sentido, lo anterior permite que los creadores puedan contar con la oportunidad de obtener un beneficio comercial por un tiempo limitado (LFDA, 2014).

Uso de material con derechos de autor en el desarrollo

La situación más común de uso de material con derechos de autor en lo que corresponde la creación de videojuegos, es la descarga de pistas de audio o de imá-

genes con licencia para su integración; en caso de hacerse sin el consentimiento del autor y ser detectado, puede llevar a un juicio legal muy costoso (Abelson, Ledeen y Lewis, 2008). Por ello, es importante revisar el tipo de licencia que pueda tener una imagen o pista musical, y si esta es de carácter para uso personal o si se puede redistribuir con un producto.

En este contexto, las licencias de tipo creativo común (en inglés *creative commons*) funcionan muy bien para estos fines. Con ellas se puede obtener un permiso en línea para una idea o contenido digital que uno desee promover en internet, posibilitando mantener algunos derechos, pero buscando que dicho contenido sea usado por otros creadores, y a la vez ellos lo usen para construir otros contenidos, contribuyendo así al desarrollo de las sociedades y las tecnologías. Estas licencias se pueden generar en el sitio de Creative Commons,⁸ en el cual actualmente ya hay varios tipos de licencias legalmente reconocidas a nivel mundial, según lo que requiera el autor. Es común entonces encontrar repositorios de objetos 3D, documentos plantilla, fotografías o texturas y pistas de audio bajo este tipo de licencia que puede permitir la construcción de un videojuego.

Tipos de protección de ideas, invenciones y productos

En total existen cuatro formas de proteger las ideas y los desarrollos de alguien. Estas son las patentes, los derechos de autor, las marcas y los secretos industriales (Laplante y Costello, 2005).

- *Las patentes.* Protegen, de forma general, el derecho de hacer uso de invenciones. Los inventores pueden resguardar procesos, dispositivos y técnicas que deben ser útiles y novedosos. También se pueden obtener patentes de diseños relacionados con los ornamentos y estética de un artículo manufacturado.
- *Los derechos de autor.* Amparan el trabajo de un autor, en el sentido de que se puedan desplegar, copiar o mostrar. Para poder tener estos derechos, la obra debe ser original y declarada por un medio tangible.
- *Las marcas.* Son generalmente palabras protegidas, símbolos, diseños o combinaciones que las empresas utilizan para identificar sus productos y servicios o para diferenciarse de sus competidores. El derecho de registro como marca requiere de que se pruebe la descripción del producto o servicio que se identifica; debe servir para reconocer y discernir los productos de una empresa con los de sus competidores.
- *Los secretos industriales.* Son técnicas, procesos, habilidades, métodos u otra forma de conocimiento que dan a una empresa una ventaja competitiva en su mercado. Dicho conocimiento no debe ser divulgado fuera de la empresa.

⁸ <http://creativecommons.org>

Solo participan los socios comerciales que hayan firmado el documento legal de protección. En términos de los videojuegos, es importante considerar cuál será el esquema de protección intelectual a usar. Normalmente se busca que la empresa sea la dueña de la propiedad intelectual y que todos los empleados que participen en el proceso de desarrollo del juego firmen el acuerdo, de lo contrario, si un empleado sale de la empresa podría exigir regalías de un producto por autoría con un juicio legal, e incluso hasta un vendedor.

De esta manera, según la conveniencia de la empresa, se pueden generar licencias perpetuas o por tiempo limitado para el uso de los derechos de la propiedad intelectual. Además, podrán ser exclusivas o no exclusivas, y variar.

Cómo proteger y registrar

En México, el encargado de proteger y registrar los derechos de un autor es el Instituto Mexicano de la Propiedad Industrial (IMPI).⁹ Así, para patentar una invención es necesario hacer el registro y someterla a una aplicación que pasará a revisión para identificarla como única. Cabe mencionar que este proceso puede tardar hasta cinco años, por lo que, mientras concluye el trámite, se otorga un registro del mismo. Aunado a esto, el registro también puede hacerse a nivel internacional, en varios países simultáneamente, dependiendo del mercado que se quiera explotar. La duración de una patente es de 20 años a partir de la fecha en que fue sometido el registro.

En el caso de los videojuegos, el producto puede basarse en un nuevo proceso descubierto que acelere o revolucione la forma de producir o de interactuar. Su anotación cae en el término de derechos de autor como *software* y producto multimedia. Queda en términos similares al registro de un libro o una composición musical, y protegida por la vida del autor durante los 100 años posteriores a la muerte del mismo antes de que pase a dominio público.¹⁰ En el país esto se hace en el Instituto Nacional de Derechos de Autor (Indautor), que depende de la Secretaría de Educación Pública (SEP).¹¹ El tiempo de registro es rápido; desde un par de semanas hasta un mes.

Asimismo, para la creación de un videojuego es necesario hacer el registro de marca en el país. Este es un trámite que se realiza en el IMPI, y puede tardar de una semana a un mes máximo.

⁹ Véase <http://www.impi.gob.mx/>

¹⁰ En Estados Unidos son 70 años después de la muerte del autor.

¹¹ Véase <http://http://www.indautor.sep.gob.mx>

Por otro lado, como ya se dijo, los secretos industriales, en caso de tenerlos, deben estar resguardados con un convenio de confidencialidad en la empresa, limitando la diseminación de la información relacionada con el secreto que se protege y el material que posteriormente pueda ser motivo de patentes o registro de derechos de autor. Para evitar que los empleados o vendedores puedan sacar beneficio personal de los desarrollos que se están realizando, es importante limitar los controles de acceso a la información y tener contratos claros que detallen que la empresa es dueña de la propiedad y la autoría; que los vendedores no pueden contratar a empleados clave de la empresa (si sucede esto, el empleado clave se convierte en competencia), así como otras cláusulas que eviten este tipo de problemas.

Metodología Agile Scrum para el control y gestión del desarrollo de los proyectos de videojuegos serios

Uno de los principales factores de éxito que ayudan a concluir satisfactoriamente los proyectos de videojuegos serios, logrando los objetivos planteados inicialmente, tiene que ver con la estrategia de desarrollo del equipo de diseño de estos (véase figura 1.1).

Debido a la presencia inevitable de variables que pueden generar restricciones sobre los objetivos (presupuesto disponible, tiempo de entrega del proyecto, recursos humanos, materiales asignados, cambios de versiones integrando nuevas funciones y curvas de aprendizaje en algunas herramientas de desarrollo), la tarea de seguimiento de un proyecto no es simple. Incluso las actividades de integración de resultados, por parte de los diferentes equipos de trabajo en el proyecto, pueden tener una alta complejidad y consumir más tiempo y recursos humanos, materiales y económicos de los planeados inicialmente.

Por ello, el equipo de diseño no solo debe establecer los objetivos claros del videojuego serio que se quiere lograr, sino que también debe dar seguimiento a todas las actividades del proceso de producción, y contar con las herramientas y metodologías que faciliten esta compleja labor que conlleva un alto nivel de multidisciplinariedad demandada. En este sentido, a continuación se presentan las técnicas probadas por el equipo de redacción de esta obra, y que, como retorno de experiencia, está confirmada su utilidad y funcionalidad. Es necesario aclarar que algunas de las herramientas y técnicas que se plantean tienen el objetivo de dar una referencia y orientación general, ya que hay autores que tratan el tema de la gestión de proyectos en obras completas tales como Klooppenborg (2011), Lewis (2007), Schwaber (2004) y Pichler (2010), y sobre las que convendrá posteriormente que los líderes de proyecto se respalden.

Dentro de lo que se propone para lograr un buen resultado en un equipo de diseño, en un principio se sugiere la aplicación de esta metodología en equipos pequeños —de 5 a 15 personas—, que sería lo equivalente a la conformación de una pequeña y mediana empresa (pyme), ya que es en este nivel donde se cree que puede tener mayor impacto. Es por esta razón que no se profundiza en aspectos de ingeniería de *software* en un sentido estricto, es decir con todos sus ciclos y fases de desarrollo; solo se enuncian algunas prácticas y herramientas que pueden ser de utilidad y pueden facilitar la labor de integración y de control del seguimiento y desarrollo de cada proyecto de videojuego serio. Adicionalmente, tampoco se tomaron de forma integral todas las reglas y prácticas del área de ingeniería de proyectos, dado que en un inicio, cuando se tiene poca experiencia en el desarrollo de este tipo de videojuegos, la planeación de proyectos puede ser frustrante debido a la gran cantidad de consideraciones que son necesarias, y que solo se dominan cuando se obtiene más experiencia.¹²

Algo muy importante en el desarrollo de videojuegos son los objetivos descritos en la historia y el guión, que permitirán programar las diferentes tareas para los equipos de arte visual, sonora y de programación a manera de desarrollar el producto final. Estas tareas serán descompuestas en la metodología de gestión de proyecto con una estimación de tiempo y costo por cada equipo de producción de arte y programación, con lo que se integra el proyecto final. También las actividades van de acuerdo a los requerimientos que establecen los inversionistas y el dueño del producto final, lo cual se logra involucrándolos en el proceso de planeación y con un seguimiento continuo. Para conseguir lo anterior, se requiere entonces la metodología ágil para gestión del proyecto que será presentada en la siguiente sección.

Las metodologías ágiles como una solución para equipos de desarrollo de videojuegos serios

Desde una perspectiva tradicional, un equipo de desarrollo de videojuegos se podría basar solamente en los principios de gestión de proyectos convencionales y trabajar en los objetivos principales del juego. Posteriormente, generaría una descomposición de todas las actividades necesarias para lograr dichos objetivos e identificaría todas las relaciones y dependencias plasmándolas en un calendario y asignando costos. Como resultado de lo anterior le sigue un diagrama de Gantt, sobre el cual se traza la ruta crítica que se tiene que cuidar para que

¹² Para profundizar tanto en la ingeniería de *software* como en la ingeniería de proyectos hay excelentes obras que se pueden adquirir o consultar en las bibliotecas públicas o directamente en internet.

todas las actividades puedan fluir y el proyecto llegue a su fin; no obstante, todo lo anterior funciona cuando se tiene una experiencia en la gestión de proyectos y, además, existe una completa certeza de las características del producto que se quiere generar. Por ello, la recomendación es trabajar en el desarrollo de videojuegos empleando metodologías ágiles, particularmente aquella denominada Scrum,¹³ ya que durante este proceso la complejidad de integración, los avances y los cambios de las plataformas de desarrollo tecnológico presentan un cierto nivel de incertidumbre que no es tan fácil de medir, sobre todo cuando se está incursionando en el tema.

Una de las características de las metodologías ágiles es que, a diferencia de las tradicionales de gestión de proyectos que esperan ciclos largos para integrar y dar por completo el proyecto, se busca iterar muy rápido. Asimismo, en lugar de definir objetivos muy precisos a cumplir, se intenta que estos surjan de forma global en cada iteración, pero con una gran capacidad de adaptación a las condiciones que se tengan en el presente al cierre de cada repetición, conocida como un ciclo o *sprint*. Otra diferencia es que a través de equipos pequeños busca motivar altamente la interacción entre todos los participantes para cumplir las metas como equipo y no tanto en lo individual, y así no tener participantes inactivos.

Para llevar a cabo este tipo de metodologías es necesario contar con herramientas de gestión que permiten que todo el equipo de desarrollo pueda ver cómo está trabajando en cada ciclo. También se estimulan las reuniones breves de cinco minutos todos los días, para poder estimar los logros y generar un sentido de pertenencia. En la actualidad las metodologías ágiles son empleadas en la industria de la alta tecnología, y han sido el factor clave para el desarrollo de productos innovadores ante la incertidumbre tecnológica generada por sus cambios continuos, donde es sustancial tener una gran capacidad de adaptación, como en el caso del Valle del Silicio (Silicon Valley en inglés) en California, EEUU, donde hay una alta concentración de compañías involucradas en la fabricación de computadoras y en el desarrollo de *software* (Parrado, 1999).

Principios básicos de Agile Scrum

En primer lugar, esta metodología busca establecer los diferentes roles que participan en el desarrollo de un producto, los cuales son: el dueño del producto, el líder del producto, el maestro Scrum y el desarrollador.

Dueño del producto. Representa al cliente o a los inversionistas que están interesados en el proyecto de videojuego serio que se quiere desarrollar, por lo que es

¹³ <http://www.scrum.org>

posible tener más de un dueño. Para comenzar establece una lista de propiedades y características que desde su punto de vista debe tener el videojuego al final de su desarrollo. Al cierre de cada ciclo de trabajo, que regularmente puede ser cada dos o tres semanas, se reúne con el equipo y autoriza satisfactoriamente que se lograron las características y propiedades establecidas como metas en determinado ciclo. Esta figura es importante porque existen algunos desarrollos en que el cliente y futuro dueño del producto solo interactúa con el equipo de desarrollo al final, cuando, tal vez por una mala interpretación, no se logran los requerimientos y expectativas iniciales con el proyecto. Es importante mencionar que la interacción frecuente con el dueño del producto permite que los equipos de desarrollo lo entiendan como usuario final y facilite una mejor comunicación y comprensión del producto en desarrollo. Con base en esto, al inicio del proyecto se tiene una reunión para establecer todas las propiedades del producto, que se denominarán *historias*. Entonces, un producto completo y todas sus propiedades están descritas en todas las historias equivalentes, de forma muy general, a todos los objetivos que se deben cumplir.

El líder del proyecto. Es quien tiene toda la visión integral y experiencia de las diferentes áreas de trabajo para la producción completa de los videojuegos serios. Se encarga de monitorear todas las actividades de los equipos y de generar los reportes de avance a los dueños del producto y al equipo en general. En pocas palabras, es el gestor completo del proyecto, necesario para que se puedan integrar los resultados de todo el equipo de desarrollo. Con el tiempo, es quien facilita el llenado y documentación para definir las historias y las actividades que se derivan. También acuerda con los equipos de desarrollo qué historias se realizarán en cada ciclo o *sprint*.

El maestro Scrum. Es el líder que representa a cada uno de los equipos de desarrollo. Para los videojuegos serios se tienen tres equipos básicos, que son los de desarrollo, arte y programación (véase figura 1.1). Los líderes de cada equipo cuentan con experiencia en los procesos de desarrollo, y su misión es buscar resolver cualquier problema que se presente a los desarrolladores y administrar los resultados de los mismos. También es un integrador y promueve la proactividad y capacidad de adaptación ante los cambios que se puedan presentar o los requerimientos nuevos que se den de alta.

El desarrollador. Es la base unitaria del equipo de desarrollo que, por lo general, se agrupa en equipos de cinco personas (si bien el número de desarrolladores puede ser mayor o menor, este es el tamaño recomendable). Su rol es estimar para cada historia que se decide qué va a entrar en un ciclo para su trabajo, cuál es la prioridad de la misma y el esfuerzo que costará realizarla. De esta manera, se establece de

forma básica una ruta crítica buscando resolver las historias más prioritarias desde un inicio y cerrando las que tienen menos peso o importancia. Junto con el maestro Scrum, por cada historia, los desarrolladores dan de alta una serie de actividades en lo cotidiano, para que la historia se pueda cumplir con todos los requerimientos establecidos. Aunado a ello, se fija una estimación de horas necesarias de trabajo para poder lograrla, por lo que una historia tiene varias actividades que, a su vez, representan una cantidad de tiempo a invertir para poder lograrla y que se refleje en el producto final. Ahora bien, cuando se están ejecutando las actividades de una historia dentro de un ciclo, se tienen tres posibles estados: 1) aquellas por iniciar, 2) en proceso y 3) terminadas o cerradas. Al respecto, cada desarrollador puede ver las historias por iniciar y elegir una, de forma tal que se vuelve dueño de la misma. Según los resultados que vaya teniendo el desarrollador, irá disminuyendo el tiempo restante para terminar la actividad hasta cerrarla. Cabe la posibilidad de que este no pueda concluir una historia, y entonces la ceda a otro desarrollador para que la termine. Así, al inicio de un ciclo todas las actividades declaradas de una historia están en un punto inicial y terminan como cerradas, con lo que la historia puede ser evaluada por el dueño del producto y determinar si se puede cerrar.

Otro principio básico de Scrum es que cuando se trabaja con esa metodología, las únicas fechas de entrega de resultados son al cierre de cada ciclo; sin embargo, las rutas críticas y la duración de actividades son establecidas por los equipos de desarrollo, permitiendo que el equipo vea resultados continuos y alcanzables en función de las capacidades que va realizando. En este sentido, en cada ciclo se tiene una integración de resultados, así como la aprobación de los mismos por los dueños del producto para poder proseguir con el desarrollo.

Igualmente, esta metodología facilita tener una métrica de la eficiencia con la que trabaja todo el equipo de desarrollo, la cual sirve para revisar cómo mejora su desempeño con base en la experiencia lograda. Cada ciclo permite una nueva oportunidad de mejora, de adaptación y de entender lo que busca el cliente o usuario final mientras se lleva una documentación no tan intensa —como sucede en los métodos tradicionales—, pero suficiente para no perder de vista los objetivos que se quieren lograr.

Por último, una vez que se cierra un ciclo y se da inicio a uno nuevo, cada equipo de desarrollo tiene lo que se llama una sesión de retrospectiva, donde se busca plantear, por un lado, que lo que se ha hecho funciona, para que siga y se reconozca; por otro, que se identifique lo que no se debe seguir haciendo, pues impacta de forma negativa al equipo de trabajo; y, finalmente, que también se señale aquello que se vea como nuevo, que nunca se había implementado, pero que se vuelve necesario empezar a hacer. Todo esto permite, como se comentó previamente, que el equipo siempre esté en capacidad de asimilar cambios y adaptaciones, en la búsqueda siempre un mejor desempeño.

Ya contando con una descripción a grosso modo de la metodología ágil, en la figura 1.4 se puede apreciar cómo opera el desarrollo de un videojuego serio con Scrum.

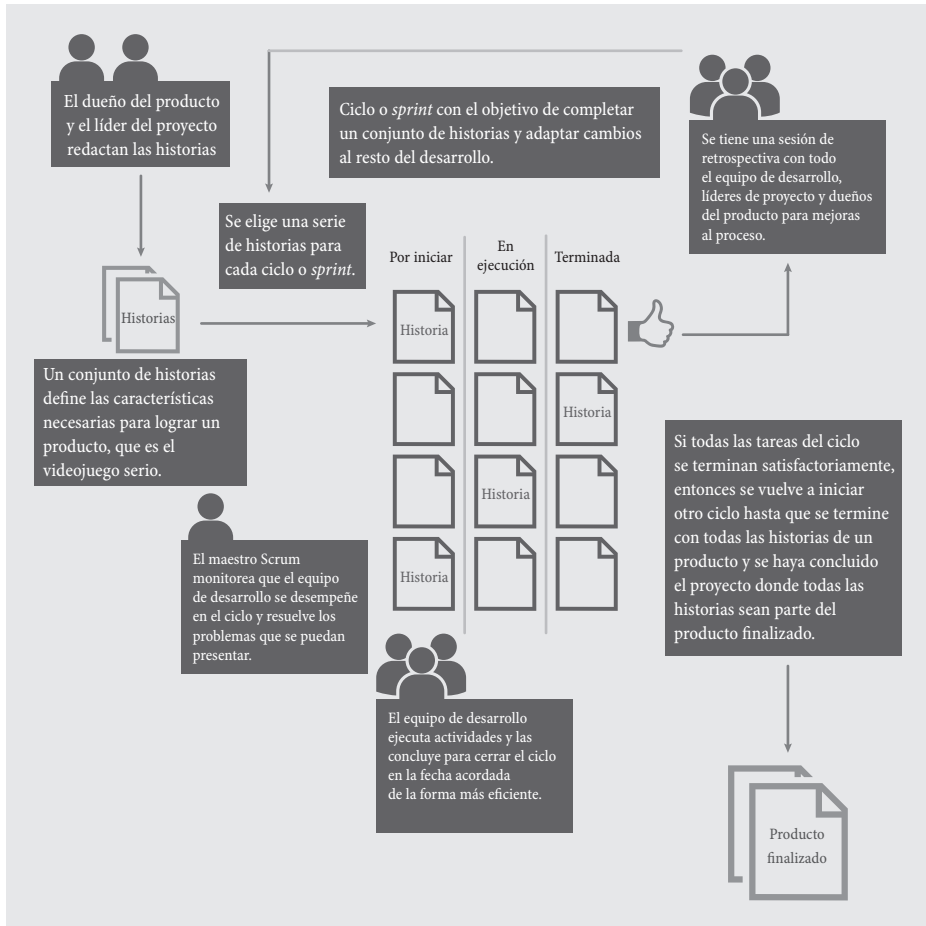


Figura 1.4. El proceso de desarrollo del proyecto de un videojuego serio mediante la metodología Scrum

Herramientas de soporte para gestión y seguimiento del proyecto

Kunagi para soporte de procesos Scrum

La metodología Scrum se puede llevar a cabo manualmente (con pizarrones) o se puede tener el respaldo de una plataforma informática que facilite el trabajo y los procesos metodológicos. Si se decide por la segunda opción, dentro de una gama de posibles herramientas se encuentra la plataforma que se instala en un servidor web denominada Kunagi.¹⁴ Esta plataforma permite que todos los participantes de un proyecto dispongan de una cuenta; proporciona la ventaja de contar con tableros electrónicos para que todos vean el estado del proyecto, repositorios para documentación y archivos propios de los entregables en formato electrónico de manera básica. Además, por medio de un tablero, respalda en una gráfica la velocidad en la que se está desempeñando el equipo con relación a la estimación de tiempo para todas las actividades a cubrir en un ciclo o *sprint*. Finalmente, promueve la proactividad de los desarrolladores desde un acceso a la plataforma, gracias a que pueden ver el estatus de todas las actividades y enterarse si han terminado con las que tenían asignadas o retomar otras para respaldar la velocidad de trabajo del equipo en general.

Algo muy importante que se debe tener en cuenta en todos los proyectos es la identificación y gestión de riesgos, aunque sea básica. En este sentido, la plataforma de Kunagi permite que ante una tarea que enfrenta un problema, se pueda crear un riesgo y asociar este a dicha actividad y, por ende, a una historia. Así es como los maestros Scrum, líderes de proyecto y dueños del producto pueden ver todos los riesgos que emergen desde los equipos de desarrollo y tomar un rol activo para decidir cómo los van a resolver y cómo se adaptará la historia. Aunque el tratamiento es sencillo, el manejo de riesgos queda respaldado en la documentación, brindando una ayuda invaluable para la experiencia que se va generando para futuros proyectos de videojuegos serios, así como para que los inversionistas o dueños del producto estén sensibles al costo real de desarrollo del mismo.

Herramienta complementaria para diagramas de Gantt y gestión global del proyecto

Si bien la herramienta de Kunagi es suficiente para realizar pequeños proyectos, no permite dar una visión global a los inversionistas sobre el seguimiento de la

¹⁴ Véase <http://www.kunagi.org>

ruta de trabajo con las actividades más importantes basadas en las historias, ni es capaz de representar los riesgos u oportunidades que pueden arribar en el proyecto. Por lo tanto, en algún punto, aunque hay toda una programación de actividades con Agile Scrum, parte de los inversionistas estarán esperando ver una ruta crítica de actividades en las cuales se debe focalizar el equipo para garantizar la inversión. Una recomendación al respecto es transcribir una síntesis de todas las historias y la secuencia que se espera en un diagrama de Gantt, marcando los recursos y estimación de costos, así como riesgos y objetivos contra un cronograma.

En caso de no querer hacerse una inversión en un *software* como Project de Microsoft,¹⁵ lo ideal es trabajar con Gantt Project,¹⁶ ya que es una herramienta en licencia de código abierto multiplataforma (no tiene costo) mediante la cual se pueden proyectar todas las actividades a desarrollar, poner su prioridad, recursos asignados, fecha de inicio y fin. Una vez cargada la información, se genera un diagrama de Gantt con la ruta crítica y un reporte global muy completo, que se puede ir actualizando cada dos o tres ciclos de Scrum con la finalidad de mantener los objetivos claros y, sobre todo, para que los inversionistas tengan una mayor seguridad sobre el desarrollo sano del proyecto, además de los riesgos y alcances de este.

Soporte de control de versiones y repositorio de documentos

En todo proyecto y para cada uno de los equipos se vuelve importante llevar un control de versiones, con un almacenamiento seguro y que tenga puntos con diferentes niveles de acceso para modificar o leer los documentos o archivos digitales que se vayan generando. De acuerdo con la estructura organizacional para desarrollar videojuegos, se pueden mencionar algunos ejemplos de lo que cada uno de los grandes equipos puede estar almacenando y controlando.

Equipo de diseño de videojuegos. Requiere de un repositorio donde vaya almacenando las diferentes versiones de los documentos de historia, los *storyboards* para revisión y toda la gama de modificaciones subsecuentes, los reportes de métricas de avance y medición del proyecto, etc. Todo lo anterior debe contar con niveles de acceso a la información solo visible para el equipo de diseño. Asimismo, deberá clasificarse una información para el equipo de arte, otra para el de programación e incluso otra donde los inversionistas o dueños del producto puedan ver demostraciones de las diferentes versiones del juego y su progreso.

¹⁵ Véase <http://www.microsoft.com/mexico/estrategicos/adminproyectos/>

¹⁶ Véase <http://www.ganttproject.biz>

Equipo de arte. Este equipo generará muchos archivos digitales, los cuales, para conveniencia del proyecto, tendrán que estar en secuencia progresiva y organizada. Con ello, los objetos 3D generados para una escena, por ejemplo, pueden ser reutilizados para otras; o las texturas de estos objetos, con algunas modificaciones, pueden servir de base para escenas de día o en lugares oscuros. De la misma manera, los personajes pueden ir sufriendo adecuaciones, pasando de un concepto abstracto en el guión a un concepto evolutivo más adaptado a toda la idea de la historia del videojuego, donde de forma creativa es necesario explorar versiones de las cuales se tome un camino y luego se retorne a versiones anteriores para poder comparar lo desarrollado. Finalmente, también en términos de efectos auditivos o audiovisuales, se vuelve necesario contar con varias versiones, de simples a complejas, por lo que se requiere tener una buena estructura de almacenamiento.

Equipo de programación. Realiza una práctica indispensable en el proceso, dado que la programación de *scripts* de animación puede iniciar con versiones primitivas, y en versiones posteriores puede integrar una gran complejidad algorítmica. Este equipo, además de llevar el control de código por versión en un repositorio, también debe hacerlo cuando se realizan desarrollos simultáneos en librerías, con la intención de que todos los desarrolladores de programación puedan estar trabajando de manera concurrente siempre bajo la versión vigente. Al igual que con el equipo de arte, el código de los programadores puede tener diferentes ramas evolutivas, debido a que en algunos momentos puede ser de gran utilidad regresar al origen de algunas ramas y programar o crecer el código en otra dirección, sobre todo cuando se requiere programar para diferentes plataformas operativas.

Para llevar dicho control de versiones se recomienda el *software* libre de código abierto Alfresco,¹⁷ en su versión para comunidades de desarrolladores. La herramienta se puede descargar para su ejecución como un servicio en la web, y así, con una vasta documentación, poder crear repositorios con soporte de control de versiones. Hay que señalar que estos repositorios tienen la ventaja de que se pueden personalizar al dar de alta diferentes grupos de usuarios con diferentes niveles de privilegios, para que, con base en una jerarquía, permitan facilitar el nivel de acceso y controlar los diferentes puntos de información entre pública y privada. También existe una versión de Alfresco que opera en la nube y que, mientras no se exceda la capacidad de 10 GB, no tiene costo, lo que permite tener una cantidad ilimitada de cuentas de acceso para una organización o empresa. En caso de exceder esta capacidad de almacenamiento, una opción viable —si no se tiene un servidor donde se pueda instalar el *software* y no se quiere gastar en

¹⁷ <http://www.alfresco.org>

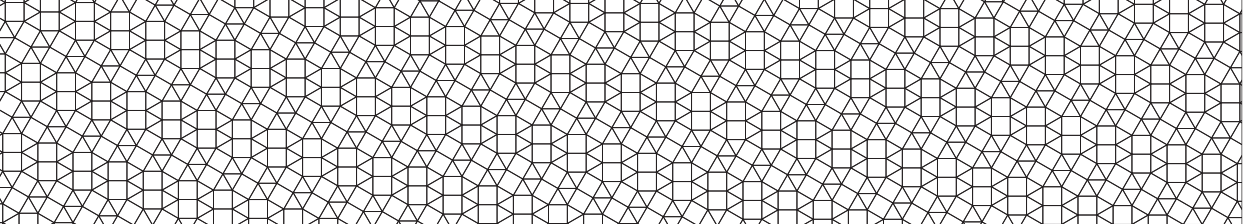
mantenimiento del sitio y soporte técnico— son los planes de renta mensual, que oscilan sobre los 10 dólares americanos, con derecho a soporte técnico.

Conclusiones

Esta sección tuvo el objetivo de presentar la importancia y evolución de los videojuegos en general, estableciendo un énfasis en los videojuegos serios. En casi 40 años de desarrollo de estas tecnologías podemos concluir que con las plataformas de cómputo actuales y equipos relativamente pequeños, es posible generar videojuegos serios de una calidad aceptable. Dichos videojuegos proponen un área de oportunidad para diferentes nichos estratégicos como son la industria, la educación, la medicina y el gobierno. El desarrollo y acceso a estas tecnologías que son propiamente sustentadas en la creatividad de las tecnologías digitales implican una economía basada en la gestión del conocimiento. Con el dominio de las técnicas será posible, entonces, desarrollar regiones más competitivas, ya que ellas reemplazan, mediante el plano digital, la necesidad de infraestructura real con un alto costo.

Otro de los objetivos de este capítulo fue también mostrar la alta multidisciplinariedad requerida para poder desarrollar los videojuegos serios, lo cual plantea nuevas estrategias para comunicar y formar el talento que desee trabajar en estas plataformas digitales, dado que hoy en día la educación está muy especializada y enfocada a nichos con poca interacción con otras disciplinas. Por este motivo se requiere romper el esquema de gestión de proyectos tradicionales y se introduce la organización de proyectos con las metodologías ágiles. Lo anterior permite que equipos de trabajo multidisciplinarios desarrollen habilidades de comunicación e interacción como equipo para lograr objetivos a corto plazo que facilitarán en varias iteraciones llegar al desarrollo del producto final. Se vuelve muy importante mencionar que si bien tiene un valor muy importante el guión de un videojuego, su implementación depende de muchas tecnologías tanto para los equipos de arte y programación que están en continua evolución, ya que esto implica la necesidad del desarrollo de capacidades de adaptación rápida a los cambios, uno de los puntos que promueve la metodología de gestión de proyectos de Agile Scrum, misma que además fue probada por los autores de esta obra.

Finalmente, para facilitar aún más el desarrollo, se presentaron algunas herramientas tecnológicas y flujos de procesos a implementar que pueden respaldar la organización de los equipos de trabajo. También se discutió, siendo el desarrollo de videojuegos serios respaldado en una economía del conocimiento, lo importante de respetar la propiedad intelectual: un proyecto puede desaparecer si emplea recursos que no tengan permisos o tengan cubiertos los derechos de autores que se estén integrando en los equipos de arte visual y sonora.



CAPÍTULO 2

Elementos de la cadena de producción

VÍCTOR MANUEL LARIOS ROSILLO

ÓSCAR FERNANDO GONZÁLEZ DELGADILLO

Introducción

Existen desarrolladores de videojuegos que apuestan todo por los gráficos, y si bien son una parte importante, también lo es el hecho de que contenga una buena historia.

Si se hace un cotejo del paralelismo entre una película y un videojuego, el resultado es una estructura con características muy similares, que de acuerdo con el orden de importancia, los fundamentos y pilares que soportan las diferentes consolas y plataformas donde se puede ejecutar un buen videojuego inician por una historia sólida, seguida del arte visual y sonoro y la programación (véase figura 2.1).

Asimismo, para poder generar videojuegos de forma eficiente se requiere de una secuencia de procesos, con la finalidad de ilustrar los conceptos fundamentales en cada elemento de la cadena de producción, para que, independientemente de los avances tecnológicos, todo lo presentado siga siendo vigente y adaptable (véase figura 2.2).

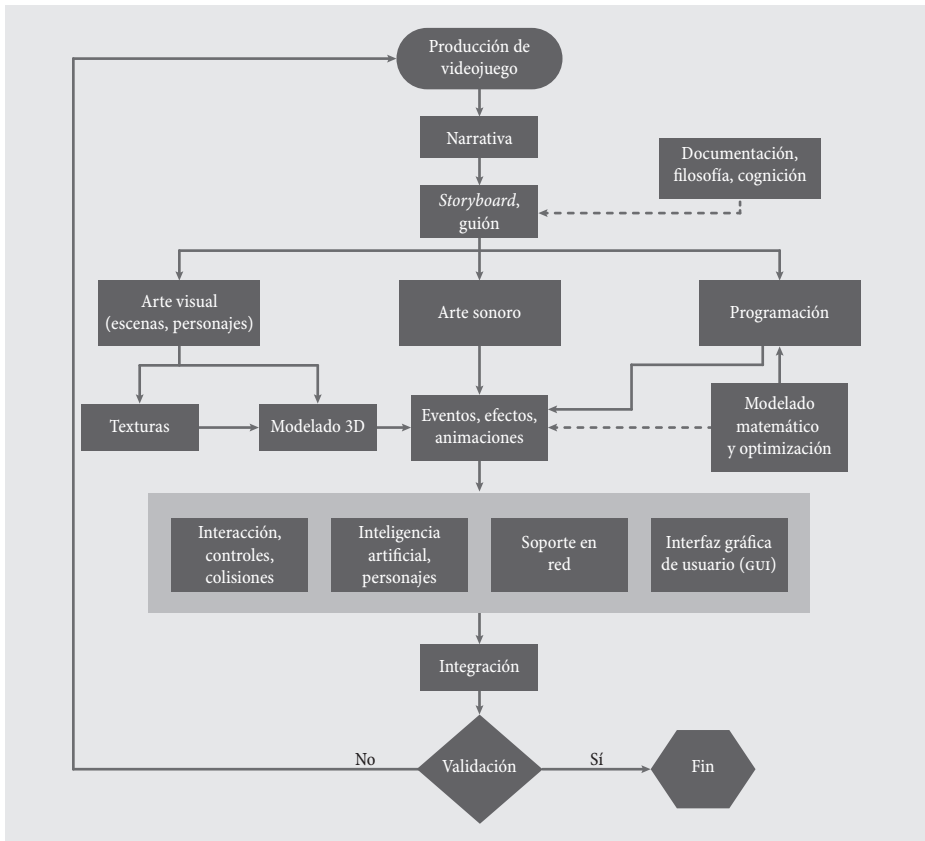


Figura 2.1. Estructura con las bases y pilares importantes en la producción de videojuegos

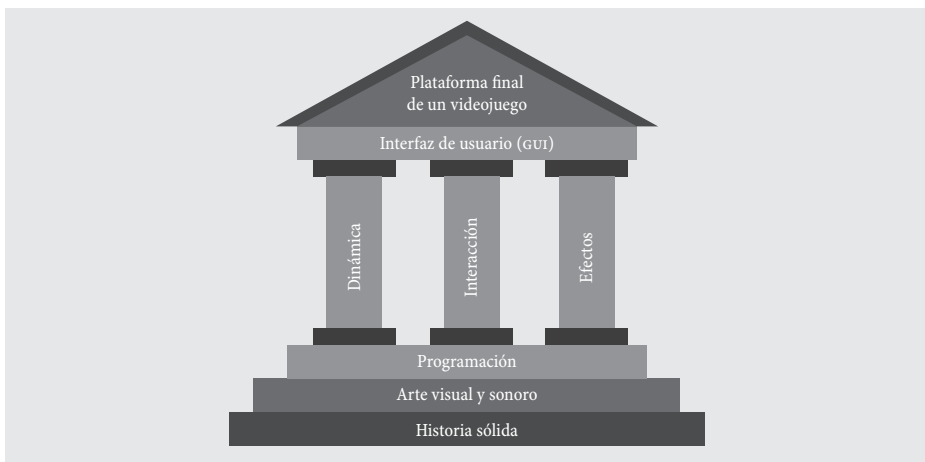


Figura 2.2. Secuencia de procesos en general para una cadena de producción de videojuegos

Cadena de producción de videojuegos

Dentro de la cadena de producción de un videojuego, viéndola tal como si fuera una receta, se tienen los siguientes ingredientes:

- *La historia.* Un buen guión es vital para que el videojuego sea inmersivo.
- *Arte visual.* Debido a que un 80% de nuestra percepción es visual, se requiere de trabajo de modelado 3D, texturizado y manejo adecuado de la iluminación.
- *Arte sonoro.* Cuando se tiene un soporte de distancia, los sonidos son clave para mover los estados de ánimo del jugador y llevarlo acorde con el guión.
- *Programación.* Genera los efectos especiales y lleva toda la secuencia de control del juego en sí.
- *Dinámica e interacción del juego.* Se relaciona con la coherencia del mundo propuesto según las leyes físicas y comportamiento de objetos, entre otros.
- *Efectos.* Estos están cubiertos en el arte sonoro y visual; sin embargo, es importante que estén bien ligados a la programación, ya que son un elemento que puede hacer la diferencia en la experiencia del jugador.
- *Interfaz de usuario (GUI).* Permite al usuario tener la información sobre su estado en el juego, además del control en él, los elementos que ha colectado o que puede emplear y la información complementaria que le deje progresar en este.
- *Plataforma de videojuego.* Es la consola o *hardware* de destino final, que puede ser desde un teclado (computadora), un *joystick* o sensor con reconocimiento de gestos por videocámara (consolas Xbox Kinect, PS3 Move) hasta las pantallas táctiles con acelerómetros (Apple iPad, iPod Touch, iPhone, Android, etc.) o que requieren una adaptación particular debido a las tecnologías que emplean.

Elementos para la construcción de la historia

En la historia de un videojuego se trata de establecer el nivel de abstracción para dar una base de narrativa con una secuencia a seguir y establecer cómo lograr los objetivos planteados, derivando en las reglas y la dinámica del juego. Si la historia no se sustenta en la implementación del juego, entonces este no tendrá lugar y el proyecto no tendrá éxito.

En términos de la historia, lo primordial es precisar un inicio u origen que establezca una necesidad de iniciar una aventura (nuestro juego propiamente); posteriormente se debe emprender un hazaña que especifique un objetivo a cumplir y ciertos retos a superar para lograrlo; y finalmente, cuando los objetivos se

cumplen, se tiene entonces el clímax o parte final para lograr un desenlace y terminar el juego. Todo lo anterior se conoce como la narrativa del juego, aunque es importante saber que antes de iniciar la redacción de la misma, hay que identificar dos elementos fundamentales: primero, los lugares donde se desarrolla la historia, con una estructura que pueda ser modificada por el usuario (Jenkins, 2004), y segundo, los eventos y las formas como estos se presentan y el establecimiento reglas que regulan su desarrollo (Frasca, 2003). Por un lado, con referencia a los lugares de desarrollo de narrativa, se pueden considerar los siguientes puntos:

- Hay que determinar cuáles son los lugares y espacios donde se desarrollará la narrativa. Estos espacios tendrán que evocar emociones que refuercen los objetivos y conflictos para lograrlos.
- Los lugares deberán contar con incidentes —conocidos como micronarrativas— que permitan saber que tiene o no sentido seguir en ese lugar para avanzar a otro que signifique continuar en la narrativa.
- En función de los lugares, el usuario afecta el estado de los objetos que se encuentran ahí, dando la pauta para saber cómo irá estructurando la narrativa.
- Entre la navegación que el usuario tiene sobre los lugares, se definirá hasta dónde se permite que el usuario construya narrativas emergentes.

Por otro lado, en relación con los eventos del videojuego, hay que tomar en cuenta lo que se dice enseguida:

- Lo que debe suceder en cada situación que se presenta.
- Los modelos de manipulación que permitan al usuario saber lo que puede hacer en el evento.
- Las reglas para lograr un objetivo y la aplicación por parte del jugador.
- Las metarreglas que se aplican de acuerdo con los niveles logrados y la evolución en la narrativa.

Así pues, como ya se mencionó anteriormente, una vez que se empieza con una narrativa definida, le sigue la construcción de la historia, en donde se identifican tres partes importantes (Tosca, 2003):

- La convocatoria (configuración).
- El emprendimiento de la misión (confrontación).
- El regreso (desenlace).

La *convocatoria* corresponde a un llamado espiritual que se realiza al héroe del juego. Aquí se le indican el objetivo para poder ayudar a una causa y la necesidad de emprender una aventura mostrando todas las condiciones iniciales.

El *emprendimiento*, por su parte, se refiere al camino que seguirá el jugador para cumplir la misión, con las reglas necesarias para que sepa cómo lograr el objetivo y cómo confrontar los retos que aparecerán.

Finalmente, el *regreso* equivale a haber logrado el objetivo y recibir una recompensa. Con ello cierra toda la historia, la cual, según el desempeño del jugador, tendrá un determinado final. Un ejemplo de un videojuego con una gran narrativa sería la historia de una princesa capturada en una torre por una hechicera, quedando custodiada por un dragón. De esta manera, la primera parte sería cuando se convoca al jugador que tomará el rol del héroe a rescatar a la princesa. Para esto, previamente tiene que recibir una explicación de que debe emprender un camino lleno de trampas hasta encontrar la torre custodiada por el dragón, combatirlo y rescatarla. Este llamado a la aventura, la necesidad de ayudar a la princesa como una causa noble y los retos con los medios para poder lograr su cometido entran en la segunda parte de la construcción de la historia; el héroe emprende la aventura, de la cual debe desarrollar habilidades para poder sortear los lugares con trampas durante su jornada. Por último, una vez concluida la misión y rescatar a la princesa, este héroe recibirá una recompensa, como puede ser un reconocimiento, un nombramiento como caballero real u otras formas que le dejen al jugador una satisfacción por la aventura realizada.

Otros detalles importantes que, al mismo tiempo que la historia, son necesarios, es la identificación de los personajes que participarán en la narrativa, los cuales se pueden clasificar como principales y de soporte. Los principales incorporan al protagonista, al antagonista, a aquel que convoca a la misión, al donador, al proveedor de ayuda, a la persona buscada, y a un falso protagonista. Entre los personajes de soporte podrían incluirse a los miembros de la familia y a aquellos que se definen como conectores, debido a que permiten avanzar en una secuencia lógica entre los diferentes lugares y eventos relacionados con el objetivo a alcanzar.

Entre las consideraciones de la historia se pueden identificar diferentes modelos para estructurar la secuencia de eventos y lugares. Estos pueden ser:

- Lineal
- Ramas
- Ramas controladas
- Telaraña
- Modular

En la figura 2.3 se expone toda una historia con inicio y fin, pasando por diferentes escenas que son parte de la narrativa. Lo que se quiere representar es la conexión que se puede dar entre las diferentes historias, es decir simple o compleja; esta última se reproduce en el caso de la historia en telaraña. Igualmente, en cada escena habrá una serie de retos y actividades a desarrollar para poder pasar al siguiente módulo, donde puede o no haber una descisión del jugador en función de la conexión.

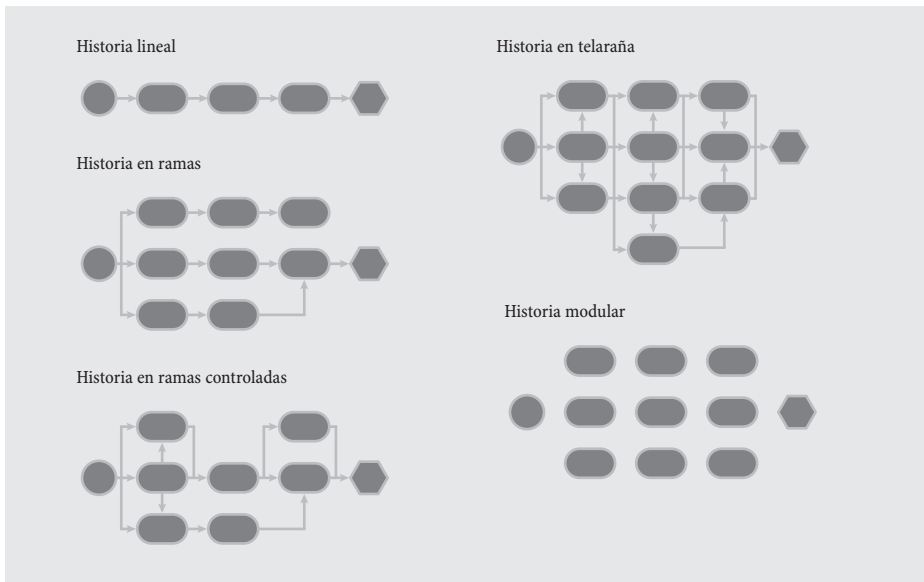


Figura 2.3. Los cinco tipos de historia en que se puede estructurar el guión de un videojuego, donde cada bloque representa una escena y la forma como esta se interconecta con las otras debe llevar al final de la historia completa

Una forma de poner en práctica todo lo anterior es comenzar con la narrativa de un juego tomando un documento modelo. Inicialmente el documento contará con todos los detalles en narrativa de cómo se desarrollará el juego, por lo que será una referencia para todos los equipos. Este documento puede tener ilustraciones y lo que se conoce como *storyboards* que esclarecen a los diseñadores gráficos y artistas digitales la forma en que se deben presentar los detalles de las escenas descritas en la narrativa. (Domsch, 2013; 1p, 2011; Kennerly, 2008; Egenferldt-Nielsen, y Pajares, 2012).

Modelado 3D: escenario

El escenario es una parte elemental del realismo de un juego y puede incentivar la inmersión del jugador en el mismo; no obstante, se debe de encontrar un balance entre la inversión (tiempo, dinero y esfuerzo) que representa la realización de dicho escenario y el beneficio percibido por el jugador. En términos de recursos de procesamiento y memoria del dispositivo final, el modelar todos los detalles del escenario con polígonos puede ser algo muy costoso. Es por este motivo que el modelado de objetos 3D va respaldado de diseñadores gráficos que buscan

obtener el máximo realismo de la escena mediante fotomontajes en diferentes planos de texturas en 2D, así como un trabajo de los detalles de texturas 2D de objetos con pocos polígonos, pero con geometrías representativas del objeto final.

La complejidad del texturizado en 2D reside en la valoración de la iluminación esperada de la escena y las tomas de la cámara del videojuego, para que los efectos de sombras y todas las texturas sobrepuestas en los objetos 3D se vean uniformes y coherentes. Al final del proceso, la escena debe engañar a los ojos del espectador, con el propósito de generar el máximo realismo y una experiencia de inmersión en el juego.

Arquitectura del escenario

Una vez más, el arte y la ingeniería se entremezclan para concebir el trabajo de la escenografía. Al igual que las obras de teatro o las películas, los videojuegos también requieren de un entorno espacial que permita contextualizar la historia y que le dé soporte al mensaje que se pretende transmitir (Chalmers, Howard y Moir 2009; Claypool, 2009; Ploor, 2011).

El diseño del escenario es el arte de crear experiencias basadas en la observación; se trata de la percepción controlada del espacio (Lamster, 2000). Hay que tener en cuenta que por más simple que sea un escenario en la vida real, en una computadora se representa un nivel de alta complejidad debido a todas las variables de control que involucra para su modelado. Es entonces que se vuelve relevante el hecho de engañar la vista; se necesita imitar la complejidad visual de los escenarios que hay en la vida real. Un buen diseño de los escenarios es un trabajo vital que requiere, en gran parte, del talento artístico en el diseño gráfico, pero acompañado de las técnicas computacionales para optimizar todos los modelos.

El tipo de escenario a realizar estará en función de lo que se haya establecido en el guión; sin embargo, un escenario completamente plano (terreno), carente de obstáculos, puede desmotivar al jugador, pues es posible que perciba un pobre manejo de los elementos del juego. En mayor o menor medida casi todos los juegos contarán con algún tipo de construcción, sea esta un edificio o una simple pared. En otras palabras, se puede decir que la estructura de edificios y objetos arquitectónicos que constituyen una escena, acorde con la historia, las animaciones de objetos y personajes, contribuyen a crear estados emocionales en el jugador (Lamster, 2000).

Se estima que el elemento arquitectónico 3D completo más simple que hay en un escenario es un edificio, ya que puede ser representado fácilmente con pocos polígonos mediante un cubo (o un rectángulo) con una o varias texturas, lo cual funciona bien cuando la escena requiere que la construcción esté distante. En caso de que el personaje y la cámara estén cerca de la fachada de un inmueble,

entonces el modelo se volverá más complejo. Para conseguirlo se pueden generar extrusiones positivas o negativas para simular marcos de ventanas, puertas o cornisas. Estas extrusiones gozarán del beneficio de la iluminación del ambiente y los efectos de sombra y rayos de luz que se puedan producir, dando así mayor realismo cuando la cámara del jugador en la escena entra en movimiento. Cabe señalar que es necesario tener cuidado de no colocar demasiadas extrusiones, ya que de hacerlo aumentaría el conteo de polígonos, lo cual puede incrementar significativamente el consumo de recursos de procesamiento, dando como resultado una baja interacción del jugador con la escena y por lo tanto una mala experiencia. En la siguiente figura se ilustra la composición de un escenario de la ciudad con una toma lejana, donde la mayoría de los edificios están compuestos de cubos con texturas pegadas para las fachadas de los edificios.

Si bien más adelante se profundiza en el concepto de escena, en este punto hay que resaltar que si la historia requiere de partes específicas de desarrollo en interiores, estas se deberán construir como escenas independientes. Lo anterior tiene su justificación en el hecho de que al tener un modelo del escenario externo, con interiores, incurre en un gran uso innecesario de memoria, pues el personaje principal no puede estar en el interior y en el exterior a la vez.

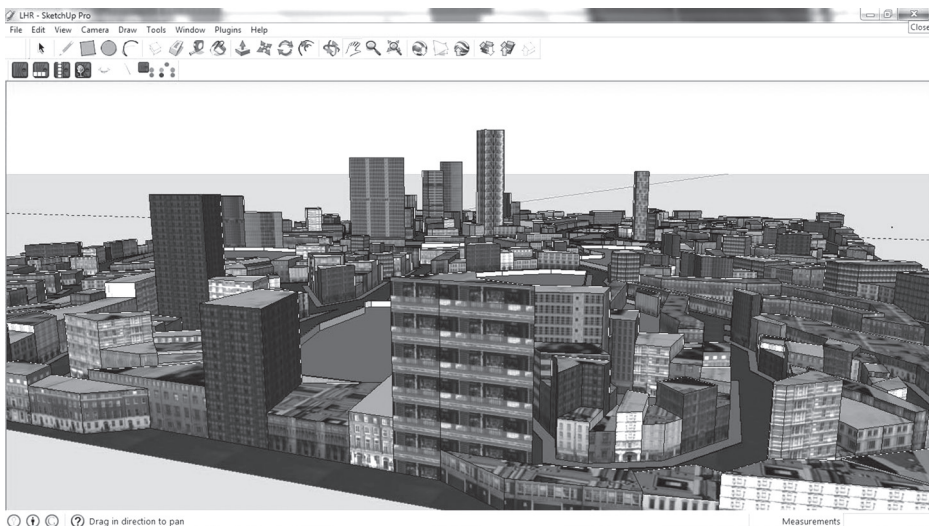


Figura 2.4. Escenario del centro de Londres modelado en SketchUp

Como se observa en la figura 2.4, la escena pasa de los edificios u objetos estáticos a los objetos menores que, en conjunto, componen el escenario; estos últimos pueden ser plantas, árboles, postes u otros elementos similares y que tal vez se repitan. Tanto los objetos como el escenario en general deben guardar co-

herencia con los parámetros de la historia, es decir, si la historia se desarrolla en el siglo x en Inglaterra, se entenderá un estilo arquitectónico medieval y, por ende, un ambiente ad hoc, lo que requerirá de una buena documentación de la composición de objetos de la época con base en la historia del juego (Clayton, 2003).

Proceso de producción de escenarios 3D

Los componentes necesarios en una escena para la elaboración de un escenario 3D son estos:

- Diseñar y detallar el croquis con base en el guión.
- Modelar los objetos.
- Crear y editar texturas.
- Texturizar los modelos.
- Acomodar los modelos 3D en el motor de videojuegos, de acuerdo con el croquis.

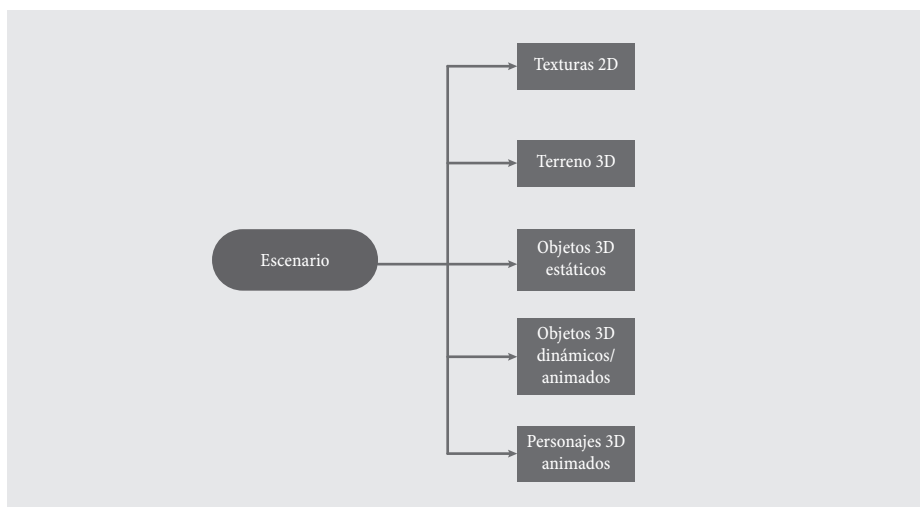


Figura 2.5. Elementos que componen una escena 3D que hay que considerar en el modelado

Croquis y elementos de la escena

Antes de proceder al modelado del escenario es fundamental plasmar en un croquis la estructura que tendrá. Esto permitirá identificar el acomodo de los distintos objetos, además de establecer el enlace entre el escenario y el guión. El escenario, por ejemplo, puede estar pensado de tal manera que el personaje principal deba recorrer forzosamente un camino; en el croquis se puede determinar

este camino y se pueden establecer cuáles objetos bloquearán la posibilidad de tomar otros (automóviles, paredes, basura, escombros, etc.). Asimismo, el croquis ayudará a reconocer si se debe configurar algún objeto de manera individual o si será parte del modelo general de la escena, de acuerdo con la función que se haya establecido en el guión para cada objeto. En caso de que hubiera una pared en el entorno, esta bloquearía un camino. Ahora bien, dicha pared podría estar integrada en el modelo general; sin embargo, si se quiere que la pared reaccione a eventos físicos de manera individual (como derrumbarse o deteriorarse después de cierto número de colisiones) sería necesario modelarla y exportarla en un archivo por separado.



Figura 2.6. Vista 3D del croquis de la escena con información detallada

En más de una ocasión se deseará replicar un lugar de la vida real, por lo que para alcanzar resultados lo más cercanos a ella, habrá que utilizar como base un mapa. Se recomienda hacer uso de Google Maps¹ en la vista de satélite, siempre que sea posible, para obtener una referencia de lo que se quiere. Para ilustrar lo anterior, la figura 2.6 muestra un mapa de una zona de monumentos históricos de la ciudad de Guadalajara desde una vista de fotografía satelital y sobre la cual se puede empezar a construir y tomar dimensiones para la escena. Lo anterior parte del principio de que algunos juegos son más interesantes cuando se toman escenarios de espacios reales y cotidianos que posteriormente pueden sufrir transformaciones para adecuarse a la historia del juego.

¹ Servicio gratuito de Google que permite al usuario acceder a los mapas de un creciente número de ciudades, además de proporcionarte su vista de satélite.

En este sentido, manejar un croquis mediante una herramienta de fotos satelitales cuando se tienen escenarios reales —en este caso Google Maps—, cuenta con muchas ventajas. La primera es que este servidor muestra la escala que está utilizando, por lo que habrá una mejor aproximación al cálculo del área que se necesitará. Incluso en las ocasiones que no se tienen a la mano las medidas de los edificios que se modelarán, esta herramienta ayudará a mejorar la proximidad y, por ende, se conseguirán mejores resultados.

Es importante apuntar que cuando se trata de modelar lugares reales se requiere emplear el sentido común, limitando el diseño a calles y avenidas principales, así como edificios y monumentos más representativos, ya que el modelado de cada rincón de un área específica es una tarea exhaustiva que involucra una gran inversión de horas de artistas y diseñadores. Apegarse al guión, a la historia y a lo general de la escena que se está modelando es lo principal. Muchos detalles pueden pasar desapercibidos si el juego tiene movimientos rápidos de la cámara, como ocurriría en un juego donde se maneja un vehículo o se genera una carrera entre personajes.

Texturas en objetos 3D

El proceso de texturizado consiste en utilizar mapas de bits en 2D. Estos se crean en programas de manipulación y creación de imágenes como Gimp² o Photoshop³ y son aplicados a la superficie del modelo. Lo anterior se hace con un proceso conocido como *uv mapping* (Pardew, 2004), en el cual se define la manera en que los mapas de textura se deben alinear en la parte exterior del modelo (Franson, 2002).

Una manera conveniente de traer realismo al escenario es creando texturas basadas en fotografías reales. Si bien se sugiere que la textura en sí no sea muy pesada, también resulta cierto que es muy probable que se requiera editar la fotografía tomada, para obtener la textura deseada y hacer que coincida con la iluminación y ambientación del escenario pretendido (Ahearn, 2001; Pardew, 2004).

La recomendación general al respecto es tomar fotografías con un tripié, con la finalidad de evitar ángulos. En muchos casos esto será un tanto difícil, pues al intentar capturar edificios es posible que la poca distancia entre la cámara y el objetivo, o la misma curvatura que se genera por la altura del edificio, produzcan fotografías con cierta inclinación angular. Para solucionar esto existen trucos que ayudarán a crear texturas que logren ser lo suficientemente realistas sin requerir

² GIMP es un *software* de uso libre con licencia que forma parte del proyecto GNU. Tiene más de una década en desarrollo continuo y funciones muy potentes similares a las de los programas profesionales. Para más información, véase www.gimp.org

³ Photoshop es el *software* bajo licencia de la empresa Adobe, siendo el más empleado por profesionales en el mundo en diseño gráfico y fotografía digital. Para más información, véase www.adobe.com

un gran esfuerzo en la edición digital; por ejemplo, si se quiere replicar un patrón de ladrillos, basta con que se tome una muestra; si se quiere igualar un color con una muestra, el programa de modelado 3D se encargará de envolver al objeto con la textura y replicarla cuantas veces sea necesario. La figura 2.7 ilustra cómo con el texturizado se genera, con una foto, un objeto 3D complejo.

Para obtener los mejores resultados, las fotografías deben tomarse con la más alta resolución posible a manera de captar una mayor cantidad de detalles; al momento de la edición de las texturas se podrá reducir el tamaño. Como experiencia personal, se estima que una resolución de ocho megapíxeles o más ofrece insumos bastante aceptables para el proceso de edición y texturizado de objetos 3D. Cabe remarcar que los archivos finales que van a usarse para texturizado llevan solo unas cuantas decenas de kilobytes en tamaño, para eficiencia de representación gráfica.

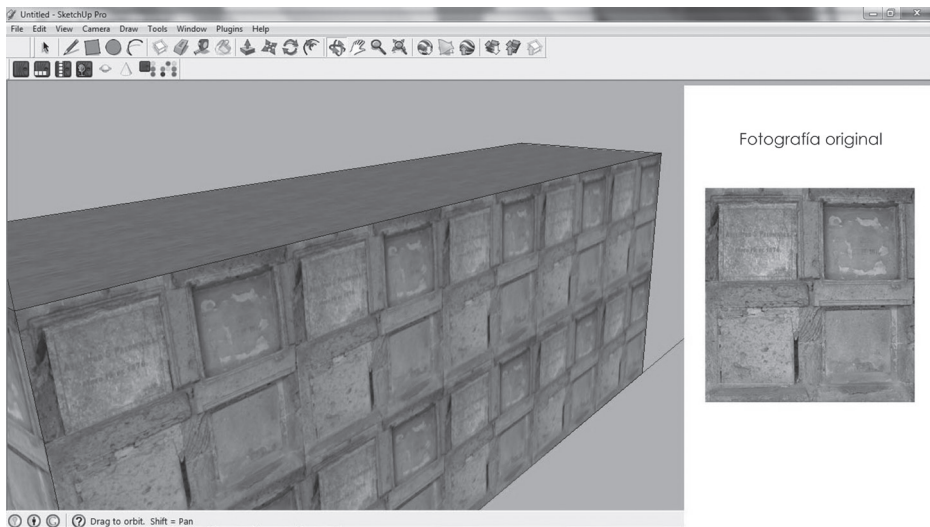


Figura 2.7. A la derecha se puede ver la fotografía que dio origen a la textura que usa este modelo

Modelado 3D

Cuando se habla de modelado 3D, la mayoría de las herramientas para generar los polígonos y trabajar las geometrías son de una alta complejidad y requieren una curva de aprendizaje lenta. Los programas adecuados para llevar a cabo este trabajo cuentan con más de dos décadas y han crecido en la cantidad de funciones que disponen. En un esquema de innovación, la empresa Google ha desarrollado una poderosa herramienta de modelado 3D que ha invertido un gran esfuerzo para que su ventaja principal resida en una curva corta de aprendizaje y que esta

resulte sumamente intuitiva para los usuarios; cuenta con una interfaz que dispone de una barra de herramientas que consta de unos cuantos botones, pero que permiten crear modelos de gran complejidad. Su nombre es SketchUp.

En relación con las funciones de este programa, hay que decir que es extremadamente útil para modelar el escenario compuesto de objetos estáticos, tales como edificios, construcciones y hasta árboles o vegetación. A pesar de su rendimiento, no sirve para modelar personajes con esqueleto u otros objetos 3D animados debido a que no cuenta actualmente con soporte para huesos o estructuras compuestas de articulaciones y modelado de movimiento cinemático. El *software* de SkecthUp incluye opciones para modelar con geolocalización y mejoras en continuo en las herramientas para integrar texturas que permitan que los objetos 3D tengan acabados foto realistas. Esta es una técnica que permite optimizar el uso de polígonos y crear una sensación de un ambiente similar al mundo real en poco tiempo para los escenarios de los videos juegos serios.

Con la idea de que SketchUp sea accesible a grandes comunidades de usuarios existe la versión gratuita que se puede descargar en la página web <http://www.sketchup.com> y que permite trabajar en el modelado de forma casi profesional. También este *software* comparado con otros cuenta con una interfaz muy intuitiva que permite una curva de aprendizaje muy rápida. Ya con experiencia en el uso de SketchUp se puede buscar adquirir la licencia profesional que tiene descuentos importantes para estudiantes, profesores y organizaciones sin fines de lucro. La licencia profesional de SketchUp tiene un precio en el mercado altamente competitivo y una interfaz como se mencionó altamente intuitiva y un excelente soporte y mejoras desde ya casi una década de existencia comparando con la competencia como es Autodesk. La principal diferencia entre la versión sin costo y la de paga es la posibilidad de importar o exportar a distintos formatos de objetos 3D haciendo la herramienta compatible con el *software* comercial. La versión gratuita solo puede leer o grabar archivos en un formato propietario de Google que emplea para su *software* Google Earth⁴ con la idea de que una gran cantidad de usuarios aporten modelado 3D de forma colaborativa. Entre los formatos que se pueden exportar con la versión profesional o de paga de SketchUp se encuentra el FBX,⁵ con la terminación *.fbx en sus archivos. Dicho formato trabaja bastante bien con Unity 3D, mas no significa que deba comprar la licencia de este *software* para que sus modelos puedan ser usados en su proyecto

⁴ *Software* que sirve para ver el planeta Tierra y mapas con fotografías satelitales y relieve de terrenos, así como algunas ciudades en 3D. Para más información, véase <http://www.googleearth.com>

⁵ Formato de intercambio de archivos 3D creado por Autodesk. Para más información, véase <http://usa.autodesk.com>

3D.⁶ Cabe mencionar que en cada versión nueva de SketchUp se va aumentando la compatibilidad de archivos para que el *software* pueda interactuar con otras plataformas de modelado 3D.

Asimismo, SketchUp tiene la posibilidad de exportar en formato COLLADA,⁷ que tiene una extensión *.dae; sin embargo, no se considera un formato que funcione del todo apropiado. En estos casos, la alternativa para mantener un bajo costo de producción sacrificando un poco de funcionalidad de SketchUp es utilizar Blender⁸ como puente para importar y exportar modelos Unity 3D. Esto permite tomar ventaja de la facilidad de uso de SketchUp y aprovechar la versatilidad de Blender, ya que cuenta con una gran madurez de más de dos décadas en desarrollo. El proceso consiste en exportar el modelo creado en SketchUp bajo un formato COLLADA. Posteriormente, se ejecuta Blender y se importa el archivo *.dae, el cual se guarda en el formato nativo *.blend. El formato anterior es soportado sin problemas en Unity 3D.⁹

Al igual que otros paquetes de modelado 3D, SketchUp cuenta con la posibilidad de crear grupos de objetos (Evry, 2004), lo cual es de gran beneficio puesto que permite determinar una entidad independiente a la que se le pueden atribuir propiedades específicas dentro del modelo general. Por ejemplo, si estuviésemos modelando el escenario para un ambiente en interiores, se puede hacer que los objetos que tendrán un comportamiento dinámico, como las puertas al abrirse o cerrarse, se conviertan en grupos. Lo anterior presenta la ventaja de que la puerta se encontrará perfectamente posicionada dentro del modelo. De la misma manera, los objetos con comportamiento dinámico, tales como mesas o sillas (con la alternativa de ser movidas por el personaje, golpeadas, etc.), podrían convertirse en grupos y de esa manera manipularlos.

Un detalle importante que hay que mencionar del formato Blender, son los inconvenientes que tiene. Cuando un modelo se exporta en FBX, el sistema de Unity3D reconoce estos grupos; no obstante, al utilizar el puente de Blender existe un comportamiento más inestable, pues si bien las texturas y el modelo se conservan, los grupos y las texturas de estos pueden sufrir alteraciones. En síntesis, este es el sacrificio que se puede presentar con Blender como puente para llegar a Unity 3D, que no es recurrente pero sí puede consumir, en algunos casos, tiempo en ajustes de los modelos.

⁶ Para más información y descarga del programa, véase <http://sketchup.google.com/intl/es/index.html>

⁷ Formato libre que busca la estandarización en lo referente intercambio de archivos 3D. Para más información, véase <http://www.khronos.org>

⁸ Blender es un *software* de modelado 3D de código abierto, de una alta complejidad. Para más información, véase <http://www.blender.org>

⁹ Para obtener información específica de los formatos que se mencionan, véase el apartado de formatos de importación de objetos 3D.

La figura 2.8 presenta los diferentes procesos y pasos a seguir en el modelado de objetos 3D estáticos.

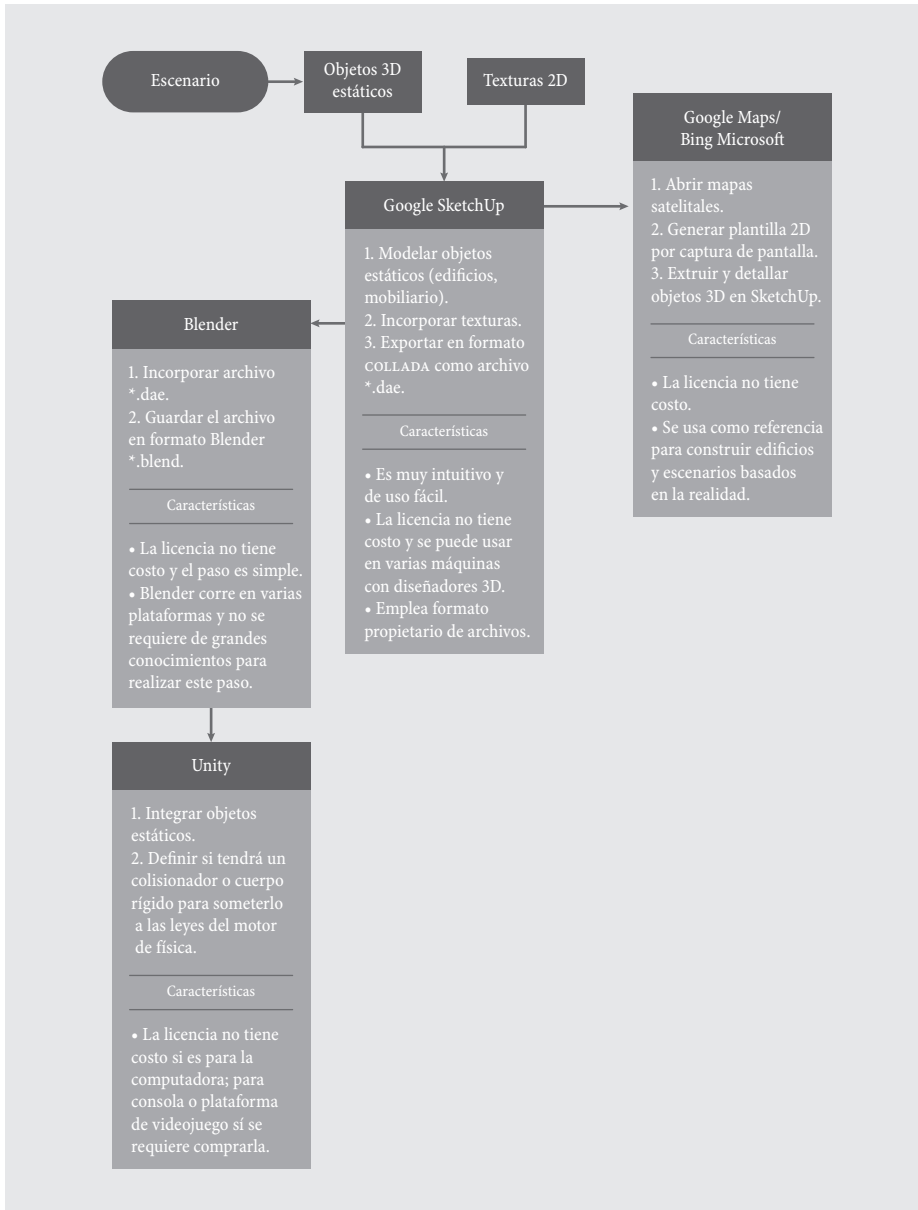


Figura 2.8. Procesos identificados para modelar objetos 3D mediante el *software* de SketchUp, el puente con Blender y su exportación para uso en Unity 3D

Principios básicos

SketchUp se basa en la premisa de que un modelo complejo inicia con una geometría básica. Estos simples polígonos se van refinando hasta obtener patrones más detallados, ya sea porque junto con otros polígonos forman un objeto más grande o porque se subdividen para esculpir su forma final (Cunningham, 2006). En principio, casi todo objeto de la vida real se puede representar mediante figuras geométricas simples, incluso se puede reproducir el cuerpo humano con el uso de esferas y cilindros. Al iniciar el modelado es recomendable trabajar de lo general a lo particular. Es decir, todo gran modelo inicia con cuadrados, cilindros, conos, esferas, etc., y posteriormente estos objetos se van detallando, ya sea con texturas fotorealistas o con un mayor número de polígonos trabajados sobre la geometría de base (Evry, 2004). La figura 2.9 ejemplifica este principio de construcción de la escena con objetos de geometría básica.

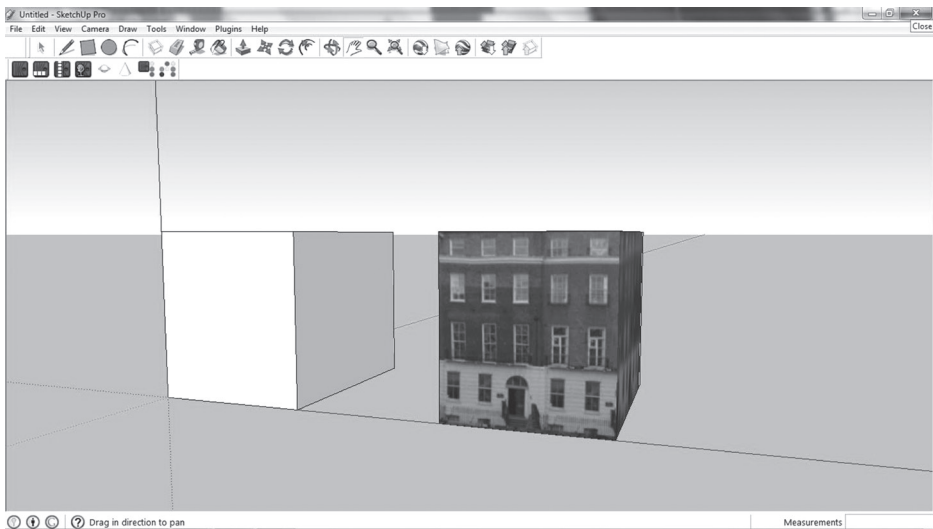


Figura 2.9. Una geometría básica convertida con una textura fotorrealista en la fachada de un edificio para constituir una ciudad

Como se observa en la figura 2.10, cuando se trabaja con formas simples, el modelo resultante queda como un dibujo sin colorear y con poco impacto. Por lo tanto, para aumentar el realismo, lo que procede es agregar el proceso de texturizado con base en fotografías reales o diseños gráficos trabajados por un artista. Este es un toque que dará presencia al modelo y se discutirá más adelante. Adicionalmente, es muy recomendable que todo modelo que se haga en SketchUp se centre en el origen, lo que significa que hay que colocarlo donde convergen los

ejes x , y y z , pues a la hora de importarlo en Unity 3D el modelo se manipulará a partir de este centro.

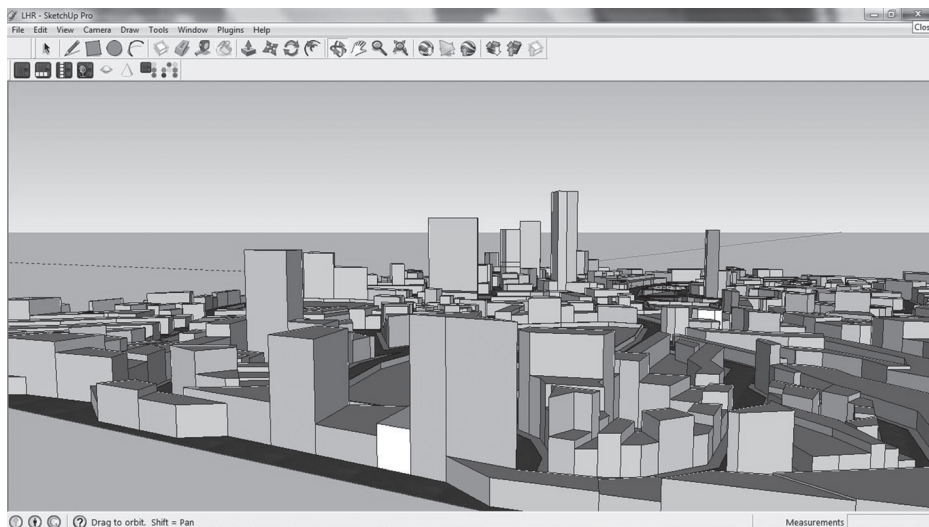


Figura 2.10. Escenario del centro de Londres sin texturizar

Optimización de recursos para videojuegos

Una clave en el modelado 3D para videojuegos es identificar el punto exacto en el cual el modelo debe detener su complejidad de polígonos y trasladarla al proceso de texturizado. Esto tiene su fundamento en el hecho de que la cantidad de polígonos que una tarjeta gráfica puede soportar es limitada; un árbol puede estar construido con miles de polígonos, y si lo que se busca es representar la escena de un bosque, esto puede llevar a millones de polígonos, lo cual no es eficiente para el cálculo que requiere hacer el procesador de la tarjeta gráfica y la memoria para la escena en visión. A continuación se citan algunas técnicas de optimización que se deben tomar en cuenta en el proceso de modelado 3D.

Reducción de polígonos. Es posible que el proceso de modelado sea tan absorbente que dé como resultado modelos demasiado complejos y estéticos para un proyecto. Para esos casos, muchos paquetes de modelado 3D cuentan con una herramienta que reduce la complejidad del patrón en términos de la cantidad de caras del modelo poligonal y, por ende, la cuenta final de polígonos. En el caso concreto de Blender, el modificador Decimate genera esta reducción, de manera que se puede controlar la calidad de los modelos 3D. Para esclarecer esta función, la figura 2.11 muestra un objeto con alto conteo poligonal, así como el resultado

de optimización y reducción cuando se ha aplicado la función. Hay que remarcar que lo anterior es solamente el punto de partida en la optimización, pues existen otras técnicas que se ejecutan principalmente en el motor de videojuegos (Unity 3D) (Luebke *et al.*, 2003). También es necesario indicar que al reducir el número de polígonos, el objeto a optimizar puede tener vértices que no queden bien conectados y que distorsionen drásticamente la forma del objeto en desarrollo, por lo que se requiere una inversión de tiempo para poder reducir el conteo de polígonos y tener un modelo similar al inicial.

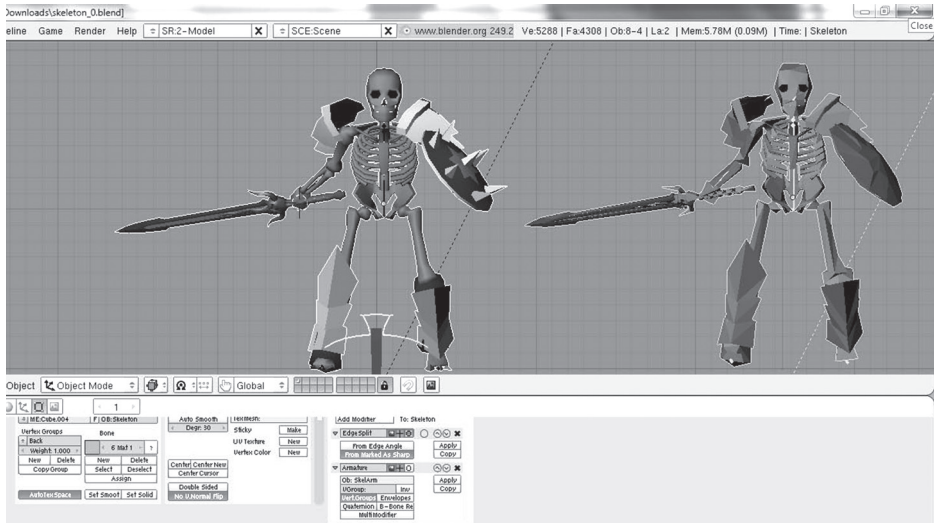


Figura 2.11. Diferencia entre un modelo detallado (izquierda) y uno con la versión de la función Decimate al que se le han reducido polígonos (derecha)

Objetos. La mejor manera de obtener un mejor desempeño al representar los gráficos en un videojuego es combinar objetos para que cada grupo de polígonos tenga aproximadamente 1,500 o más y emplear un solo material. La noción de material corresponde a un parámetro que se mete al momento de generar el modelo sobre el color y según el tipo de este, será el efecto de luz; por ejemplo, si el objeto va a ser opaco, absorbe la luz; si va a ser transparente, la deja pasar; o si es reflectante como un metal, la refleja. Por su parte, es posible definir el material para todo el objeto o para varias caras del mismo, pero lo óptimo es tratarlo con uno solo. La documentación de Unity señala que los objetos que tengan aproximadamente 1,500 polígonos requieren el mismo esfuerzo de procesamiento que los objetos de 100 polígonos (Blackman, 2013).

Texturas ligeras. El trabajo de las tarjetas gráficas se puede disminuir al reducir el número de texturas que se incluyen en los diferentes objetos y sus caras, todo

ello a través de mapas de texturas o *uv maps*. Lo que hace este proceso es enlazar ciertos píxeles de la imagen con alguna superficie del modelo 3D, de forma que el modelo se compone de una sola textura, aunque esta no sea necesariamente de un color uniforme. Si lo anterior se quisiera comparar con algo, lo más parecido sería lo equivalente a quitarle la cáscara a una naranja y luego extenderla de forma plana. Si bien la cáscara toma una forma extraña, lo cierto es que al envolver la naranja con ella, esta se amoldará perfectamente.

Ya anteriormente se hizo mención de que las fotografías deben tomarse con la más alta resolución disponible; no obstante, para evitar saturar un proyecto con archivos cuyo peso o detalles son innecesarios, es importante reducir el tamaño de las fotografías ya editadas y guardarlas en un repositorio. Un tamaño ideal es aquel que permite que la textura no se vea burda y con cuadros, efecto que se denomina técnicamente como *pixeleado*. Así pues, este es un proceso de prueba y error que se basa en la experiencia del diseñador. Como regla general, siempre es recomendable dejar intactas las fotografías originales y trabajar con copias. Casi todos los programas de edición de imágenes cuentan con la función de duplicar el archivo. En este sentido, si se comete algún error, siempre se podrá regresar en el tiempo y reparar el daño. Incluso se recomienda trabajar con copias de la textura final, ya editada la misma, debido a que el diseñador seguramente tendrá que manejar distintas versiones que se adapten a las diferentes condiciones de iluminación requeridas con la escena.

Nodos Billboard. Esta técnica es especialmente útil cuando existen objetos complementarios que añaden realismo a la escena, pero no requieren una cantidad abundante de detalles. Por ejemplo, se puede pensar en árboles o edificios de fondo como objetos que caractericen la escena principal. Sin embargo, ya que el personaje principal no estará lo bastante cerca para notar sus detalles, no es necesario que se tengan modelos 3D pormenorizados. Es entonces que se usan fotografías realistas recortadas que se ponen en varios planos para dar la impresión de profundidad, tal como se hace en los escenarios de teatro o de cine.

Dicho proceso consiste en introducir a la escena objetos 2D recortados con la forma de árboles, plantas, postes o edificios, por citar algunos casos. Estos se pegan como textura a un objeto plano rectangular transparente y están programados para siempre mirar de frente a la cámara. Dependiendo del motor de los videojuegos, se puede personalizar el comportamiento de estos nodos para que se vea como nosotros deseamos cuando pasa la cámara. En el caso de los árboles u otros objetos con los que el personaje principal eventualmente podría tener contacto, es posible contar con un código que detecte la proximidad del objeto en cuestión. Si se cumple un rango de cercanía de la cámara, entonces el código sustituye el objeto 2D por uno 3D, con detalles que se puedan admirar, y en cuanto se aleja el objeto principal vuelve a ser 2D. Como último punto, se busca mantener

el realismo con varios trucos visuales para el usuario final sin saturar la memoria y las capacidades de procesamiento. En la siguiente figura se presenta un árbol recortado que se puede introducir como nodo Billboard.

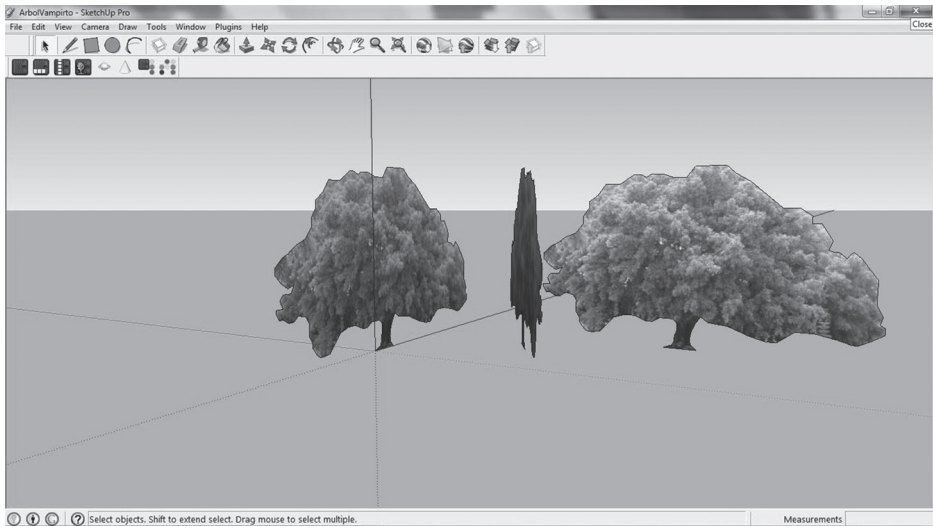


Figura 2.12. Objeto árbol 2D pensado para ser un nodo Billboard sin profundidad

Instancias. Se refieren a copias de objetos que necesitan ser replicados. En el caso de una escena ambientada en un restaurante, se requerirá de varias mesas y sillas que estructuralmente son iguales. Lo que se hace aquí es solo modelar una mesa y una silla; una vez que está en motor de videojuegos se pueden crear copias (instancias) de esas sillas y mesas y colocarlas en la escena. Al hacer esto se ahorra una gran cantidad de memoria de la computadora y un manejo más eficiente en términos de la representación gráfica (Luebke *et al.*, 2003).

Combinación de texturas. Con ello se pretende reducir la cantidad de texturas al integrar en un solo archivo todas las que se necesitarán en el modelo. Llevar a cabo esta acción hace que la dificultad del trabajo se encuentre en posicionar adecuadamente cada parte de la textura. Para llevar a cabo esto se requiere de habilidad en el manejo de la herramienta de texturizado, debido a que, por se un trabajo muy artístico, se trabaja con un solo archivo a pegar en una de las caras del modelo y puede consumir bastante tiempo (Luebke *et al.*, 2003).

Texturizado con GIMP

Como ya se habló de la importancia que tienen las texturas en el impacto y realismo de un videojuego, ahora toca el turno de explicar cómo llegar a esas texturas finales que adornarán su escenario. GIMP es una potente herramienta para edición de imágenes, la cual también es producto del *software* libre con más de dos décadas de desarrollo y evolución. Su interfaz en realidad no es tan simple, pero es similar a la de su principal competidor, Photoshop de la empresa Adobe, que requiere una licencia que puede llegar a costar por encima los \$800 dólares (ssc, 2002).

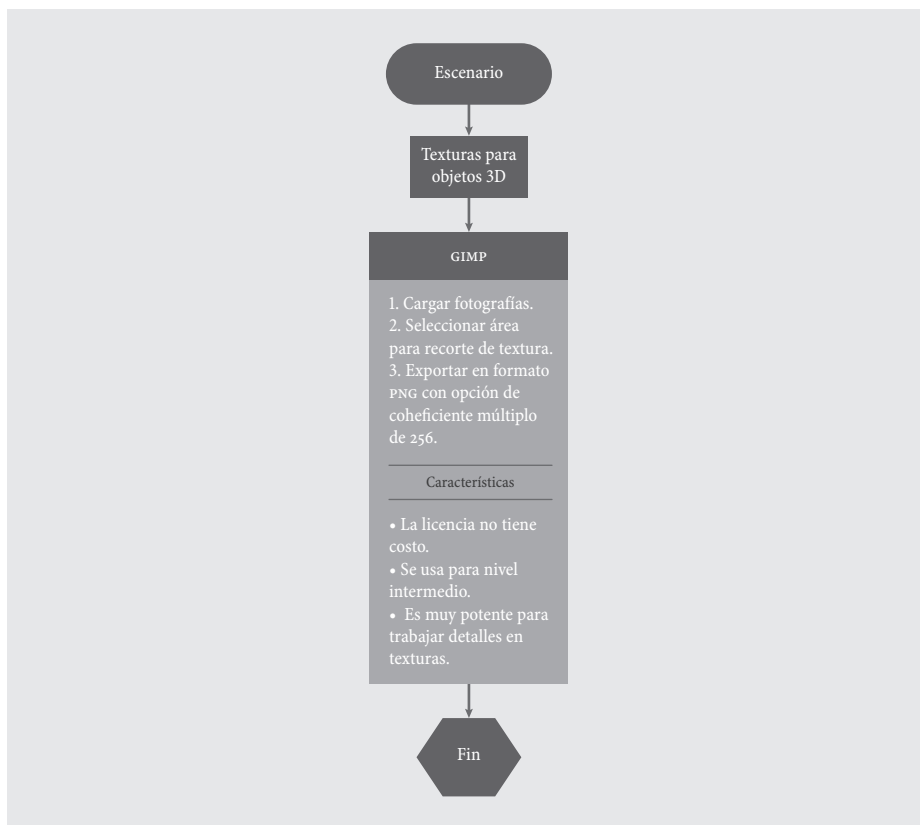


Figura 2.13. Procesos a seguir para el modelado de texturas con GIMP

Principios básicos

Durante el proceso de edición de imágenes para el modelado de texturas se puede emplear el *software* que se quiera. Si bien la propuesta de esta obra está enfocada

al *software* libre o de muy bajo costo, con la finalidad de hacer más accesible la cadena de producción de videojuegos, es importante decir que una licencia de Photoshop tiene otras capacidades y soporte, y además es más intuitiva. Asimismo, para trabajar con una licencia sin costo, se estima que GIMP ofrece una buena cantidad de funciones que ayudan a la edición de videojuegos. En la figura 2.14 se expone la interfaz y la barra de herramientas que se dispone en este *software* para trabajar una foto que se convertirá a textura.

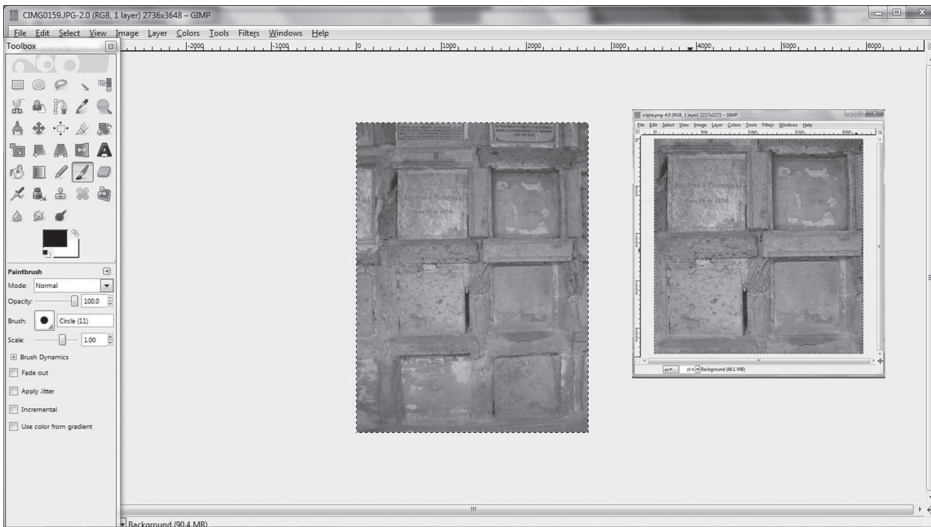


Figura 2.14. Texturizado con GIMP

En esta parte de la cadena de producción, las fotografías que se convertirán en las texturas finales de los modelos ya deben de estar tomadas. Muy probablemente será necesario realizar al menos alguna de las siguientes acciones: recortar, redimensionar, igualar color, rotar, crear efectos de relieve, borrar algunas partes de la imagen, crear fotomontajes, etc. A continuación se explican las acciones más comunes que se deben llevar a cabo en el proceso de texturizado (Ahearn, 2001).

Texturas en potencia de 2. Una buena recomendación en este punto es la de siempre trabajar con texturas en potencia de dos. Esto es, la resolución de la imagen (ancho y alto) debe ser 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, etcétera, de tamaño en píxeles, y GIMP o Photoshop pueden ayudar a calcular este tamaño. Lo anterior se justifica porque las tarjetas gráficas convierten la resolución de la imagen a potencia de dos de cualquier manera, así que lo que se hace es ahorrarle este paso. No obstante, esto no solamente es un tema de optimización, ya que también puede ser un asunto de requerimientos, puesto que muchos programas

de 3D solamente aceptan texturas en esta potencia. En el caso de Unity 3D, es así cuando se requiere el empleo de una textura para el terreno.

Creación de relieves en 2D. Un relieve ayuda a marcar la diferencia en profundidad de dos o más superficies. Esta técnica crea texturas más realistas y el impacto que genera significa un ahorro que se refleja en un menor número de polígonos de las escenas (Pardew, 2004). Por ejemplo, para lograr un efecto de la diferencia en relieve entre el cemento y los ladrillos de una pared, se separa la imagen en dos capas, de manera que en una se colocan todos los ladrillos y en la otra se pone el cemento. Al hacer un procesamiento especial con la función de *bump map*¹⁰ sobre estas dos capas y el archivo de la imagen, el resultado que se consigue cuando esta textura se pega sobre un modelo 3D es que, con el paso de la luz, los ladrillos dan una impresión de estar en relieve. El efecto final sería parecido al de la figura 2.15, donde se aprecia el detallado de la textura frente a la iluminación del ambiente en el pasillo.

Bump mapping. Esta técnica es muy parecida a la creación de relieves en 2D, pero con la diferencia de que aquí los niveles que dan la ilusión de relieves sobre la textura se crean de manera desordenada, lo que resulta en un efecto de corrugado. En este sentido se facilita la generación de efectos más realistas cuando las luces de la escena 3D se proyectan sobre las texturas tratados con *bump maps*. Por citar un caso, el terminado irregular de una pared con acabado de cantera puede lograrse utilizando *bump mapping* en lugar de crear los relieves en 3D. De esta manera se disminuye de forma eficiente la cuenta de polígonos sobre el modelo 3D en que se aplica la textura y aumenta el nivel de realismo de la escena completa (Zerbst, 2004).

Alineación de texturas. El manejo de texturas fotorealistas implica que su origen viene de fotografías reales, las cuales deben tener una correcta alineación para poder ser pegadas al objeto 3D. Sin embargo, lograr la alineación perfecta en una toma fotográfica no siempre es posible, ya que puede haber distorsiones por la perspectiva de la imagen y los problemas de escalamiento. Es por esta razón que se vuelve necesario saber cómo alinear manualmente la imagen de la fotografía tomada para emplearla como textura. Una herramienta con la que cuentan la mayoría de los programas de edición de imágenes que sirve para esto es la de *rotación*. Por lo general basta con hacer rotaciones pequeñas de un grado o dos, en contra o a favor de las manecillas del reloj. En ocasiones se dará el caso de que la textura se logre alinear perfectamente de un lado y no corresponda del otro.

¹⁰ Esta función se aplica desde las herramientas de diseño gráfico, tanto con GIMP como con otro *software* similar.

Una estrategia para lograr una alineación perfecta es crear una nueva capa donde se coloque una copia de la imagen y después se rote horizontalmente esa capa. Con esto se tendrá una textura simétrica.

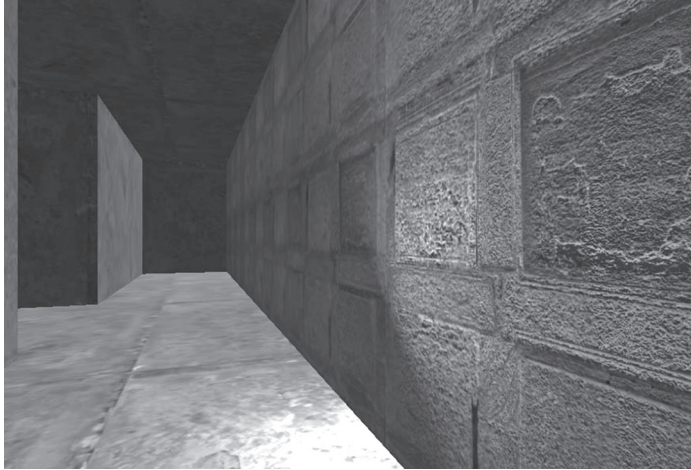


Figura 2.15. Resultado del proceso de *bump mapping*

Homogeneizar colores. Es muy probable que después de tomar las fotografías de base para las texturas que se van a aplicar, estas no tengan una gama de colores, contrastes y brillo uniformes, vinculado con las distintas posiciones de la cámara al capturar la imagen o por el ángulo en que se encuentra el objeto fotografiado en relación con alguna fuente de luz. De hecho, cuando se está fotografiando en exteriores, sucede frecuentemente que la tonalidad del color sale muy variada debido al ángulo con el que la luz se refleja en el objeto. Asimismo, hay otros factores que pueden afectar la tonalidad, como pueden ser nubes u otros objetos que provoquen sombras en la superficie del objeto capturado, lo que genera problemas al pegar la foto como textura de una superficie 3D.

Por lo tanto, si se quiere hacer una textura con colores homogéneos, en casi todos los programas de edición de imágenes se cuenta con una herramienta para igualar tonalidades de secciones de textura. Simples o complicados, estos métodos deben arrojar como producto un conjunto de texturas estandarizadas con el mismo tono y gama de color. Por ejemplo, imagínese que se obtienen fotografías de un edificio, cuyo color principal de una fachada de cantera en ocasiones aparece azul-verdoso y en otras grisáceo por la iluminación irregular. Lo anterior se explica de acuerdo con las diferencias de hora del día en que son tomadas las fotografías de la fachada; con mucho sol el brillo del mismo sobre las piedras da tonos azul-verdosos, mientras que con poco sol serán grisáceas. Resta decir que el tono uniforme de la textura puede ser modificado intencionalmente para adap-

tarse al guión del videojuego para generar un ambiente, transmitir una emoción o delimitar una época, por lo que, entonces, no solo es útil homogeneizar colores de texturas para corrección de las mismas, sino también como parte del desarrollo ambientado requerido en la escena según lo establece el guión.

Integración de la escena en Unity

Una vez hecho el modelado y texturizado de todos los objetos que se establecieron en el guión, es el momento para integrarlos en la plataforma Unity. Cabe señalar que en esta parte solamente se tomarán en cuenta los modelos estáticos, pues los dinámicos (avatares u otros objetos con animaciones) deben considerar otros factores que incluyen programación para generarles una dinámica, así como su asignación de sensores para la gestión de eventos y colisiones de los mismos en el ambiente del videojuego.

Lo que sigue, pues, es que en un motor de videojuegos se integran los elementos (modelos estáticos, modelos dinámicos, sonidos, etc.). Se trata de un programa que permite definir la interacción que los objetos de la escena tendrán con base en una serie de reglas dadas por el guión. El motor de videojuegos de Unity trabaja con la noción de proyectos que agrupa el conjunto de escenas que componen el videojuego. Una escena puede ser tan simple como la pantalla de inicio del videojuego con texto y fotografías o tan completa como la escena con objetos 3D texturizados y animados del videojuego.

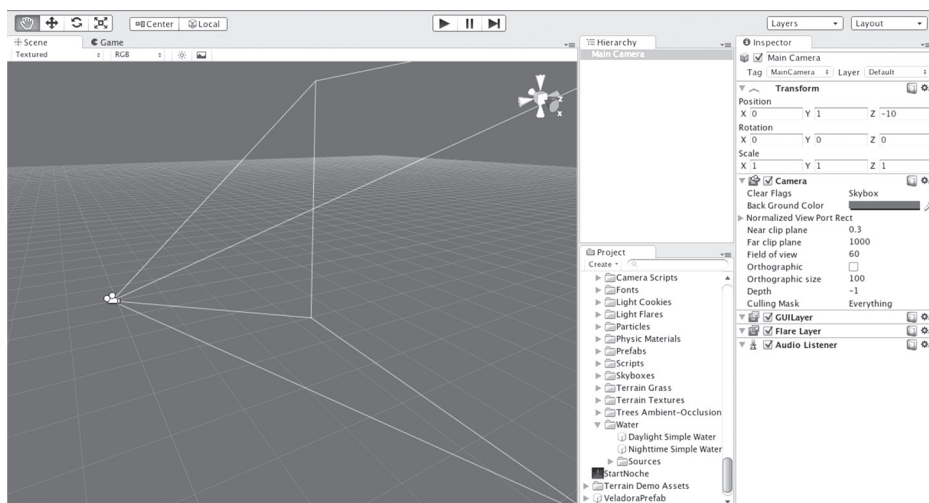


Figura 2.16. Interfaz de Unity. El único objeto en la escena representa a la cámara; el cuadrado que aparece muestra la parte del mundo virtual que será escenificada

Los proyectos que opera Unity están compuestos de carpetas que contienen los elementos del juego (*prefabs* o modelos que son copias de un objeto 3D que se repetirá, *scripts* que son pequeños códigos para ejecutar un comportamiento o animación de objetos, etc.). Dado que el ambiente de Unity es gráfico, la forma de importar algún elemento es verdaderamente fácil, ya que solo hay que abrir la carpeta de dicho proyecto y arrastrarlo mediante el apuntador; Unity automáticamente lo detecta y lo carga en el programa agilizando los procesos de producción de la integración y dinámica del videojuego. Todo elemento que se agrega a la escena es un objeto del juego (*game object*) al cual se le pueden agregar componentes (*components*), ya sea de masa, partículas, física, audio, renderizado, misceláneos, *scripts* o control de cámara). En esta parte se pueden ver los *scripts* que se programan, los cuales tienen la opción de volverse componentes que se pueden agregar a cualquier objeto del juego. Por último, todos los objetos de juego cuentan con una propiedad llamada *transform*, que incluye la posición del objeto en el espacio virtual, la rotación del objeto que se refiere a la orientación que guarda y la escala. Estas tres variables van en función de las coordenadas *x*, *y* y *z*, dado que se está hablando de un espacio cartesiano en tres dimensiones, en donde *x* es el eje horizontal, *y* es el eje vertical y *z* es el eje de profundidad.

Formatos de importación de objetos 3D

El proceso de importación de objetos 3D es simple, pero la selección del formato que se va a usar puede resultar una empresa compleja debido a la diversidad de formatos existentes no estandarizados, por lo que está la posibilidad de que se convierta en el cuello de botella de la cadena de producción de videojuegos.

Por un lado, algunos de estos formatos han sido creados pensando en un *software* específico, que al estar tan bien posicionado, el formato adquiere relevancia. Tal es el caso de la empresa Autodesk con sus formatos de archivos *.dwg, *.dxf o *.max. Además, como un esfuerzo de lograr un estándar convergente de formatos de archivos, esta misma firma tiene el formato FBX. Por otro lado, existen más formatos de otras empresas y asociaciones que promueven desarrollos no propietarios y que fueron los pioneros del intercambio de archivos 3D, como es el caso de los formatos *.dae, *.obj, *.vrmf; o aquellos que son un tanto desconocidos, como *.xsi, *.lwo; o un tanto específicos, como *.bvh, que sirve para capturar animaciones (Pipho y Lamothe, 2002; Chen, 2002).

Con casi cuatro décadas de desarrollo tecnológico en el mundo de los formatos de archivos 3D, este se ha vuelto un tanto complejo. Con el paso del tiempo se han ido agregando formatos que en su época estaban ligados a una arquitectura específica de *hardware* combinado con el sistema operativo de las computadoras, y hoy en día se siguen manteniendo. Aunado a ello, muchos formatos siguen evolucionando y disponen de varias versiones; algunas pueden ser soportadas

por el motor de videojuegos y otras no (Petridis *et al.*, 2010). En este sentido, la información que se proporciona por parte de los motores de videojuegos no siempre es muy detallada. También es necesario tener en cuenta que la propuesta aquí presente resulta válida al tiempo que se escribe esta sección.

De acuerdo con la documentación (*Unity Manual*), Unity (2015) puede leer FBX, DAE, 3Ds, DXF, OBJ, aunque importa nativamente los archivos de Maya (.mb o .ma), Cinema 4D (.c4d), 3Ds Max (.max), Cheetah3D (.jas) y Blender (.blend). Esta propuesta presenta soluciones de código abierto que aumentan la complejidad de soporte, ya que Unity tiene un mayor grado de integración y soporte con sistemas propietarios como lo es 3DMax y el formato FBX. Así pues, es en igualdad de términos el soporte que se le puede dar a un archivo de imágenes trabajado con Photoshop —*software* propietario de la empresa Adobe— contra el formato de imágenes que soporta GIMP —iniciativa abierta con licencia de uso libre—.

En consecuencia, determinando que existe un sinnúmero de formatos 3D, la industria está buscando converger a los estándares DAE y FBX; sin embargo, hay que señalar que el primero tiene más soporte entre las aplicaciones de *software* libre, mientras que el segundo está más enfocado a las aplicaciones por las que se tiene que pagar una licencia. Es por ello que, a pesar de que otros formatos puedan funcionar, se recomienda emplear principalmente estos dos formatos.

Una opción libre pero que se puede mejorar con un costo mínimo de la licencia original es la de SketchUp. Si se compara con los costos de licencia de los *software* profesionales de las firmas de la competencia, esta resulta la mejor alternativa. Entre sus funciones permite exportar los modelos 3D en formato FBX, lo cual facilita la cadena de producción para su integración con Unity, ya que al colocar los archivos de los modelos creados en la carpeta de los proyectos, Unity los detectará automáticamente. Lo primero que hay que hacer es seleccionar el modelo; a la izquierda (en el *Inspector*) aparecerá el importador FBX. Es importante asegurarse de seleccionar la opción *generate colliders* para que el modelo pueda detectar colisiones una vez en la escena. La escala de los modelos FBX está predeterminada a 0.1, pero se puede modificar este valor, ya sea directamente en el importador de Unity o en la propiedad *scale* del objeto en la escena. La diferencia estriba en que al modificar la escala en el importador afecta directamente al objeto, mientras que modificarla en la propiedad mencionada afecta a esa instancia (copia) del modelo en la escena en concreto que se esté editando.

En caso de no contar con la licencia original de Google SketchUp, se pueden exportar los archivos en formato DAE, después abrir el *software* de Blender e importar el archivo para después guardarlo como un archivo de Blender, es decir, en formato *.blend, el cual será reconocido automáticamente por Unity a través de su FBX *importer* (aunque en realidad no sea un archivo FBX). Este punto es de suma relevancia, dado que a pesar de que el *software* de Blender cuenta con soporte para exportar a un gran número de formatos, por las versiones que maneja

y opciones que ofrece deja archivos que no están bien integrados en el estándar (como aquellos en formato DAE) y que Unity no logra cargar correctamente.

Modelado de superficies de terrenos en Unity

En la escena de un videojuego no siempre se requerirá de un terreno irregular. Por ejemplo, en un videojuego donde la historia está centrada en el interior de un edificio, es muy probable que termine empleando solo el modelo 3D de la construcción y decorado interior de la misma. Sin embargo, para escenas exteriores realistas se debe construir un terreno que pueda incluso ser irregular. Unity, al igual que otros motores de videojuegos, cuenta con una herramienta para este fin; dispone de un menú exclusivo para el terreno que permite manipular parámetros específicos, como la resolución o el tamaño del mismo según sus necesidades.

Un terreno implica la generación de una malla de polígonos que formen un plano sin colores. Para modificar la malla y hacer la superficie irregular se dispone de un botón (el cual puede ser encontrado en el *Inspector*) que permite hundir o levantar la malla de forma intuitiva, creando relieves que hacen más realista la escena exterior. Cabe señalar que la creación del terreno también es un proceso artístico que requiere de la sensibilidad adecuada para identificar la cantidad de relieves requeridos, así como la textura idónea. Adicionalmente, en el *Inspector* también se localiza un comando para texturizar el terreno. La primera vez que se seleccione una textura, Unity coloreará automáticamente todo el terreno; no obstante, hay más herramientas bajo la forma de distintas brochas para generar diferentes acabados de texturas y crear un ambiente más realista. Así, puede haber zonas del terreno con tierra, vegetación o formaciones rocosas.

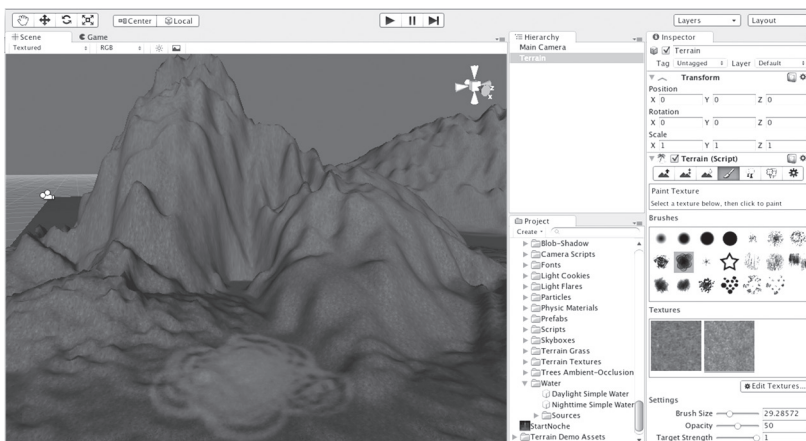


Figura 2.17. Ejemplo de modelado del terreno. En el *Inspector* se muestran las herramientas que modifican el terreno

En la plataforma de Unity se ofrece la posibilidad de detallar el escenario con árboles o arbustos que ya se tienen precargados y trabajados artísticamente con alto realismo, ya que tienen la peculiaridad de contar con animación de movimiento simulando el paso de corrientes de aire. Solo basta buscar en el *Inspector* y seleccionar el objeto *terreno*. De la misma forma, un desarrollador puede utilizar sus propias texturas o incluso sus propios modelos, siempre y cuando cumplan con las condiciones del *software*, entre las cuales destaca que las texturas deben ser potencia de dos.

Pruebas del escenario

Llegada esta parte de la cadena de producción, donde ya se cuenta con un terreno para exteriores y modelado 3D con texturizado para completar la escena, se requiere hacer la integración de todo dentro de la plataforma de videojuegos. También, dentro de la plataforma, se deberán ajustar los diferentes tipos de iluminación, efectos sonoros y preparar la programación de códigos *scripts* para la animación de los objetos 3D. Para llevar estos objetos a la escena basta con arrastrarlos de la vista de *proyecto* hacia la vista de *escena* en el plano cartesiano tridimensional. El modelo aparecerá igual que como se creó en el programa de modelado empleado. Ahora bien, si el modelo está centrado, este podrá manejarse sin problemas. En caso de no ser así, se observarán comportamientos inestables cuando se intenten hacer manipulaciones de transformación de escala, rotación o movimiento.

En relación con lo anterior, una manera para resolver problemas de objetos que aparecen no centrados es crear un *objeto de juego vacío* exactamente en el lugar donde está el modelo no centrado y en la vista de *jerarquía* hacer a este objeto vacío padre del objeto mal orientado. Aun así, es mejor atender la recomendación de modelar en el origen desde el *software* de modelado.

Nuevamente en la vista de *escena* del ambiente de Unity, es posible que el objeto, al ser incorporado, no aparezca en el terreno en la posición que se quiere. Para poder alinearlos sobre la superficie deseada, solo basta con manipular los parámetros de su nodo *transform* en el panel derecho de propiedades de los objetos del ambiente de edición de Unity. Es aquí donde esta plataforma se vuelve fácil de usar, dado que proporciona una vista previa en tiempo real de la construcción de la escena, e incluso permite manejar los parámetros del nodo *transform* de forma gráfica. En este sentido, se cuenta con la capacidad de detectar gráficamente el punto exacto en el que el modelo se desea colocar de acuerdo con el guión desarrollado. El nodo *transform* se puede identificar por las pequeñas flechas de colores que representan el plano cartesiano al momento de seleccionar el objeto, las cuales aparecen en el plano 3D a la par del panel con datos del lado del ambiente de Unity.

Manejo de cámaras en la escena 3D. Una nueva escena creada en Unity viene por defecto cargada con una cámara llamada *main camera* o cámara principal. La lógica del editor de Unity asocia esta cámara como un *objeto de juego*, por lo que también tiene la propiedad y nodo *transform* con los parámetros de base para su manipulación. Si no se tomaron previsiones para emplazar la cámara con un ángulo y vista específica, no se verá la escena en construcción la primera vez que se presione el botón *play* en el panel superior. Al utilizar las vistas 3D de Unity se podrá mover la cámara y corregir los ángulos de visión para que no quede muy abajo, muy arriba o muy alejada de la escena en construcción. Por su parte, con la finalidad de facilitar la manipulación y orientación de la cámara, esta cuenta con un cono que representa su alcance de visión, por lo que con base en esto se podrá valorar que aquellas partes del escenario fuera de este cono no saldrán en la pantalla al ejecutar el videojuego. Cabe señalar que todo lo anterior equivale al posicionamiento de una cámara que estará fija; sin embargo, rara vez un videojuego empleará solamente una toma fija, por lo que la plataforma de Unity cuenta con un componente de control de cámara llamado *seguimiento sutil* (*smooth follow*), que se puede agregar a la cámara. Para hacerlo se requiere seleccionar el panel conocido como *Inspector*, donde aparecerá la lista de parámetros en la que se debe especificar cuál objeto se seguirá y a qué distancia. De esta manera la cámara siempre seguirá al personaje principal. También es importante decir que este pequeño código es sumamente simple y puede ser útil en un proyecto sencillo para pruebas preliminares, aunque se debe saber que es un comportamiento de cámara poco realista. Otro aspecto que se tiene que destacar es que el código no detecta si hay obstáculos entre la cámara y el personaje, por lo que en ocasiones se pueden ver los interiores “huecos” de los edificios u otros detalles que realmente no deberían ser visibles para el jugador. Por lo tanto, es conveniente saber que hay códigos más complejos que se pueden desarrollar para el seguimiento de objetos mediante cámaras, los cuales pueden tener comportamientos de aceleración y de elasticidad con relación al movimiento de la cámara, filtros de detección de objetos para no atravesarlos y enfoques en función de la actividad de la escena.

Manejo de iluminación. Si se ejecuta el ambiente con *play* en la plataforma de Unity, al menos hasta este punto de la producción de videojuegos, solo será posible ver la silueta de los modelos y terreno en edición. Esto se debe a la ausencia de fuentes de luz establecidas. Así, en Unity y en casi todos los sistemas existen tres tipos de luces (Zerbst, 2004):

- a. Luces de punto (*point light*). Se genera un objeto de emisión de luz radial equitativa en todas las direcciones; p. ej., un foco o un área de alcance limitada.
- b. Luces direccionales (*directional light*). Estas se colocan en puntos distantes y afectan a todos los objetos en la escena; p. ej., el sol.

- c. Luces de reflector (*spot light*). Emiten luz desde un punto en una dirección y solamente iluminan los objetos dentro de un cono; p. ej., los faros de un automóvil. Este tipo de luz es la más compleja por los cálculos que se deben realizar.

En pocas palabras, las luces también son objetos de juego y cuentan con parámetros específicos en el panel derecho de inspección de la plataforma Unity, donde se puede controlar su intensidad, la gama de color e intensidad de esta, el ángulo en el que se emite, etc., lo que permite personalizar el ambiente que se va gestando al combinar el terreno, los modelos 3D y la iluminación para lograr la ambientación establecida originalmente en el guión. En este sentido, es probable que un videojuego cuya temática verse sobre la acción dramatizada, tenga colores más cálidos e intensos contrastes en función de la acción; y por el contrario, un juego que busque un ambiente relajado, tenga colores más fríos o de menor intensidad de contraste.



Figura 2.18. La iluminación y el ambiente generados en esta escena son coherentes con lo establecido en el guión, ya que crean una mayor dramatización. La historia de este videojuego trata de un reciente terremoto en una ciudad y el objetivo es rescatar a la mayor cantidad de heridos y sobrevivientes

Programación de la interacción de objetos de juego en Unity

Requerimientos previos de programación

La plataforma Unity tiene la opción de programar los *scripts* de acción de objetos de juego en los lenguajes C#¹¹ y JavaScript.¹² Los lenguajes de las APIs (interfaz de programación de aplicaciones) que se proporcionan como ejemplo no son muy distantes; no obstante, se adopta el segundo (JavaScript) para la programación que se trata aquí, ya que es el que cuenta con más documentación al respecto.

Ahora bien, más que concentrarse en el lenguaje de programación, es sustancial entender el modelo de Unity para los eventos y los controles de acceso a variables del ambiente y de los objetos que generalmente están disponibles a través de funciones de muy alto nivel en librerías proporcionadas con la plataforma para los desarrolladores.

Comportamiento y parametrización del simulador de física

En Unity, los parámetros de física se controlan en dos etapas. La primera tiene que ver con las especificaciones generales para todo el proyecto; y la segunda, con los componentes especiales que se agregan a los objetos de juego. Estos componentes, básicamente son dos:

- *Cuerpo rígido* (rigidbody). Si se agrega un cuerpo rígido a un objeto, este actuará bajo los parámetros de física del proyecto. Existen algunos parámetros propios del cuerpo rígido que modifican el comportamiento específico del objeto; p. ej., la masa, el coeficiente de fricción, el nivel de fuerza de gravedad, las interpolaciones¹³ y la detección de colisiones, entre otros.

¹¹ C# está pensado para ser un lenguaje de programación simple, moderno y de propósito general orientado a objetos. Fue desarrollado por Microsoft dentro de la iniciativa .NET y más tarde fue aprobado como un estándar por la ECMA (European Computer Manufacturers Association) International y después por ISO. Su más reciente versión es la C Sharp 4.0, desde el 12 de abril de 2010 (ECMA, 2006).

¹² JavaScript es un lenguaje de *scripting* basado en objetos, sin tipo y liviano, utilizado para acceder a objetos en aplicaciones. Principalmente se utiliza integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas. Es un dialecto de ECMAScript y se caracteriza por ser un lenguaje basado en prototipos, con entrada dinámica y con funciones de primera clase. Asimismo, ha tenido influencia de múltiples lenguajes y se diseñó con una sintaxis similar al lenguaje de programación Java, aunque más fácil de utilizar para personas que no programan (Codexempla, 2010).

¹³ Efecto que suaviza los movimientos provocados por parámetros físicos, que muestran los objetos del juego.

- **Controlador de personaje (character controller).** Como su nombre lo dice, este componente se usa principalmente para que los personajes que requieren comportamientos físicos especiales puedan verse más realistas. Se representa por una cápsula cuyos bordes son de color verde e incluye parámetros como altura, radio, límite de elevación, compensación de caminata, grosor de la piel, etc.

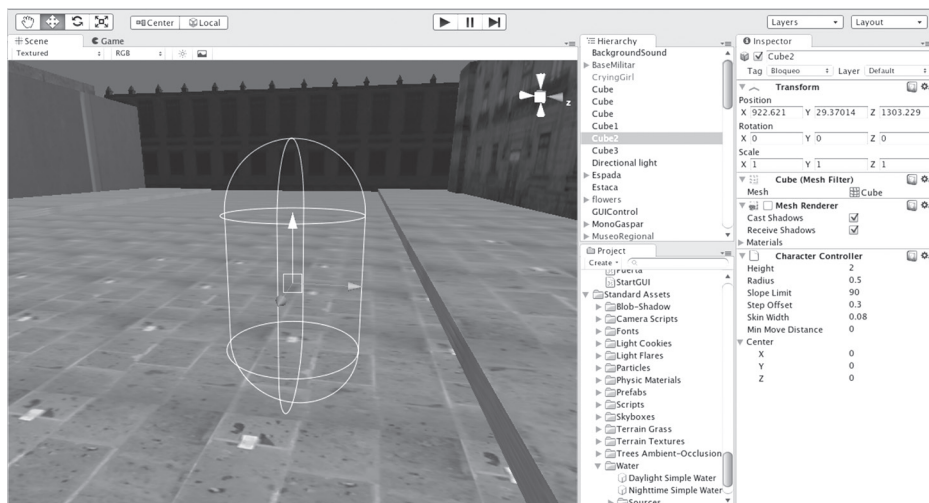


Figura 2.19. Representación de un *character controller* en la escena mediante una cápsula de bordes verdes

El simulador de física debe ser alimentado con parámetros adecuados para poder generar comportamientos realistas, es decir, si se dispone de un objeto de juego que representa un automóvil, habrá que asignarle una masa de una o tal vez dos toneladas de peso para que pueda tener una dinámica similar a la del mundo real. Así, cuando este objeto entre en interacción con otros que dispongan también de un *rigidbody*, Unity podrá hacer los cálculos de las fuerzas resultantes y simular toda la dinámica de efectos físicos.

Parámetros básicos

Con relación al simulador de física de Unity, habrá parámetros que afectarán todo el proyecto. No se requiere ser un experto en física, pero sí es recomendable tener nociones básicas de modelado matemático (leyes de Newton y física mecánica) para que se pueda lograr una manipulación realista en el comportamiento del videojuego. Los parámetros más importantes son los siguientes:

- *Gravedad*.
- *Material por omisión*. Es el que se aplica a un colisionador si no se ha especificado ninguno.
- *Umbral de rebote*. Dos objetos que colisionan con una velocidad relativa menor a este valor no rebotarán.
- *Velocidad para dormir*. Es la velocidad lineal inicial bajo la cual los objetos comenzarán a dormir (no tendrán movimiento).
- *Velocidad angular para dormir*. Es la velocidad angular por defecto bajo la cual los objetos comenzarán a dormir.
- *Velocidad angular máxima*. Es el valor máximo permitido para cualquier objeto de tipo *rigidbody*.
- *Penetración mínima para penalidad*. Es la profundidad en metros permitida para dos objetos antes de que el sistema de física los separe.
- *Cuenta de iteraciones del solucionador* (solver). Determina qué tan precisos son calculados los contactos entre objetos.
- *Intersección de rayos con colisionadores*. Si está habilitado, equivale a cualquier rayo que intersecte con un colisionador marcado como gatillo (*trigger*) que lanzará un golpe.
- *Matriz de colisión de capas*. Define el comportamiento del sistema de detección de colisiones basado en capas.

Efectos sonoros de fondo

En la plataforma Unity, cada escena debe de tener un componente llamado *audio listener*. Dicho componente permite que los sonidos que se encuentran en la escena sean escuchados. Generalmente va en el objeto de la cámara principal, lo que garantiza un flujo de sonido sin sobresaltos. A saber, una escena solamente puede tener un *audio listener*, pero puede contener varios *audio sources*. Estos últimos son componentes que emiten sonido; su intensidad va en función de los parámetros especificados (volumen, variación, etc.) y la distancia que exista entre el *audio source* y el *audio listener*. Dentro de las aplicaciones de este *software* incluso se puede presenciar el efecto Doppler, el cual modifica la frecuencia del sonido en función de la distancia del observador en relación con la fuente emisora sonora.

Shaders

En general los *shaders* son programas que permiten tener acceso directo a la unidad de procesamiento gráfico (GPU) (Zerbst, 2004), de manera que se pueden realizar transformaciones de los objetos 3D muy rápidas, es decir, se emplean

para efectos especiales. Si se habla con más detalle, los *shaders* son programas y procedimientos numéricos que instruyen al motor 3D, mediante acceso directo al GPU, la manera de producir varias texturas y componentes individuales de las mismas con base en parámetros específicos controlados por el usuario, y que pueden dar lugar a la simulación de partículas en movimiento para la generación de efectos de fuego, humo, neblina o iluminación especial (Evry, 2004).

En relación con lo anterior, Unity cuenta con un amplio soporte de *shaders* preprogramados que permiten lograr efectos realistas e interesantes a través de los materiales. Aunado a ello, hay que remarcar que el éxito en la ejecución de un buen efecto de partículas depende de una correcta utilización del *shader*. A continuación se hace una breve explicación de los *shaders* con los que cuenta Unity:

- *Normales* (normal). Son los más básicos y se utilizan en objetos opacos no reflectivos, como madera o plástico.
- *Transparentes* (transparent). Usan el canal alfa¹⁴ de la textura base. Se pueden determinar áreas de un objeto con partes más o menos transparentes. Esto permite crear grandes efectos para vidrio, efectos de ciencia ficción o incluso en la interfaz gráfica.
- *Transparentes recortados* (transparent cutout). Se emplean para transparencia y opacidad totales. Se usan cuando los objetos contienen partes que son completamente opacas o transparentes, tales como cadenas, cercas, árboles, hierba, etc.
- *Autoiluminados* (Self-illuminated). Estos emitirán luz solamente hacia ellos mismos en un canal alfa adjunto. No requieren de ninguna iluminación ajena en su dirección para despedir luz, y cualquier luz de vértice o luz de pixel solamente agregará más intensidad de luminosidad.
- *Reflectivos* (reflective). Permiten usar una malla sobre la superficie en que será reflejada la masa; se pueden definir áreas con más o menos reflectividad por medio del canal alfa de la textura base, donde una alta reflectividad es muy eficiente para crear efectos de cromado o aceitado y una baja reflectividad puede agregar efectos a metales, superficies líquidas o monitores de video.

Motor de partículas

Este motor, con programación basada en *shaders*, sirve para crear efectos atmosféricos, que por otro medio serían muy difíciles de simular. En la plataforma de Unity se presenta como un conjunto de herramientas que contienen los *shaders* de partículas que permite la creación de objetos de juego, lo que agiliza la creación de efectos especiales de la escena en desarrollo. El principio

¹⁴ Canal de una imagen que representa la transparencia de la misma (Franson, 2004).

del motor de partículas sobre los objetos de juego que se crean consiste en una serie de imágenes en 2D. Estas se comportan con base en ciertos parámetros que establecen si se pueden cambiar, aparecer o desaparecer, para dar efectos de fuego o humo de diferente densidad. Enseguida se mencionan estos componentes preconfigurados:

- *Emisor de partículas elipsoide* (ellipsoid particle emitter). Emite partículas dentro de una esfera en la que se pueden definir sus límites de producción y su velocidad inicial. A partir de este emisor, para crear efectos más realistas, se deberá agregar un *particle animator*.
- *Animador de partículas* (particle animator). Mueve las partículas a través del tiempo. Se emplea para agregar viento a estas, así como arrastre en el espacio 3D y ciclos de colores a generar. Asimismo, les añade dinamismo, dado que les permite cambiar de color, aplicar fuerzas y rotaciones; y las destruye una vez que ha concluido la emisión en el tiempo definido.
- *Renderizador de partículas* (particle renderer). Despliega las partículas en la pantalla. Este componente es necesario para que el usuario pueda ver los efectos programados en objetos mediante *shaders* con los sistemas de partículas.

Si bien todo lo anterior comprende componentes básicos de un sistema con motor de partículas, hay que señalar que Unity también cuenta con otros componentes como el *line renderer*, que dibuja líneas; o el *trail renderer*, que se usa para crear rastros detrás de objetos que se mueven. En síntesis, todos estos elementos generan efectos puramente visuales; sin embargo, se puede hacer que el sistema de partículas interactúe con el resto de los objetos del juego si se agrega un *particle collider*. Este componente determinará si hay contacto por colisión con un objeto de juego y enviará mensajes a los objetos involucrados. Por su parte, en combinación con *scripts* que detecten estas colisiones, se puede generar un flujo de juego más interesante; por ejemplo, al emplear un *particle collider* se puede hacer que el personaje principal, al ser tocado por el fuego, pierda puntos de su energía o vida en el ambiente.

En los siguientes párrafos se muestran tres de los efectos de partículas más empleados. Cabe mencionar que los parámetros que se presentan son a manera de sugerencia.

- *Explosiones*. Unity cuenta con dos *prefabs*¹⁵ para crear estos efectos: la normal y la pequeña. Simplemente se debe arrastrar de la carpeta de *standard assets* hacia la escena y la explosión será agregada.
- *Fuego*. En Unity también existen dos *prefabs* que se pueden usar para representar fuego: *flames* y *small flames*. Hay que especificar que estos son efectos genéricos y en ocasiones se requerirá de una intensidad determinada, por

¹⁵ Copias de algún objeto de juego que se actualizan de manera automática al modificar el original.

lo que en estos casos lo que se recomienda es jugar con los parámetros del sistema del motor de partículas para lograr el efecto deseado.

- *Humo*. Desafortunadamente este efecto no viene establecido como *prefabs*, aunque es posible producirlo. Para ello es necesario poner especial atención en la textura del humo y en el *shader* que será utilizado (*particles/alpha blended*). Se pueden encontrar diferentes valores mediante el generador de partículas, el *shader* y la generación de varios hilos para la animación.¹⁶

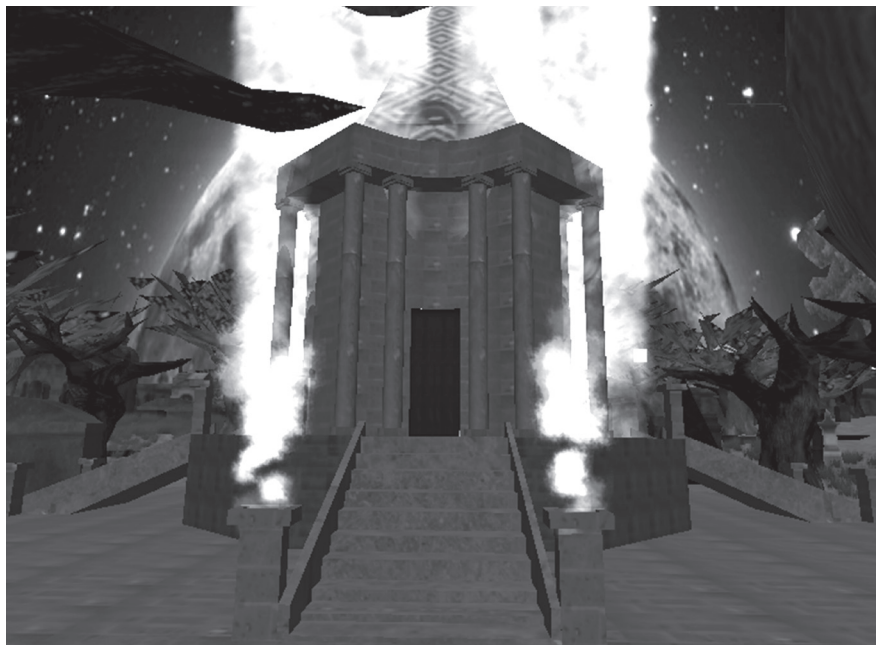


Figura 2.20. Representación en la escena de los efectos de fuego y humo, generados por el motor de partículas

Scripts básicos

A través de la práctica y del entendimiento de las librerías de programación de *scripts* en la plataforma de Unity se podrán hacer mejores comportamientos para la dinámica del videojuego. En este sentido, para iniciar la creación de uno existen algunos *scripts* que se consideran básicos:

¹⁶ Para más información veánse las siguientes páginas: <http://docs.unity3d.com/Documentation/Components/script-GlobalFog.html>; <http://www.gamedev.net/blog/633/entry-2254758-volumetric-objects-in-unity/>; <http://docs.unity3d.com/Documentation/Components/SL-Fog.html>; <http://forum.unity3d.com/threads/17034-fog>

Cámara en tercera persona

El programa de este *script* viene precargado en el paquete de *standar assets* de Unity y forma parte de todo proyecto. Su uso es sencillo debido a que solo requiere dos cosas, que el objeto de juego que será seguido por la cámara sea arrastrado y modificar los parámetros que se consideren necesarios. Este *script*, como su nombre lo indica, debe ser agregado al objeto de juego *main camera*.

Disparo de animaciones del personaje u objetos

No siempre es necesario que se esté repitiendo alguna animación de los objetos que se desea que tengan un comportamiento dinámico, por lo que se requiere crear condiciones que desaten la ejecución de cierta animación. Por ejemplo, si el nivel de energía del personaje principal llega a cero, es posible que quiera disparar la animación de *morir*, dado que dicha animación está prevista para momentos específicos del juego de acuerdo con el guión establecido. Para lograr esto es importante tener identificadas (por su nombre) las animaciones que contiene el personaje dinámico, con la finalidad de mandarlas llamar cuando se requiera. Este *script* se debe agregar al personaje de cuyas animaciones queramos controlar.

Eventos por colisiones

Otra manera de disparar animaciones (y otros eventos) es por medio de sensores de colisiones. Estos son componentes que se añaden a un objeto de juego cuya propiedad *trigger* está activada. Cuando dos objetos (y al menos uno de ellos cuenta con un *rigidbody*) que dispone de sensores de colisiones chocan, se envían tres mensajes:

- *OnTriggerEnter*. Se envía cuando la colisión ha comenzado.
- *OnTriggerStay*. Se envía si ambos objetos se están tocando pero ninguno se está moviendo.
- *OnTriggerExit*. Se envía cuando los objetos se dejan de tocar.

Vale la pena referir que al codificar las acciones que se ejecutarán cuando se desate una colisión, se tendrá que especificar el objeto que recibirá dichas acciones, ya que en caso de no hacerlo estas se aplicarán para el objeto que tiene este *script*. Si se cuenta con un objeto de juego llamado *automóvil*, el cual tiene un *script* que indica que cuando se detecte una colisión con el objeto *persona* esta deberá ser arrojada un metro en Z, y al codificar se hiciera así:

```
transform.Translate (Vector3.forward)
```

Lo único que se conseguiría con ello es aplicar la fuerza al automóvil. En consecuencia, si se requiere que la persona sea el objeto de juego afectado, entonces se necesitaría codificar de la siguiente manera:

```
persona.transform.Translate(Vector3.forward) Manejo de pantallas de texto informativas
```

Un juego que no tiene una interfaz gráfica —denominada HUD (*head-up display*)— que provea información interactiva al usuario alfanumérica, como el marcador, mapas, estados, etc., es muy difícil de entender. Por lo tanto, es importante determinar cuál será la información que el usuario deberá tener siempre visible (puntaje, nivel de salud, nivel en el que se encuentra, herramientas, opciones, entre otros). En Unity, este tipo de elementos se manejan a través de la función `OnGUI()`, la cual es llamada varias veces en cada paso de refresco de la pantalla de salida del juego. En la figura 2.21 se proporciona un ejemplo de código de bienvenida que muestra una imagen de fondo, así como dos botones que dan las opciones de *jugar* o *salir*.

Este *script* (o cualquier otro con la función `OnGUI()`), debe ser agregado a objetos del juego que estarán siempre activos en la escena. Es por esto que se recomienda crear un objeto de juego vacío que contenga este *script*, de manera que no se corra el riesgo de que dependa de un objeto que puede ser destruido (como un *avatar*) y, por ende, desaparezca de la pantalla.



Figura 2.21. Interfaz gráfica de usuario creada con la clase GUI denominada HUD

Control de personajes

Las APIs de Unity cuentan con una clase específica que maneja los controles de juego llamada *input*; a través de esta es posible indicar cuáles teclas o botones dispararán acciones específicas. En razón de que es probable que el primer control que necesite sea de movimiento, se proporciona un *script* que hace que el objeto que lo contenga se mueva. No hay que olvidar que es probable que la mayoría de los objetos dinámicos requieran de un *character controller*.

Activación de luces

Previamente ya se trataron algunos conceptos básicos de iluminación en una escena, los cuales, como ya se constató, son de suma importancia. Solo resta mencionar que las luces tienen propiedades que pueden ser modificadas mediante código, lo cual permite alcanzar un mejor control de la escena y la dinámica de eventos en la misma. Para mayor referencia consultar la documentación actualizada de Unity en el tema de gestión de luces y *scripts*: <http://docs.unity3d.com/Manual/Lightingoverview.html>.

Es recomendable que el código de iluminación generado se agregue a un objeto vacío —que se denominará *world manager*— para que no interfiera con otros objetos en la escena.

Sonidos en relación con eventos

Al igual que con las animaciones, los sonidos pueden ser activados cuando se cumpla alguna condición o evento programado en un *script*. Esto se puede lograr a través de colisionadores físicos o condicionales que activan la reproducción de un sonido predeterminado. Así, cada vez que el jugador recoja un objeto, por ejemplo, se podrá reproducir un tintineo que indique que este ha sido adquirido.

Modelado de personajes 3D

No en todas las ocasiones de un videojuego serán indispensables figuras humanas; no obstante, todo personaje, sin importar su morfología, requerirá ser animado, ya sea un humanoide, un animal o algún objeto que comprenda conductas específicas, y deberá contar con movimientos coherentes entre rótulas y articulaciones de su cuerpo.

Con el incremento de la tecnología y, en consecuencia, de las capacidades de las computadoras, los objetos de un videojuego que antes eran estáticos, ahora

son animados (Byrne, 2004). La figura 2.22 muestra de manera general todas las opciones para la creación de personajes; además, puede servir como una guía para realizar cualquier tipo de objetos animados que caractericen un personaje demandado por el guión de la historia del juego.

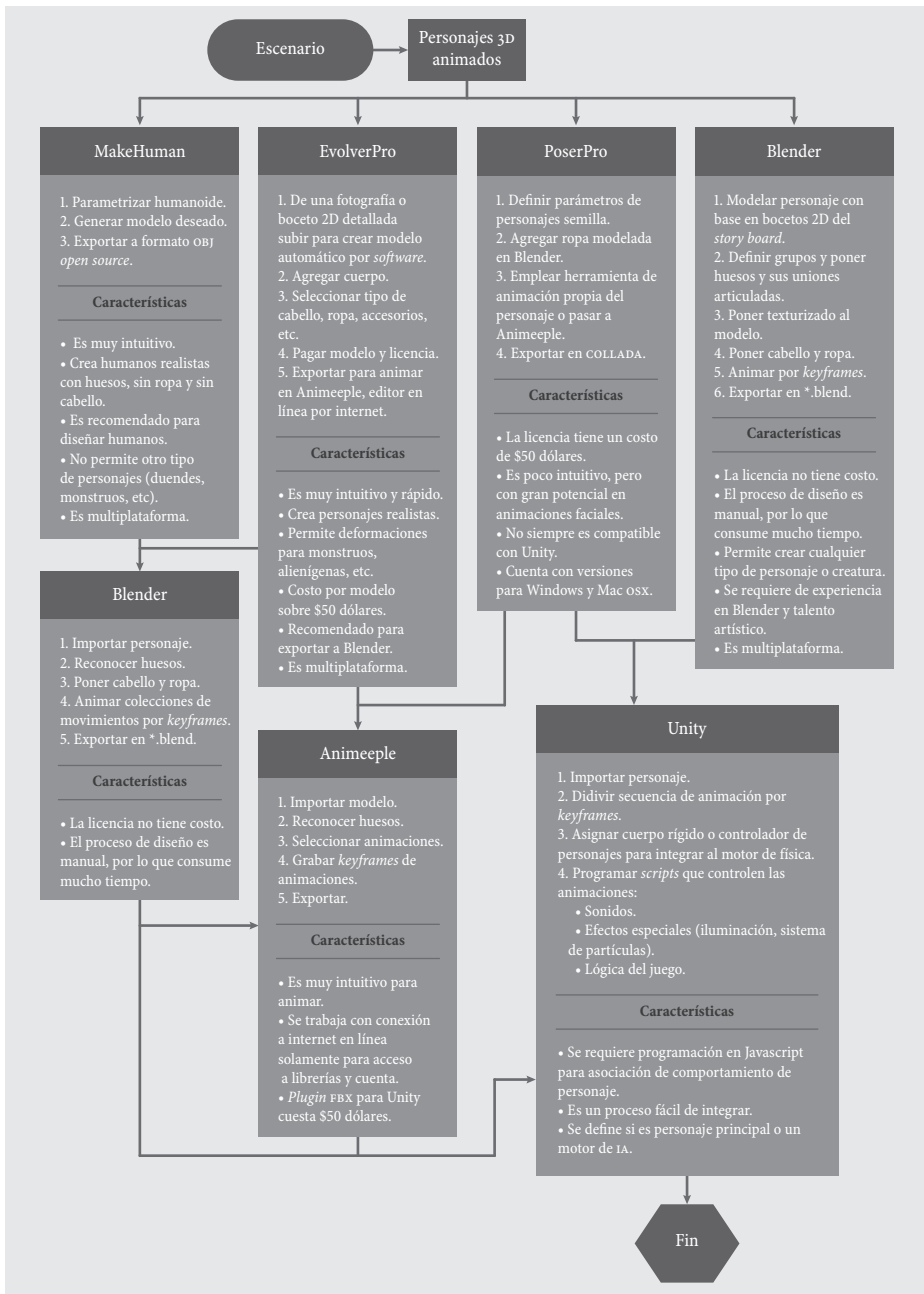


Figura 2.22. Diferentes opciones y pasos para generar personajes en videojuegos

Consideraciones del personaje

Como ya se dijo anteriormente, en un videojuego debe haber coherencia con los elementos que lo componen. En el caso particular de los personajes existen algunos aspectos en los que se debe tener especial cuidado para evitar terminar con modelos visualmente agradables, pero con movimientos incompatibles o poco naturales para la dinámica planeada del videojuego. Estos detalles son los siguientes:

Relación con el guión. Lo primero que se debe revisar antes de comenzar a modelar un personaje 3D es la especificación que se le dio en el guión. Esta puede ser textual o mediante un bosquejo, y debe tratar de apegarse lo más posible a ella. Lo ideal es una descripción mixta que represente visualmente cómo se quiere el modelo terminado y una descripción textual que aporte información relevante para el modelador, el animador e incluso el programador sobre los atributos del carácter del personaje, sus expresiones y temperamento para su reproducción digital. Por ejemplo, si se detalla en el guión que un personaje arrastra el pie derecho al caminar, el animador no tendría forma de saberlo si no fuese por la descripción textual y la explicación de por qué es importante.

Apariencia. Lo segundo que hay que revisar es el aspecto del personaje. Al igual que con el escenario, la apariencia debe respetar los lineamientos, ya sean explícitos (en el guión) o implícitos (cultura general). Si se tuviera un escenario en algún lugar de China no se concebirían personajes de 1.80 metros promedio de altura, de cabello rubio, ojos azules y de piel blanca (lineamiento implícito), sino que se esperarían personajes con complexión y fenotipo chino. Otro caso sería que si en el guión se ha especificado que la temática del juego versa sobre una invasión por parte del imperio británico, entonces tendríamos una justificación para este tipo de personajes en ese escenario en específico (lineamiento explícito).

Realismo. Se considera que en aplicaciones de juegos serios, personajes realistas ayudan a crear un mayor impacto y lograr mayor inmersión por parte del jugador, especialmente en simulaciones que requieren que el jugador interiorice ciertos conceptos o acciones. En este sentido, poder visualizar un avatar que representa a la figura humana lo más fielmente posible permite que el jugador se enganche más fácilmente con el flujo de este.



Figura 2.23. Personaje realista creado con MakeHuman

Caricaturización. Hay ocasiones en que un juego pretende maximizar la transferencia de información hacia el usuario y, por lo tanto, incorpora ciertas distorsiones. Estas distorsiones podrían tomar la forma de personajes caricaturescos que busquen resaltar un punto en específico. Por ejemplo, un videojuego serio que tratase de resaltar un problema de obesidad en un personaje podría recrear personas excesivamente obesas, al punto de parecer poco humanas. Esto estaría justificado por el hecho de que se intenta transferir información acerca de un problema de salud pública tan grande como lo es la obesidad.



Figura 2.24. Personaje caricaturesco disponible en Animeeple

Vestuario. Al igual que en la vida real, el personaje creado necesita de una vestimenta. Para lograr una mejor inmersión del usuario en el contexto del juego, esta vestimenta debe seguir modas y tendencias de la época en específico en que se sitúa el guión y la historia. En este punto la información adicional dispuesta en el guión sirve para traducir la personalidad ya definida en modelos de ropa. Es decir, se puede definir a un personaje joven ubicado en una zona específica de la ciudad de Los Ángeles, lo cual daría una idea del tipo de vestimenta a utilizar. Sin embargo, si el guión especifica que este personaje es un *estudiante de Oxford en Inglaterra*, entonces tanto el modelador como el animador sabrán que deben trabajar con características diferentes.

Modelado de ropa. Modelar un personaje 3D es un asunto y modelar la ropa es otro. Un modelo desnudo o semidesnudo en la mayoría de los casos estará incompleto. Es por ello que a continuación se muestran algunas técnicas simples para modelar las prendas de vestir más básicas que requerirá el modelo:

- a. Playera o camisa. El primer paso es crear una copia del modelo y moverla a una capa diferente del original. Después se eliminan los vértices de la cintura hacia abajo, así como cabeza, cuello y manos. El modelo resultante tendrá la forma de una camisa. Luego se aplica un nivel *multires*. También se puede cambiar a modo de escultura y después, con una brocha suave, minimizar estructuras no estéticas. Posteriormente se puede cambiar la brocha a *inflate* y asegurándose de que ambas capas sean visibles, crecer la talla del modelo de la camisa hasta obtener un resultado deseado.
- b. Pantalón. El proceso es similar al anterior, con la diferencia de que en esta ocasión lo que se elimina son los vértices de la cintura hacia arriba y los pies.
- c. Zapatos. Se suprimen todos los vértices del talón hacia arriba.

Cantidad de polígonos. Este aspecto, lejos de ser visual, está relacionado con el desempeño de la aplicación. Todos los puntos comentados anteriormente tienen un costo o un ahorro en términos de procesamiento y memoria de la computadora o consola de videojuego. Mientras más complejo sea el modelo, más polígonos tendrá, y requerirá de más memoria y capacidad de procesamiento. Dicho esto, un modelo realista es más pesado que uno caricaturesco. Modelar características muy específicas de una raza puede implicar una gran cantidad de tiempo invertido, y si se recurre al modelado 3D (en lugar del texturizado), entonces el modelo será aún más pesado. Cada polígono cuenta en un videojuego, por lo que cualquier cara sobre un polígono del objeto 3D que no sea necesaria para su modelo debe ser eliminada. Es en tal sentido que se recomienda que, una vez contando con las piezas de ropa del personaje, se eliminen las partes del cuerpo del modelo que van por debajo de la ropa y solamente se conserven las partes del cuerpo visibles.

Ahora bien, es en función de la complejidad y los requerimientos del proyecto plasmados en el guión que se puede decidir si los personajes deben ser modelados

y animados desde cero o si vale la pena comprarlos. En las siguientes líneas se habla de los pros y los contras de ambas decisiones, además de las fuentes disponibles de personajes animados más comunes o formas de identificarlas en un futuro.

Si se decide modelar y animar a los personajes de manera individual, entonces se deberá considerar la adquisición de un *software* especial para estos fines. Sin embargo, esta decisión también tendrá que tener en cuenta otros factores como el costo de la licencia, la curva de aprendizaje, la capacitación, la plataforma de destino, la complejidad de los modelos, entre otros. En caso de que se decida modelar personajes desde cero, probablemente será necesario dedicar específicamente a algún miembro del equipo para esta tarea, pues el proceso consume mucho tiempo. Asimismo, será necesario incubar talento y conocimiento tanto del *software* como de la anatomía del personaje, lo que tiene implicaciones en costos que se deben sumar a los de licencia y equipo específico para esta tarea.

Poser Pro 2010.¹⁷ Este *software* es una buena herramienta para introducirse al mundo de la animación. Desafortunadamente hay que considerar que la licencia proporcionada solo permite su explotación en proyectos personales o demostrativos, ya que de acuerdo con su EULA¹⁸ se estipula que solo se puede usar el contenido no restringido en aplicaciones comerciales. Dicho contenido no restringido está limitado a cuatro figuras:

- a. *Low res male*.
- b. *Low res female*.
- c. *Medium res male*.
- d. *Medium res female*.

Y aún así las texturas son propiedad de Poser Pro 2010, por lo que se tendrían que generar texturas propias, lo cual podría restar el atractivo comercial de esta aplicación.

Animeeple. Es un *software* gratuito en su base pero con costo de acceso a funciones más avanzadas, que permite utilizar los modelos y animaciones para fines comerciales. Cabe señalar que los modelos de personajes proporcionados en el soporte de librerías de Animeeple son un tanto caricaturescos, aunque cuentan con la ventaja de ser ligeros y compatibles con el formato COLLADA y FBX. Otra de sus ventajas es que también soporta una tienda donde, a través de la aplicación, se pueden comprar modelos más complejos diseñados por artistas independientes, lo que lo hace de un carácter dual. Además, es realmente fácil de utilizar y tiene una curva de aprendizaje corta.

¹⁷ Para más información, véase <http://poser.smithmicro.com/poserpro.html>

¹⁸ *End-user license agreement* o acuerdo de licencia con el usuario final.

Blender. Ya se ha discutido la relevancia y la potencia de Blender para estas tareas de modelado de cualquier objeto 3D (sea dinámico o estático), y se incluye dado que soporta los procesos de modelado y de animación. Tiene la ventaja de permitir realizar dos procesos en una sola aplicación sin lidiar con problemas de incompatibilidad de formatos.

Con base en las especificaciones del proyecto, se puede llegar a la conclusión de que no es necesario destinar recursos al proceso de modelado de los personajes y que es preferible comprarlos. Pero, como todo, esta decisión tiene aspectos negativos y positivos, ya que, en principio, el precio a pagar por el producto puede ser alto, y es probable que se tenga que conformar con el personaje que más se acerca a las especificaciones y no por el idóneo establecido en el guión. De igual forma, habrá que cerciorarse de que el formato en el que el modelo le será entregado sea compatible con alguno que acepte el motor 3D y de que el modelo comprado incluya texturas y materiales (dependiendo de las condiciones del proveedor, estos podrían venderse de manera independiente).

Asimismo, otro aspecto importante que no hay que perder de vista es la licencia con la que viene el modelo. En distintas variaciones de la misma se puede especificar si dicho modelo puede ser empleado en proyectos comerciales o no.

Hoy en día existe una gran cantidad de repositorios de objetos 3D soportados por artistas profesionales que actúan como mercado para elementos 2D y 3D. Se puede comprar todo tipo de elementos, desde simples texturas hasta animaciones. Por lo general estas páginas otorgan cuentas a los usuarios (tanto compradores como vendedores) y obtienen una comisión por la transacción. En este sentido, empresas como 3D Export,¹⁹ Exchange3D,²⁰ 3D Model Shop,²¹ 3D Models for Sale²² y Buy 3D Models²³ han ido aumentando la cantidad, calidad y variedad de modelos que ofertan. En el caso de la última empresa existe la opción de cotizar modelos hechos a la medida, aunque es importante atender las recomendaciones hechas al inicio de esta sección.

A manera de consejo, casi todas los repositorios de modelos 3D cuentan con una sección de modelos gratuitos. Tal vez vale la pena revisarla antes de proceder a comprar uno. Es muy probable que se prohíba el uso para fines comerciales, pero dichos modelos pueden ser de utilidad para proyectos personales o demostrativos.²⁴

¹⁹ Para más información, véase <http://www.3Dexport.com/>

²⁰ Para más información, véase <http://www.exchange3D.com/>

²¹ Para más información, véase <http://www.3Dmodelshop.com/>

²² Para más información, véase <http://3Dmodelsforsale.com/>

²³ Para más información, véase <http://www.buy3Dmodels.com/>

²⁴ Para este tipo de proyectos, se recomienda revisar el sitio Open Game Art, <http://opengameart.org/>

Animación

Una vez que se han trabajado los modelos 3D que constituirán a los personajes requeridos en el guión, se deberá continuar con el proceso de animación. Ya se ha comentado antes que es en función del programa de modelado 3D que se podrá o no seguir la cadena de producción con las animaciones en la misma plataforma. En caso de que no sea posible, se requerirá exportar y emplear otro programa de edición. Lo importante es tener presente cuáles son las características de la animación que se pretenda implementar para saber la plataforma con la que se puede lograr esto. En la figura 2.25 se señalan de una forma general los procesos y pasos para poder producir animaciones globales en objetos genéricos, sean personajes u otro tipo de objetos del juego como puertas, accesorios, vehículos, robots, etc.

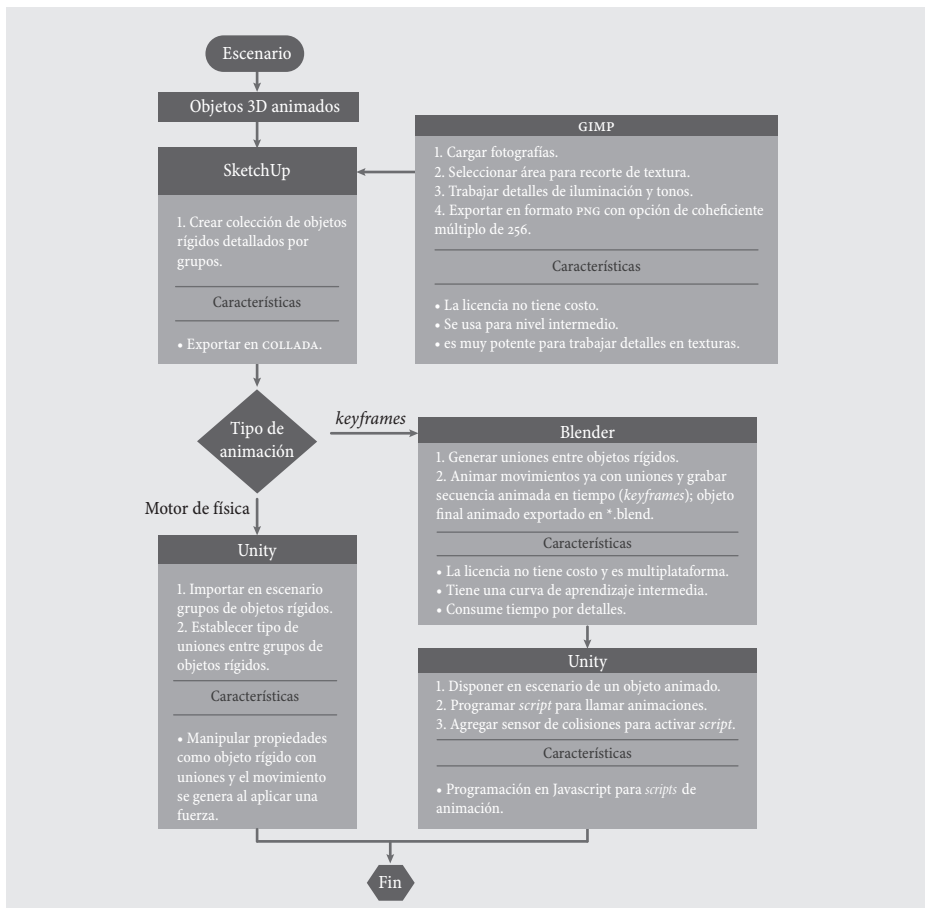


Figura 2.25. Distintos procesos realizados con todos los programas mencionados en el capítulo y los pasos necesarios para generar una animación

Principios básicos

Para comenzar con una muestra muy básica de una animación, lo que se pretende explicar es cómo mover la posición de los huesos a lo largo del tiempo utilizando *keyframes*. Para ello solo se debe especificar dónde tiene que iniciar y terminar alguna posición de una articulación de un modelo 3D, y el programa se encargará de calcular los movimientos necesarios de todas las piezas relacionadas con la articulación o rótula. Depende de la herramienta que se esté empleando saber si el proceso será de edición manual o automática.

- *Vértices*. La animación de vértices requiere que cada uno de ellos sea movido individualmente. Obviamente este proceso es tardado, manual y requiere de mucha precisión. Como resultado se suelen obtener animaciones no muy realistas, aunque en algunas ocasiones puede ser el único camino; sin embargo, para el animado de personajes se recomienda hacerlo por medio de esqueletos.
- *Esqueletos*. La animación basada en esqueletos generalmente es el estándar para la animación de personajes. Consiste en atar los vértices del modelo a puntos de control (huesos) que pueden ser movidos, y, consecuentemente, los vértices específicos de ese grupo también serán movidos. Entre los distintos beneficios que ofrece este tipo de animación destaca que se puede encontrar el mayor realismo de movimientos, el menor espacio en disco que toma almacenar la animación (puesto que el cálculo completo de movimiento de vértices se calcula en tiempo real en la tarjeta gráfica) y el ahorro del tiempo en los procesos.
- *Composición*. Ya que los esqueletos están hechos de huesos y son objetos invisibles, se necesita de un modelo 3D para ver materializados los resultados de la animación. Los huesos suelen estar divididos en tres partes: un tronco central y dos extremos; el tronco central determina el tamaño del hueso y las pequeñas partes en los extremos funcionan como tendones para unir huesos.
- *Nomenclatura tradicional*. Si bien los huesos son invisibles, también requieren de un tratamiento como objetos en la escena, por lo que se les debe asignar un nombre, además de otras propiedades. Es un estándar de la industria utilizar “L” (izquierda) o “R” (derecha) como prefijo de algún hueso junto con el nombre de la parte del cuerpo en inglés, de manera que el hueso de la mano izquierda será *Lhand*. Esto se hace para evitar confusiones en las partes del cuerpo que tienen una correspondencia simétrica. Por su parte, para el resto de los casos se emplea el nombre simple; p. ej., *head* (cabeza).
- *Pistas discretas de tiempo con secuencias de animaciones*. Lo ideal en las animaciones sería establecerlas de manera independiente y después en el videojuego mandarlas llamar por su nombre; sin embargo, esto no siempre es posible. Es muy probable que el sistema de animación más simple le permita

grabar animaciones utilizando *keyframes*.²⁵ Mientras pueda llevar el registro del inicio y fin de cada animación será posible manipular las animaciones del modelo de manera sencilla, dado que Unity cuenta con la opción de crear animaciones específicas dividiendo la secuencia entera por sus *keyframes*.

Herramientas de animación de modelos 3D

Poser Pro

Poser Pro es extremadamente útil para crear animaciones faciales en las que el personaje está hablando o haciendo gestos finos. Asimismo, cuenta con la posibilidad de modificar algunos parámetros en el Talk Designer para resultados refinados. El Walk Designer funciona con la misma lógica y tiene la ventaja de poder crear animaciones de caminata con combinaciones de estilos.

Animeeple

Las animaciones con Animeeple son verdaderamente sencillas. Para hacerlas solo es necesario buscar alguna en una librería. Una vez que esta se identifica, se debe arrastrar a la línea de tiempo. Enseguida se creará un bloque que represente dicha animación, al cual se le pueden agregar propiedades como *loop* o *treadmill*. Animeeple cuenta con un exportador de FBX que cuesta \$50 dólares, aunque lo cierto es que si se exporta en DAE también permite introducir el personaje en Unity y se comporta de manera correcta. No obstante, se puede observar la aparición de problemas al tratar de crear colisionadores para el modelo, acción que en todo caso debería ser evitada, pues las colisiones se pueden detectar mediante un *character controller*.

Blender

Al utilizar Blender para modelar a los personajes se tiene la ventaja de que también se pueden crear animaciones individuales en el mismo *software*, guardar el archivo como *.blend y cargarlo en Unity automáticamente. El primer paso a seguir luego de tener el modelo es asignar huesos; después se crean grupos de vértices que serán controlados por dichos huesos, y por último es necesario crear una unión entre los huesos (*armature*) y el modelo a través de la función *make parent*. Ya que se ha realizado esto, lo siguiente es pasar a la ventana de animación donde

²⁵ Inicio o final de una transición.

se muestra la línea de tiempo que indica los *frames*. Para construir una animación es indispensable seleccionar un *frame*, mover los huesos a la posición deseada y grabar dichas posiciones utilizando el comando *i-LocRot*. Se debe proseguir con el siguiente *frame* y continuar con el proceso hasta tener la animación terminada. Unity detectará las animaciones contenidas en el archivo *.blend.

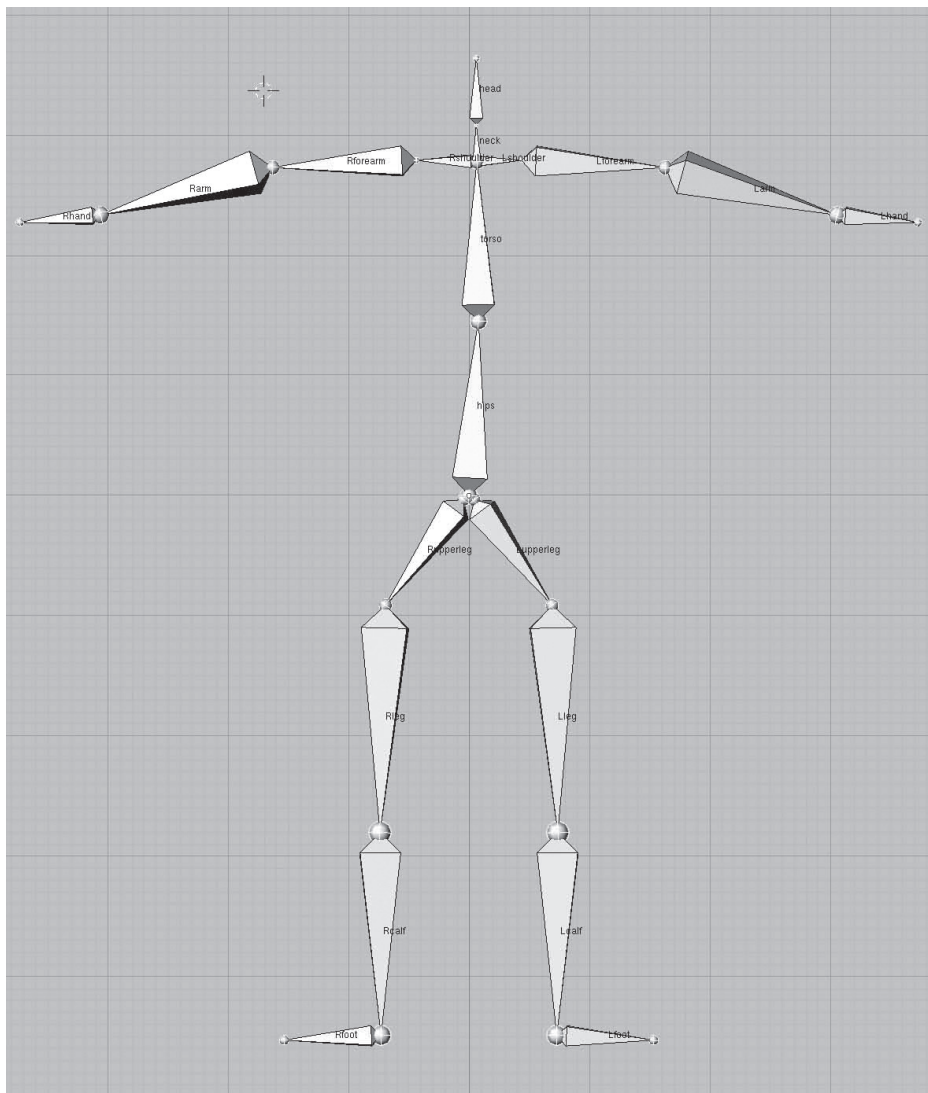


Figura 2.26. Nomenclatura de los huesos de un esqueleto diseñado en Blender

Formatos de exportación

El formato FBX trabaja muy bien con Unity, ya que almacena la geometría y las animaciones de un modelo. Cabe decir que su manipulación es sencilla, aunque es probable que un archivo con extensión *.fbx tenga su origen en una paquetería de modelado 3D y animación que requiera de una licencia de paga. Por el contrario, el formato *.blend de Blender es igualmente útil en Unity pero de manera gratuita. El caso de DAE es un poco más complejo, ya que al ser un formato abierto existen bastantes variaciones del mismo, lo que crea inestabilidad en el flujo de trabajo. Por ejemplo, por alguna razón Unity no acepta el formato *.dae que genera SketchUp; sin embargo, Blender acepta bien el formato proveniente de Google SketchUp (esto nos permite, aun no teniendo la licencia profesional de SketchUp, utilizar los modelos en Unity). Así pues, Unity lee bien los archivos *.dae de modelos estáticos, pero para animaciones se recomienda emplear el *.blend. En caso de querer usar el formato DAE generado por Poser Pro 2010 para Unity, lo primero que se debe hacer es guardar la información de las animaciones y emplear Animeeple como puente, debido a que lee correctamente el archivo DAE generado por Poser Pro 2010. Aquí el modelo es animado dentro de Animeeple y luego se exporta en formato DAE, el cual, de manera misteriosa, es adecuadamente leído por Unity.

Efectos sonoros

Si se piensa en la película de terror que más ha asustado y se quitara el sonido, muy probablemente no logre la dramatización deseada. El sonido es un elemento vital en la creación de inmersión de un jugador. Desde la música de fondo hasta los sonidos específicos detonados por eventos concretos, el jugador incorpora el elemento sonoro a la experiencia de juego.

Repositorios de sonido

En la web existen repositorios gratuitos de efectos sonoros que permiten dar acceso a sonidos ambientales: puertas abriéndose, pasos, explosiones, tormentas, etc.

Es importante revisar los términos y condiciones de cada sitio para cerciorarse de que la licencia permite el uso de estos archivos en proyectos comerciales o de lo contrario solo utilizarlos para fines demostrativos, educativos o personales.

Edición de audio básica

Habrà ocasiones en que los repositorios no cuenten específicamente con lo que se busca en cuanto a arte sonoro, por lo que será necesario encargarse de este proceso. Para llevarlo a cabo se requiere de un *software* para crear o editar sonidos; será esencial tener algún periférico que introduzca audio a este *software*. Por lo general un micrófono basta para sonidos humanos u otros sonidos básicos; no obstante, para crear música de fondo o sonidos más complejos probablemente se precisará de un sintetizador. A continuación se presentan los principales programas que pueden ser de gran apoyo en este proceso.

- *GarageBand*. Es una aplicación de Apple que permite la edición de sonido de manera profesional con una interfaz que puede ser usada incluso por novatos. Se pueden hacer mezclas y crear efectos de manera sencilla utilizando las herramientas con las que cuenta este programa.
- *Audacity*. De capacidades similares a GarageBand, esta opción de *software* libre facilita complementar la cadena de producción de videojuegos con un programa bastante completo para la edición de audio.

También existen opciones similares, tanto de paga como en *software* libre, pero estas dos representan el liderazgo en cuanto a *software* para el proceso de creación y edición de sonido.

Conclusiones

Posteriormente, se pueden elegir juegos en 2D que son más sencillos o en 3D. En el caso de este capítulo se han introducido técnicas para desarrollar escenarios y personajes en 3D por el realismo que pueden ofrecer para aplicaciones de juegos serios, pero también por la existencia de herramientas que pueden acelerar su proceso de desarrollo. Cabe mencionar que del momento que esta obra inició, la tecnología ha seguido evolucionando, pero los principios que presentamos siguen siendo vigentes debido a que nos hemos centrado en los procesos de desarrollo. Un caso es, por ejemplo, SketchUp, que inicialmente era un *software* de la empresa Google y que hoy es una empresa independiente. Sin embargo, este *software* es hoy un referente para los desarrolladores de videojuegos, sigue teniendo su versión *freeware* que permite desarrollar modelos e integrar rápidamente resultados para escenas. Con un modelo de licencia accesible, SketchUp permite hacer procesos más avanzados y detallados como elemento de producción ya profesional.

En el tema de modelado 3D para manejo de texturas, las técnicas mencionadas son tradicionales en el mundo de desarrollo de videojuegos e integradas

en la plataforma de Unity, que sigue sobre los mismos principios de diseño y programación como es la creación de relieves, el *Bump mapping*, y otras técnicas para optimizar el procesamiento. Todo el *software* mencionado sigue teniendo entonces vigencia pero con muchas mejoras y mayor soporte de formatos de intercambio de archivos.

Sobre Unity3D, particularmente un tema que se revisa es el manejo de *scripts* para poder incluir los efectos sonoros y visuales de una escena, y finalmente integrar los personajes y el control de sus animaciones. Existen *software* como Evolver que ha sido adquirido por la empresa Autodesk y que ahora forma parte de los componentes que ofrece. Así como Evolver, en similar forma pero de un bajo costo, sigue siendo una opción el *software* de Poser Pro, que ha seguido actualizando y acelerando el proceso de creación de personajes animados a integrar en videojuegos y motores como Unity3D. También proyectos como MakeHuman han seguido una evolución importante y ahora ofrecen personajes con diferentes tipos de ropa y facilidad de poder animarlos e integrarlos a Unity3D. Una de las ventajas de los *software* presentados, incluyendo a Blender, es la posibilidad de desarrollar sobre ellos *scripts* de código que permitan automatizar procesos para que la cadena de producción de videojuegos sea aún más eficiente los los fines que se estén identificando.

Se vuelve importante considerar los consejos de este capítulo pues todos están enfocados en optimizar los procesos y técnicas para que los juegos serios sean más realistas y con un menor consumo de costo de procesamiento por parte de las tarjetas gráficas, sobre todo ahora que los juegos se pueden ejecutar en dispositivos móviles como *smartphones* y tabletas, donde la prioridad es poco procesamiento para ahorrar energía pero con efectos muy realistas.

Como conclusión final, todo lo visto hasta ahora está en los elementos tradicionales para el desarrollo y producción de videojuegos serios donde hemos identificado las técnicas básicas a integrar. También se ha presentado un panorama general de desarrollo del sector y esto deberá, a quienes tengan interés de emprender en este segmento de industria, poder plantear una estrategia organizacional y de procesos. En los capítulos siguientes, se verán temas de frontera que impactan desde una perspectiva tecnológica en videojuegos serios para simulación de grandes cantidades de personajes, su comportamiento, su gestión en bases de datos y finalmente su intercomunicación en varias computadoras a través de redes como internet.

CAPÍTULO 3

Integración de multitudes de personajes artificiales a los videojuegos

MARTHA ELENA ZAVALA VILLA

VÍCTOR MANUEL LARIOS ROSILLO

Introducción

La creación de personajes artificiales en los videojuegos se considera de gran relevancia debido a la interacción que genera con los jugadores y la evocación de competencia en función de los objetivos del juego (Singhal y Zyda, 1999). Asimismo, esto ayuda a generar una mayor inmersión en la historia del juego.

Es importante mencionar que si esta tarea no se automatiza podría resultar altamente consumidora de tiempo del equipo de trabajo artístico en cuanto a modelado 3D de cada personaje, su texturizado, su adhesión de estructura ósea y la generación de las animaciones correspondientes, tal como se explicó en el capítulo anterior.

Con los avances tecnológicos que existen actualmente, tanto desarrolladores como usuarios están siempre en la búsqueda de generar y obtener personajes de alta calidad visual, que consuman el mínimo de recursos en términos de conteo poligonal, procesamiento y memoria (Ben Azouz, Rioux y Lepage, 2002). Aunado a esto, hay que aclarar que los videojuegos ya no solo se emplazan en consolas o computadoras, sino que también ya se pueden reproducir en dispositivos móviles,¹ mismos que, en términos de procesa-

¹ Dispositivos que van desde los teléfonos inteligentes como el iPhone de Apple hasta las tabletas táctiles que cuentan con capacidades gráficas no despreciables, pero no comparables con las consolas o computadoras de escritorio.

miento y memoria, tienen algunas restricciones al cuidar, entre otros aspectos, el consumo de energía.

Para que un personaje artificial tenga una apariencia más realista, es necesario considerar varios aspectos de la fisionomía del cuerpo humano, así como la definición de los parámetros que determinarán las características de su representación para poder llegar del modelado a la animación (Bastioni, Re y Misra, 2008). Lo anterior requiere incluir un conjunto de consideraciones de morforlogía del personaje, entre las que destacan generar varias simulaciones para las animaciones, establecer los parámetros de control de personaje y los algoritmos para lanzar las animaciones y movimiento, generar las texturas de la piel y ropa y trabajar las fuentes de iluminación del personaje, lo que convierte su ejecución en una tarea compleja (Sinha *et al.*, 2006).

Es en este sentido que la necesidad de programas para la creación de personajes se vuelve no solo relevante para los videojuegos, sino también para las comunidades sociales virtuales, tales como *Second Life*,² donde si el personaje virtual no cumple con cierta calidad de representación es clasificado como de bajo estatus y, por ende, no se le permite el acceso a ciertas áreas de los ambientes (Ferzzola, 2006). En otras palabras, con la gran cantidad de videojuegos que ofrecen interacción con otros usuarios en red, es esencial que estos ofrezcan mecanismos de personalización de los personajes que representan a los jugadores. Por lo tanto, poder crear personajes artificiales rápidamente podría ser de gran utilidad para videojuegos donde se requiere ambientar ciudades, estadios u otros lugares donde la historia requiera de multitudes que sean recreadas tanto en su forma visual como en su comportamiento.

Este texto presenta un camino para automatizar el proceso de creación de personajes artificiales modelados en 3D, de manera que se puedan generar multitudes con un compromiso de buena calidad visual, optimización de recursos en términos del número de polígonos y que se puedan insertar en videojuegos u otros sistemas de simulación. Cabe mencionar que la propuesta solo comprende la creación en términos de modelado 3D y sin involucrarse en las animaciones o el motor de inteligencia artificial que le da un comportamiento único a cada personaje, mismas que se tratan en el siguiente capítulo.

² *Second Life* es un mundo virtual accesible y gratuito en internet que permite a los usuarios conectados a él generar su propio personaje virtual e interactuar en tiempo real con aquellos que se encuentran en línea.

Creación de personajes artificiales

El cuerpo humano es una estructura sofisticada y de organización compleja, por lo que construir y modelar personajes artificiales en 3D con forma de humanoide para videojuegos es una tarea con un nivel importante de complejidad (Brilliant, 2003).

Existen diferentes programas, aplicaciones e incluso tecnología que permiten crear personajes basados en modelos tridimensionales del cuerpo humano (Cyberware, 2010) y que por medio de interfaces gráficas de usuario posibilitan modificar la apariencia del personaje humanoide hasta obtener las características y aspecto físico visual deseados (Heras *et al.*, 2005).

Determinar los parámetros necesarios para generar uno o varios personajes artificiales es una pieza clave en el modelado del mismo, ya sea para definir su comportamiento (García, Larios y Luga, 2010) o para representar su aspecto físico (Badler *et al.*, 2000). Por su parte, la parametrización de las características que controlan la manera de modelar dichos personajes es algo muy similar a generar una lista de cualidades y atributos deseables. Esto puede verse como originar las especificaciones y requerimientos del personaje artificial que se quiere crear con base en la historia de un videojuego.

Algunos programas y aplicaciones, como Poser,³ Avatar Studio,⁴ Daz Studio,⁵ MakeHuman,⁶ entre otros, muestran ejemplos de cómo se manejan los parámetros para el modelado de personajes. Estos trabajan generalmente mediante listas de valores manipulables a través de interfaces gráficas, lo que permite al usuario recrear y modificar diferentes personajes de una manera autónoma y sencilla; aunque si lo que se desea es crear varios cientos de personajes para un juego, esto se vuelve un proceso muy manual.

Volviendo al tema de las interfaces gráficas, cabe señalar que estas proporcionan un entorno visual de interacción entre el *software* y el usuario. Tras sus botones y controles se oculta la complejidad jerárquica del modelado de los personajes sintéticos, los problemas de deformación del modelo y la colisión entre los polígonos que lo forman (Boulic *et al.*, 1995). El tiempo requerido para la generación de humanoides sintéticos dependerá del *software* o aplicación utilizados y de la habilidad del usuario.

³ Para más información, véase <http://poser.smithmicro.com/poser.html>

⁴ Para más información, véase <http://www.avatarstudio.com/>

⁵ Para más información, véase <http://www.daz3D.com/>

⁶ Para más información, véase <http://www.makehuman.org/>

Automatización en la generación de personajes

Con la finalidad de ganar tiempo en la creación de la apariencia de personajes artificiales y reducir la intervención manual del usuario, se propone lo que se denomina la “automatización” mediante la programación (Flacher y Sigaud, 2005). Lo anterior se logra al definir las características y parámetros de representación del personaje; una vez hecho esto se generan versiones de nuevos personajes con solo cambiar los rangos y medidas, lo que ahorra mucho trabajo manual (Volz *et al.*, 2007), tal como se ejemplifica en la figura 3.1.

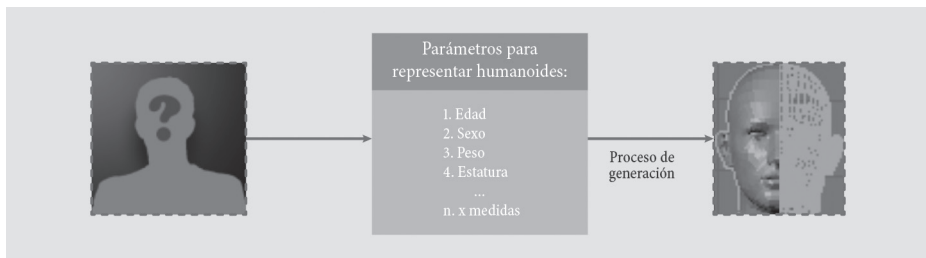


Figura 3.1. Principios de automatización en la generación de personajes artificiales

Es importante detallar que estos parámetros de representación pueden ser definidos directamente por el usuario. Así, si este quisiera caracterizar un grupo de personajes asiáticos, por ejemplo, los parámetros base a partir de los cuales generaría a la nueva población pudieran ser piel amarilla clara y ojos oblicuos (National Library of Medicine, 2012). Por lo tanto, partiendo de este principio, cada individuo de la población heredaría estas características además de las propias, obteniendo una apariencia individual que los hará diferentes entre sí.

Como ya se dijo, el cuerpo humano cuenta con cientos de particularidades que, al variar entre un cuerpo y otro, lo hacen especialmente diferente (Faller, Schünke y Schünke, 2004). Para modificar cada una de estas características es necesario parametrizarlas y otorgarles valores y medidas que permitan generar un modelo tridimensional del personaje a crear. En la figura 3.2 se puede ver una muestra de parametrización de humanoides hecha en MakeHuman, un *software* de generación de personajes artificiales en 3D con forma de humanos; un proyecto de código abierto que mediante una interfaz de usuario simplificada da como resultado un archivo que contiene una lista con todos los parámetros que originarán al humanoide, así como sus valores de modificación.

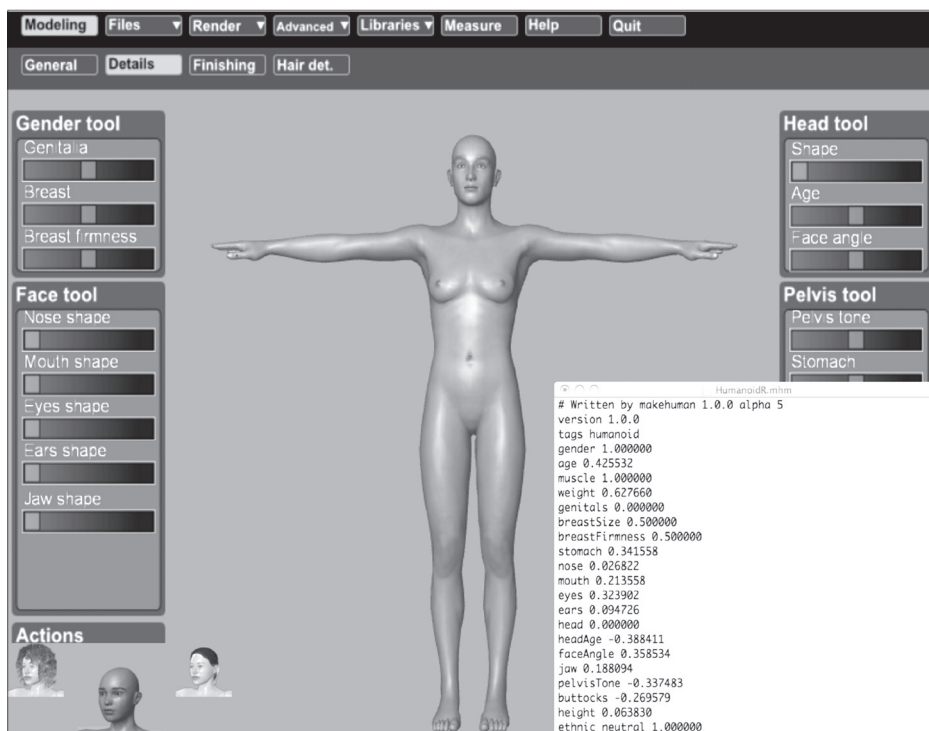


Figura 3.2. Humanoide generado en el proyecto de *software* MakeHuman y el archivo de parámetros asociados al mismo

Cuando se crea un personaje 3D, los parámetros de generación pueden ser tan variados como lo desee el usuario. Pueden ser tan simples como para representar la estatura, el sexo y la edad del personaje (MakeHuman, 2009), o tan elaborados como para pormenorizar el acomodo de la postura, la definición del comportamiento y de los sentimientos del humanoide (García, Larios y Luga, 2010). La tarea de modificar cada parámetro es responsabilidad del usuario y responde únicamente a las capacidades y necesidades de representación que desee.

Ahora bien, tomando como referencia la creación de personajes por medio de parámetros que ofrece el proyecto MakeHuman y con base en la hipótesis de que la modificación de los rangos de cada característica en los personajes dará como resultado humanoides heterogéneos, el grupo de investigación DVRmedia2 definió su propia lista de parámetros para generar una nueva población formada por cientos de personajes sintéticos heterogéneos, los cuales se explican a continuación.

Generación de multitudes artificiales heterogéneas

A simple vista, un grupo de personas luce diferente entre sí por su apariencia física, al variar su estatura, peso, forma muscular, ropa y accesorios, entre otros (McDonnell *et al.*, 2009).

En el entendido de que no es necesario tener una registro interminable de parámetros para crear una multitud de personajes artificiales (Kroemer *et al.*, 1988), la lista base de características que puede servir como una semilla para generar el conjunto de personajes diferentes corresponde a cuatro elementos esenciales: sexo, edad, peso y estatura. Podrían añadirse más parámetros, pero para poder ejemplificar se inicia con estos.

Si bien ya se dijo que las multitudes de personajes artificiales pueden ser de gran utilidad para dar más realismo al ambiente en función de la historia, también fuera del terreno de los videojuegos pueden ofrecer un espacio de investigación sobre el comportamiento de los jugadores ante las multitudes en aplicaciones de los videojuegos serios (Gutiérrez *et al.*, 2006).

Actualmente este último tipo de aplicación se emplea para estudiar el movimiento de masas en tiempo real (Houvitz, 2009), el comportamiento, la interacción dinámica, el estudio de casos emergentes y los simulacros de evacuación (Treuille, Cooper y Popovic, 2006).

Algunos ejemplos de simulación de multitudes pueden verse reflejados en herramientas de investigación como Crowdbush o Dryad. La primera es utilizada para administrar escenas con multitudes de personajes; una de sus principales características es que permite a los usuarios cargar un entorno virtual con cientos de personajes sintéticos de forma automática en tiempo real (Ulicny, Heras y Thalmann, 2004). Por otro lado, la segunda gestiona la generación de objetos tridimensionales como árboles (Talton, 2009) y edificios (Chaudhuri *et al.*, 2008) para poblar diferentes entornos virtuales.

Algoritmos genéticos como medio de multiplicación de personajes artificiales

Los algoritmos genéticos (AG) son utilizados para encontrar y aproximar soluciones óptimas que imitan la evolución biológica y genética como estrategia para la resolución de diversos problemas (Davis y Mitchell, 1991); analizan y proporcionan un conjunto de medidas que permiten evaluar cada alternativa de candidatos con el objetivo de mejorar la generación anterior y producir una nueva población aleatoria.

En la década de los sesenta, John Holland (1962) utilizó los AG como sistemas de adaptación y propuso la mutación y combinación de características para obtener nuevas y variadas soluciones. Asimismo, en 1965, Ingo Rechenberg publicó un programa titulado *Evolutionary strategies for technical innovation*, el cual imitaba la evolución biológica para resolver diversos problemas. Y un año después, Fogel, Owens y Walsh (1966), basados en los AG, programaron una máquina para que generara, de manera artificial, mutaciones de células al azar y seleccionara la mejor solución para la siguiente reproducción de células.

En este sentido, sustentados en los principios básicos de selección, combinación y mutación de los AG, hipotéticamente al someter una lista de parámetros para formar la nueva multitud a los algoritmos genéticos, se obtendrá una nueva lista de características a partir de las cuales se generará una multitud heterogénea.

Problemática en la generación de multitudes artificiales heterogéneas

Gracias a los avances tecnológicos, los entornos de videojuegos 3D actuales permiten al usuario la generación y control de los movimientos de su propia representación de avatares⁷ (Keibler, 2007), mismos que se pueden producir de manera automática mediante *software* (MakeHuman, 2009), imágenes (Takahashi *et al.*, 2008) o interfaces gráficas de usuario; sin embargo, no todos los sistemas permiten hacer uso de más de un personaje por usuario ni exportar y utilizar el modelo tridimensional en diferentes ambientes (Volz *et al.*, 2007).

Así pues, al utilizar cualquiera de las aplicaciones anteriores es posible generar poblaciones de personajes artificiales: copiando y multiplicando el modelo base (Thalmann *et al.*, 2004). No obstante, hay que hacer especial énfasis en que el verdadero reto es crear multitudes heterogéneas, todo ello a través de la representación de cada personaje perteneciente a la población, del comportamiento y la apariencia individuales.

El primer paso para lograr lo anterior es iniciar con una lista de características principales a partir de la cual nacerá la nueva población, es decir, generar una semilla de parámetros que serán modificados en cada personaje.

⁷ Entendiendo al avatar como el modelo 3D que representa al usuario o jugador en la aplicación y que en principio debe tener una forma de humanode o corresponde a un personaje de la historia del juego.

Semilla base de generación de multitudes artificiales

La semilla de generación de multitudes actúa precisamente como una semilla de reproducción; contiene los parámetros y características principales de representación de los personajes sintéticos, y con ayuda de los algoritmos genéticos y los parámetros de la semilla se logra generar multitudes heterogéneas de personajes.

Lo primero que hay que hacer en esta etapa es delimitar los parámetros de representación para generar a los personajes a través de la semilla, la cual estará formada por los parámetros sexo, edad, peso y estatura. Enseguida se utilizan los AG para replicar los parámetros de la semilla y con el apoyo de los principios básicos de selección, mutación y reproducción de los algoritmos se obtendrá la heterogeneidad de los personajes que formarán a la multitud. Cada parámetro debe tener un rango de valores mínimo y máximo de modificación. Dichos rangos oscilan entre cero y uno en valores flotantes, de acuerdo con la estimación deseada de representación para el personaje, siempre apegados a los valores de transformación que utiliza el proyecto MakeHuman (Bastioni, Re y Misra, 2008).

Una vez llevado a cabo lo anterior, el algoritmo alterará la composición genética de las listas de parámetros de las generaciones subsecuentes durante la reproducción, lo que dará como resultado una generación aleatoria de parámetros para formar a la nueva multitud (véase figura 3.1).

En síntesis, se puede decir que el trabajo de los AG es mezclar y crear mutaciones en los parámetros de la semilla a través del apareamiento o de la mezcla del parámetros de dos o más personajes generando una multitud a partir de dos listas de parámetros diferentes. Cuando ya se hizo la mezcla y se obtiene la mutación, los AG se encargan de seleccionar los parámetros más aptos para sobrevivir; los reproduce, los modifica y por medio de una función de distancia (FD)—que no es más que la generalización de los parámetros y la diferencia entre los rangos de valores de cada personaje— evita que se repita la misma combinación de parámetros entre las semillas de los personajes de la población.

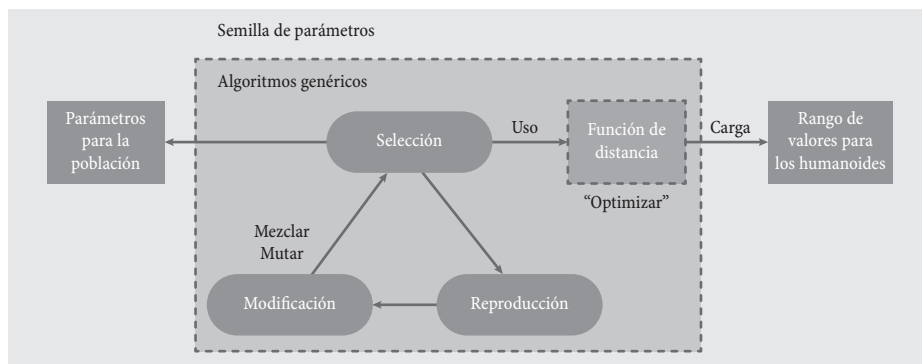


Figura 3.3. Proceso que sigue el algoritmo genético para crear una población heterogénea

Para obtener la diversidad deseada en la población, la FD mide la distancia entre los parámetros de la semilla y los parámetros de la solución óptima seleccionada por el algoritmo con la fórmula que sigue:

$$\text{distancia} = \|\sum_{li \in [\text{parámetros}]} (X[li]) - Y[li]\|$$

Donde la distancia entre los dos puntos puede expresarse en términos de parámetros como:

1. $x[li]$ representa los parámetros del humanoide.
2. $y[li]$ representa los parámetros del humanoide.

$$\text{distancia} = \|(X[\text{sexo}] - Y[\text{sexo}])^2 + (X[\text{edad}] - Y[\text{edad}])^2\|$$

Si se quiere determinar el valor de los parámetros que conformarán la edad y el sexo que tendrá la multitud, el usuario deberá establecer valores mínimos y máximos de representación. Como ejemplo se ponen los siguientes valores:

- Sexo del personaje 1: 0.20
- Sexo del personaje 2: 0.50
- Edad del personaje 1: 0.34
- Edad del personaje 2: 0.50

Consecuentemente, con base en estos parámetros, la FD calcula la distancia entre ellos y la solución proporcionada por el algoritmo, lo que resulta en el valor de generación para los personajes 1 y 2, tal como se observa en la fórmula siguiente:

$$\text{distancia} = \|(X[0.2] - Y[0.5])^2 + (X_1[0.34] - Y[0.5])^2\|$$

La distancia es: 0.639999999997

Esto significa que existe una distancia de 0.64 entre el parámetro óptimo seleccionado por el AG y el parámetro de la semilla que formará a la población. El algoritmo realiza todos los cálculos y se detiene cuando el criterio óptimo es alcanzado.

Pruebas de generación de multitudes formadas por cientos de personajes heterogéneos

De acuerdo con los principios de generación de poblaciones, los pasos necesarios para crear una multitud son estos:

1. Generar la semilla de parámetros a partir de la cual se determinaran las características de representación de la multitud.
2. Definir el número de personajes que van a conformar a la población.
3. Utilizar los algoritmos genéticos para mezclar los parámetros y obtener valores de representación diferentes en cada personaje.
4. Utilizar la función de distancia para evitar que los personajes sean iguales.

Al seguir estas indicaciones, cualquier usuario con poca o nula experiencia en ambientes virtuales podrá generar una población heterogénea a partir de una semilla de parámetros y poblar un ambiente virtual en tiempo real (véase figura 3.2).

Ahora bien, debido a que MakeHuman⁸ ya cuenta con un modo particular de determinar y valorizar los rangos en los parámetros para representar humanoides (Bastioni, Re y Misra, 2008), el presente trabajo toma como ejemplo la oscilación entre los parámetros cero y uno y se enfoca en utilizar los algoritmos genéticos para modificar y variar dichos valores en la multitud a representar.

Para realizar la primera prueba de generación de multitudes se utilizó una semilla formada por cuatro parámetros, y con ello se generó una población de cinco semillas heterogéneas. El tiempo que le tomó a la computadora procesar y generar a la nueva multitud de semillas fue de 174 milisegundos.

Como se observa en la figura 3.3, los rangos en los parámetros de la semilla se modifican, por lo que el resultado tendrá variaciones físicas considerables en la representación paramétrica de cada personaje de la multitud. De esta manera, en la siguiente figura se ejemplifica lo anterior, donde a partir de una semilla formada por el parámetro *edad*, el cual inicia con un valor de 0.5, al pasar a través del algoritmo genético crea una multitud de cinco humanoides con edades variables entre niños, adolescentes y adultos mayores.

⁸ Si requiere más información acerca de cómo obtener los valores flotantes de cada parámetro, véanse los diferentes documentos que aparecen en la pagina principal de MakeHuman, <http://www.makehuman.org/>

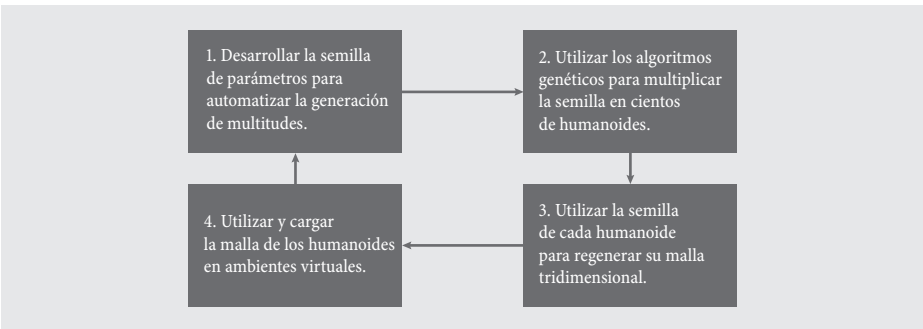


Figura 3.4. Hipótesis de generación de multitudes utilizando semillas de parámetros y algoritmos genéticos

Es decir, para representar dentro de la multitud a niños de entre cero y 10 años, el valor del parámetro oscilará entre 0.0 y 0.35; para representar adolescentes de 12 a 20 años, el valor será entre 0.4 y 0.65; y para los adultos y adultos mayores, los valores del parámetro serán entre 0.7 y 1.

```
Parámetros de la semilla:  
Sexo: 0.9  
Edad: 0.5  
Peso: 0.2  
Estatura: 0.8  
  
Iniciando la evolución con: 5 humanoides.  
Tiempo aproximado de generación: 3ms, tiempo total 174ms.  
  
Diferencias físicas entre la multitud:  
Sexo: 0.7963824064129096  
Edad: 0.3938174342435975  
Peso: 0.6388236199959538  
Estatura: 0.799207899644197
```

Figura 3.5. Captura de pantalla del sistema DVRMedia2 al generar multitudes de semillas para la formación de humanoides

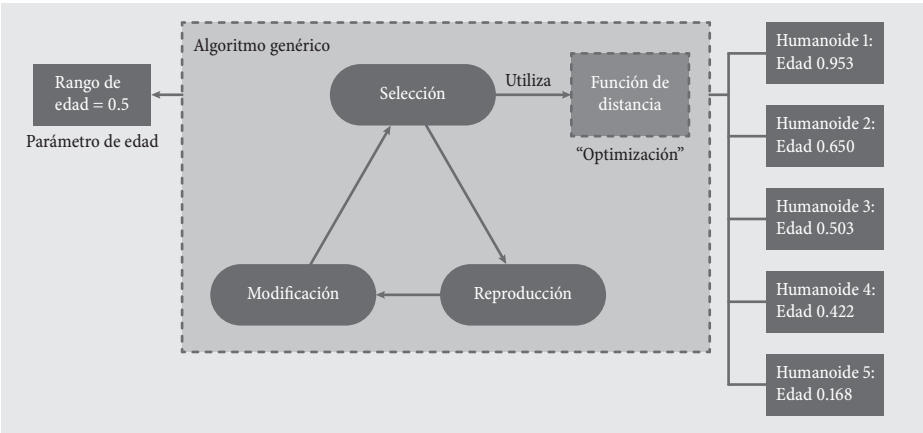


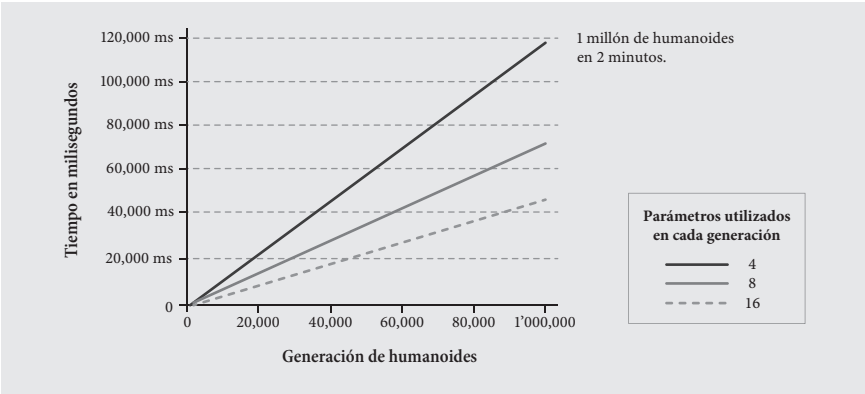
Figura 3.6. Modificación del rango de edad por medio del algoritmo genético en una población de cinco humanoides

Es importante medir el tiempo de generación de semillas para cada uno de los personajes en razón de que la multitud estará formada por cientos de personajes artificiales. Asimismo, esta semilla puede estar constituida por diferentes parámetros según lo que se desee representar en la multitud y se puede conformar por cuatro, ocho y hasta 16 parámetros (véase cuadro 3.1).

Cuadro 3.1. Posibles parámetros de selección para crear la semilla de generación de humanoides		
4 parámetros	8 parámetros	6 parámetros
Sexo	Sexo	Sexo
Edad	Edad	Edad
Peso	Peso	Peso
Estatura	Estatura	Estatura
	Forma del cráneo	Forma del cráneo
	Orejas	Orejas
	Nariz	Nariz
	Boca	Boca
		Cuello
		Cabello
		Ojos
		Color de ojos
		Cintura
		Cadera
		Piernas
		Pies

En lo que respecta a las pruebas de generación de las semillas de la multitud, hay que especificar que fueron realizadas en una computadora MacBook Pro, con las siguientes características: microprocesador Intel Core Duo @2.4 GHZ, 4 GB de RAM, JVM 1.6.017 y OSX 10.6.3.

Los resultados de tiempo de generación para cada semilla se pueden consultar en la gráfica 3.1. Ahí se muestra el tiempo que tarda una computadora común en modificar los parámetros de cada semilla que formará a los humanoides de la multitud.



Gráfica 3.1. Medida de tiempo para generar poblaciones de uno a un millón de personajes artificiales con variación de hasta 16 parámetros

Es importante referir que con la medición de los tiempos de creación de las semillas se facilita la distribución de los recursos de cómputo disponibles en tiempo y espacio, siempre considerando el punto de vista del usuario en relación con el ambiente virtual que se va a representar en el videojuego (véase cuadro 3.2).

Cuadro 3.2. Medidas de tiempo para generar poblaciones de un millón de humanoides con variación de hasta 16 parámetros			
Humanoides	4 parámetros	8 parámetros	16 parámetros
1	4 ms	6 ms	7 ms
10	12 ms	18 ms	28 ms
100	213 ms	228 ms	247 ms
1,000	440 ms	476 ms	488 ms
10,000	955 ms	1.167 seg	1.66 seg
100,000	5.108 seg	7.629 seg	12.172 seg
1'000,000	46.253 seg	1.181 min	1.901 min

Como se puede apreciar en el cuadro 3.2, para generar una multitud formada por un millón de semillas heterogéneas, una máquina común tarda aproximadamente dos minutos, lo cual, si se tienen en cuenta el número de personajes que conforman la multitud, es un tiempo relativamente corto. El desarrollo de la malla tridimensional de cada semilla, así como sus texturas y su carga en el ambiente, pueden tomar más tiempo por el número de polígonos que se van a generar y representar para cada personaje.

Adicionalmente, dado que MakeHuman es un sistema de código abierto, se aprovechan sus módulos para generar las mallas tridimensionales de los humanoides a través de las semillas generadas con ayuda de los algoritmos genéticos. De esta forma, al utilizar los *plugins* del proyecto MakeHuman escritos en lenguaje Python, se obliga al *software* a leer directamente los parámetros de la semilla y a que con esos datos genere la malla tridimensional del personaje en formato COLLADA.

En la figura 3.7 se detalla la estructura interna del proyecto MakeHuman y el diagrama de cómo trabajan los módulos para el desarrollo de las mallas tridimensionales de los personajes.

Cabe mencionar que las pruebas de generación de las mallas tridimensionales se realizaron en la misma máquina donde se crearon las semillas de la multitud. Los resultados de tiempo en el desarrollo de las mallas de los humanoides se especifican en el cuadro 3.3.

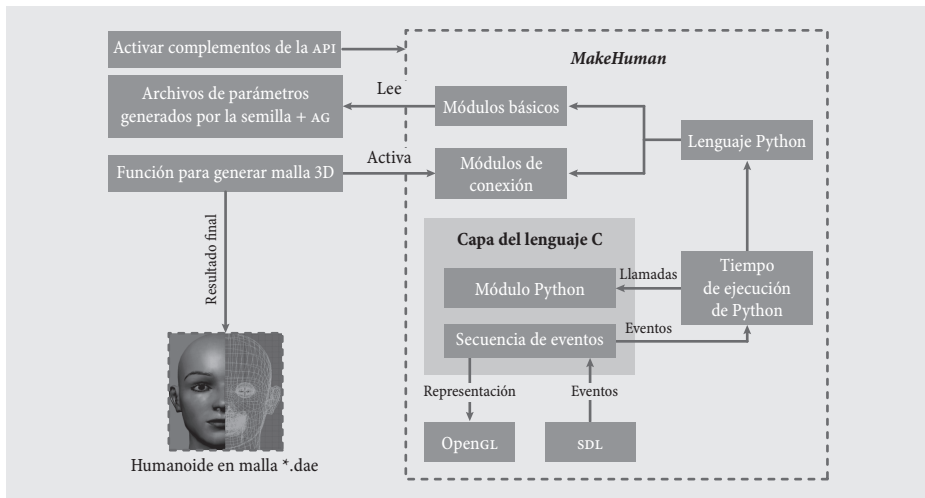


Figura 3.7. Estructura interna del *software* MakeHuman

Cuadro 3.3. Generación de mallas tridimensionales mediante una máquina común con dos núcleos			
Humanoides	Segundos	Minutos	Horas
1	5.9	0.10	0.00
10	20.41	0.34	0.01
100	139.82	2.33	0.04
1,000	1,368.18	22.80	0.38
10,000	19,470.67	324.51	5.41

Tal como se manifiesta, una computadora común que trabaja con dos procesadores a la vez tarda cerca de seis horas para generar una multitud formada por 10 mil humanoides. Evidentemente estos tiempos no son eficientes; sin embargo, si esta prueba se distribuye en diferentes máquinas o se realiza en clústers que contengan más de un núcleo de procesamiento, el tiempo de generación disminuirá considerablemente. Se habla, pues, de cientos de personajes tridimensionales formados por millones de polígonos sin contar la textura de la piel, la representación del cabello y su ropa, entre otros muchos elementos que se podrían representar.

Finalmente, resta decir que los personajes en malla tienen un peso aproximado de 3,5 MB de información. Por lo tanto, utilizar diferentes procesadores para generar una multitud permitirá incrementar la velocidad de procesamiento, carga y representación de los personajes dentro del ambiente virtual.

Conclusiones

Los algoritmos genéticos contienen elementos que definen el dominio del problema que se va a optimizar a través de sus principios biológicos y evolución orgánica, lo cual permite lograr variaciones considerables en los parámetros de representación de los personajes que conformarán la multitud. Entre sus principales ventajas se encuentra la automatización en la generación de multitudes, por lo que con una mínima intervención del usuario se consigue el desarrollo y generación de todos y cada uno de los personajes a representar. Lo anterior impulsa el crecimiento de simulación de multitudes con mayor número de pobladores e incrementa las posibilidades de paralelizar y desarrollar simulaciones en tiempo real.

Al respecto, es importante resaltar que en este campo se cuenta con algunos avances en proceso. Uno de ellos es la afinación de la generación de la malla tridimensional de cada uno de los personajes de la población, con la finalidad de que estos puedan ser cargados y utilizados en diferentes ambientes de videojuegos en formatos optimizados en la cantidad de polígonos, además de garantizar un esqueleto.

Asimismo, otro elemento relacionado es el desarrollo de los parámetros de comportamiento que tendrán los personajes y que serán igualmente automatizados y modificados para obtener completa heterogeneidad en la población. Es en este sentido que la automatización no solo comprende el aspecto morfológico y visual de los personajes, sino también el comportamiento que tendrá dentro del juego cuando sea controlado por un cerebro artificial.

En suma, el objetivo que se pretende alcanzar con dicha automatización es que cada personaje sea capaz de tomar sus propias decisiones, así como planificar sus propias acciones dentro del videojuego, tal como se desarrolla en el mundo real en función del guión de la historia que se esté representando. Se busca, también, entre otros aspectos, tener ambientes de videojuegos realistas, que puedan ser utilizados para entrenamientos, simulaciones u otras muchas áreas del conocimiento que beneficien a la población a prevenir problemas futuros o simplemente en un aspecto lúdico de alto realismo.

CAPÍTULO 4

Motor de inteligencia artificial

CÉSAR A. GARCÍA GARCÍA

Introducción

De la misma manera que el jugador ha tomado control de su personaje, es necesario que el sistema tome control del ambiente y de los oponentes con los que el jugador puede interactuar. Esto se logra a través de un motor de inteligencia artificial que toma las decisiones y acciones de los oponentes.

De acuerdo con John McCarthy (1958), los humanos somos malos para identificar y definir nuestros propios procesos mentales. Si ya de por sí es complicado definir el concepto de *inteligencia*, el de *inteligencia artificial* lo es aún más.

En el contexto de las ciencias de la computación, la inteligencia artificial resuelve problemas para los cuales la solución numérica o iterativa no existe o es inviable (Wise, 2004). Desde un sentido pragmático, las técnicas de inteligencia artificial solucionan problemas complejos haciendo el mejor uso posible del poder de procesamiento disponible, es decir, dirigiendo la fuerza bruta del procesador por caminos prometedores, evitando aquellos en los que no se espera encontrar la solución al problema.

Sin embargo, hay que hacer énfasis en que muchas aplicaciones de la inteligencia artificial no parecen nada inteligentes. Un problema muy común al respecto es buscar un dato en una estructura muy compleja, o en otras palabras, una proverbial aguja en un pajar. La manera iterativa de solucionar este problema consiste en revisar cada paja individualmente y verificar que no se trata de la aguja, antes de desecharla y buscar la siguiente. La técnica propuesta por la inteligencia artificial sería la aplicación de métodos no visuales, como el uso de un imán con el cual barrer el pajar. Una analogía extrema sería prender fuego al pajar para al final recoger la aguja de entre las cenizas.

Funciones de la inteligencia artificial

Para el propósito de los videojuegos, se considera que la inteligencia artificial es un *agente* actuando en un *ambiente* que se percibe a través de un sistema de sensores o sentidos artificiales (véase figura 4.1). Podría decirse que lo anterior es un modelo muy rudimentario, aunque en un sentido estricto es válido tanto para la programación de agentes inteligentes como para la robótica.

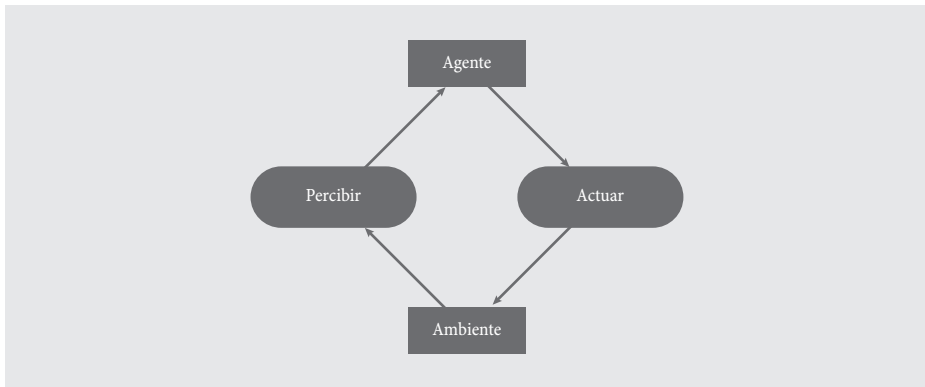


Figura 4.1. El ciclo externo de percepción-acción de un sistema inteligente

Fuente: Elaboración propia con base en Wise (2004).

Algunas de las funciones de la inteligencia artificial incluyen encontrar estados ideales —que *ganan* el juego— y mostrar el camino para llegar a ellos; identificar y reconocer objetos presentes en el ambiente; realizar la toma de decisiones de los personajes; hacer mejor uso de los recursos del juego; y, finalmente, predecir posibles estados futuros a partir del estado actual y de las acciones de los jugadores.

Búsqueda

En muchos juegos de mesa comunes, tales como el ajedrez, las damas o el go, es posible representar toda una partida como un árbol, en el cual cada nodo es un estado del tablero y sus hijos son los posibles estados a los que se puede llegar realizando un solo movimiento.

Un ejemplo podría ser una partida del tradicional juego de gato (también conocido como *tic-tac-toe*), que consiste en nueve casillas en las que dos jugadores se alternan para colocar marcas diferentes (véase figura 4.2), y el primero que logre una línea (incluyendo diagonales) de tres de sus marcas gana el juego.

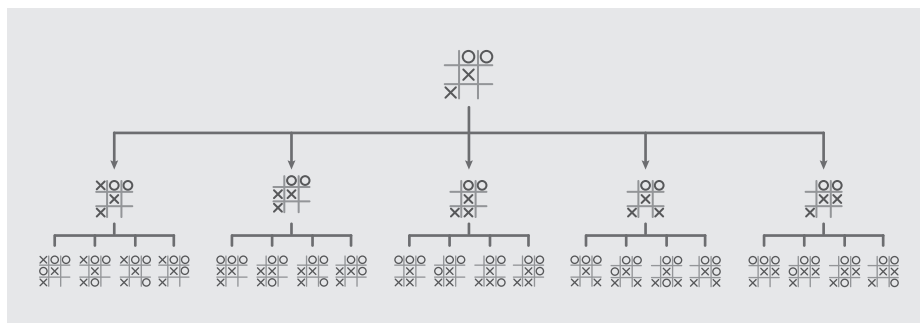


Figura 4.2. Parte del árbol de búsqueda del juego de gato

Parece sencillo, pero si se analiza, resulta lo siguiente: existen 18 estados o maneras de abrir el juego; dos posibles marcas en nueve posibles casillas. A partir de aquí el espacio de búsqueda crece exponencialmente hasta incluir las 19 mil 683 posibles configuraciones del tablero, a lo largo de 362 mil 880 rutas, de raíz a hoja, que representan todas las posibles maneras de jugar.

Clasificación

El reconocimiento de patrones, particularmente en imágenes, suele utilizar sistemas inteligentes adaptativos —como redes neuronales o algoritmos genéticos— para deducir condiciones que se repiten en circunstancias similares (Sánchez, Díaz y Bock, 2005).

Al aprender patrones de colores y texturas en las imágenes, es posible implementar ciertas funciones:

- *Reconocimiento.* Identificar si un patrón conocido se encuentra presente.
- *Clasificación.* Distinguir entre diferentes patrones conocidos presentes en la misma imagen.
- *Segmentación.* Seleccionar partes relevantes de una imagen a partir de un patrón conocido.

La figura 4.3 ilustra el uso de esta tecnología en el reconocimiento de armas ocultas en el equipaje. Este es el mismo concepto detrás de sistemas como el de Sony Playstation Move y otros similares.

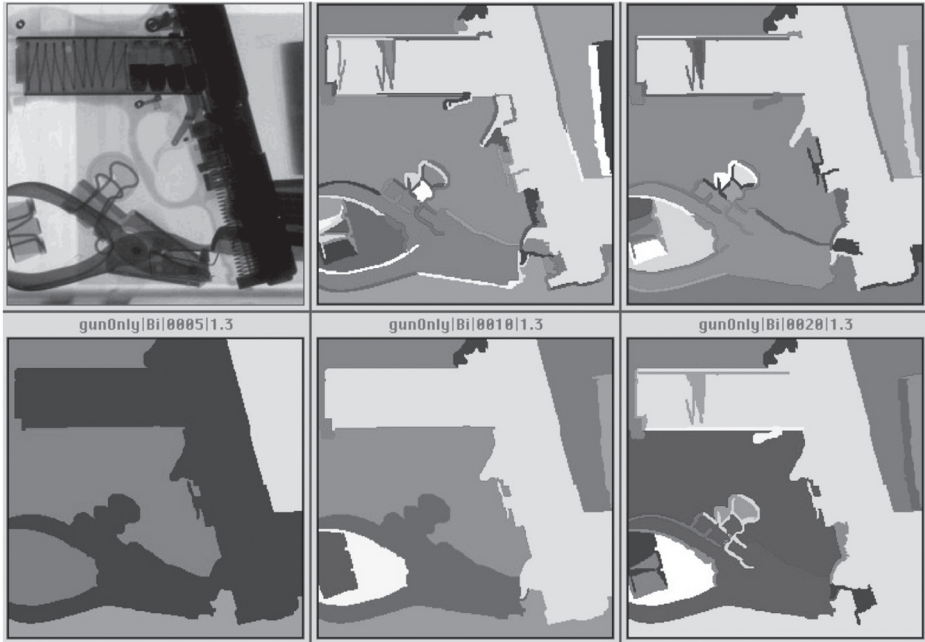


Figura 4.3. Clasificación de imágenes aplicado al reconocimiento de armas ocultas
Fuente: Cortesía de Peter Bock (2013).

Control

Un modelo más complejo para el control de acciones podría ser el de percibir-planear-actuar, que se muestra en la figura 4.4. En este modelo, el agente advierte su entorno a través de un módulo de percepción, que alimenta una representación interna del mundo, basada en esta percepción, pero complementada por conocimiento previo acerca de cómo funciona ese mundo.

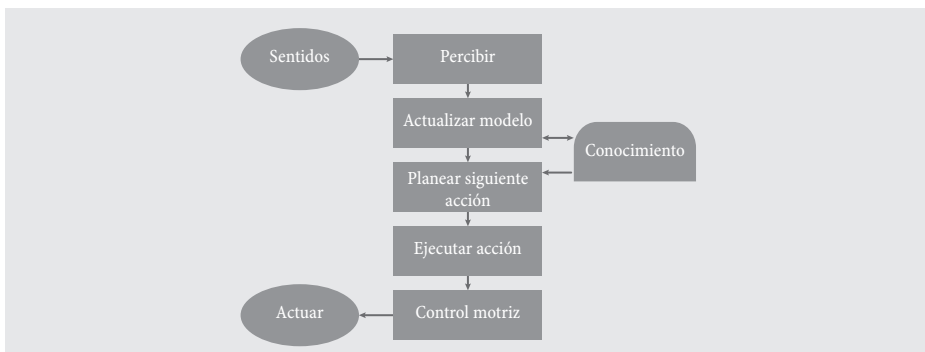


Figura 4.4. El ciclo interno de percibir-planear-actuar de un sistema inteligente
Fuente: Elaboración propia con base en Wise (2004).

Con esa información, un módulo de planeación elige la mejor acción con base en el estado del mundo; dicha acción es entonces ejecutada, y, finalmente, se activa un módulo de control motriz que realiza la animación en el entorno visual.

Heurísticas y árboles de búsqueda

Previamente se mencionó que es práctico representar el espacio del juego —todos los posibles estados del tablero y todas las posibles transiciones— como un árbol de búsqueda, ya que existen numerosos métodos o recorridos para buscar en árboles que ofrecen tiempos y complejidad algorítmica no-lineales.

A fin de poder evaluar cuál de las transiciones es más conveniente, cuál nos acerca más a ganar, es necesario asignarles pesos o costos. Con base en estos costos es posible establecer una métrica para compararlos, aun si esta métrica está apoyada en conocimiento empírico no demostrable. Este tipo de reglas o comparaciones se denominan *heurísticas*. La calidad de una heurística determina la calidad de un algoritmo de búsqueda.

Por ejemplo, una heurística sencilla es la de “*hill climbing*”, la cual se basa en la analogía de escalar una montaña: cualquier camino que te acerque a la cima, es un buen camino. Pero como todo, siempre hay un sentido opuesto, por lo que este podría no ser siempre el caso, aunque esa es precisamente la clave de las heurísticas: incluso si no son apropiadas para todos los casos, pueden serlo en muchos, con lo cual ya se consideran buenas. Así pues un algoritmo de búsqueda informada es tan bueno como su heurística, y esta es tan buena como la experiencia del desarrollador en el problema específico.

En el caso particular de juegos para dos participantes, existen algoritmos muy específicos para recorridos de árboles de juego. A continuación se presentan los algoritmos comunes minimax y *alpha-beta pruning* o poda alfa-beta.

Minimax

Este es un método de búsqueda que encuentra la ruta más prometedora en un árbol de juego al explorar una profundidad limitada. Su nombre se deriva de los objetivos conflictivos de ambos jugadores: a cada paso se busca minimizar las posibilidades de ganar del otro mientras se intenta maximizar las propias. La heurística es sencilla, simplemente se resta la ganancia potencial de *mini* (el adversario) de la ganancia potencial de *max* (el propio jugador).

Existen dos maneras de evaluar un nodo; el primero depende de su profundidad y el segundo de si es un nodo hoja. Si la búsqueda ha alcanzado la máxima

profundidad permitida o un estado sin salida, el valor del nodo es simplemente la resta de puntuaciones. Los nodos que no son hojas se evalúan en función del jugador que representan: los nodos *max* toman el valor del mejor de sus hijos, y los nodos *mini* del peor de ellos. Es aquí donde la calidad de la heurística entra en juego, ya que la manera de determinar los nodos mejor y peor (ya sea por puntaje directo, cantidad y calidad de piezas, dominio del tablero, etc.) es fundamental para el correcto funcionamiento del algoritmo.

Poda alfa-beta

Una importante mejora de este algoritmo es que esencialmente reduce el espacio de búsqueda y, por ende, el tamaño del árbol en el que se necesita buscar. Esto se logra al evaluar solamente los nodos raíz antes de entrar en un subárbol, para explorar solamente el subárbol más promisorio —el que parezca mejor para la heurística— sin necesidad de explorar los subárboles menos prometedores; de ahí el termino *poda*.

Durante una poda alfa-beta, cada decisión —y su subárbol asociado— favorece a alguno de los dos jugadores. Esta ventaja puede cambiar durante la búsqueda y se va acumulando, de tal manera que se busca maximizar la ventaja de un jugador y minimizar la del otro.

El algoritmo además mantiene dos valores: *alfa*, que representa la ventaja mínima asegurada del jugador que se busca maximizar; y *beta*, que representa la ventaja máxima asegurada del jugador a minimizar.

Por su parte, al momento de evaluar una decisión es necesario minimizar la distancia entre alfa y beta, de modo que ambos tengan ventajas similares en todo momento. En caso de que una decisión aumentara la diferencia entre las ventajas, representaría un error por parte de alguno de los jugadores, por lo que todo el subárbol se desecha. Hay que recordar que el objetivo de los árboles de decisión, en este caso, es el de representar las mejores estrategias contra los mejores oponentes, por lo que se asume que no cometerán errores.

Planificación de rutas

La función de encontrar caminos o rutas, tanto para los personajes dentro del entorno como para los diferentes estados del juego y sus transiciones, es de suma importancia en este contexto.

El objetivo principal de un sistema de planificación de rutas es guiar a un personaje para que recorra o navegue por su entorno haciendo uso de diferentes

instrucciones establecidas en la inteligencia artificial. En este sentido, la navegación basada en *grid* se crea mediante la superposición de una malla o *grid* sobre el escenario del videojuego, con la división del entorno en celdas. Además, a cada una de estas celdas se le asignan coordenadas y se indica si son transitables o no. La malla superpuesta se convierte en un grafo por medio de la creación de un nodo de abstracción de cada celda y luego se utiliza la conectividad entre los nodos o arcos para determinar la adyacencia del grafo.

Búsqueda informada con el uso de A*

Al igual que la mayoría de los planificadores de rutas, A* —pronunciado *a-star* o *a-estrella*— divide el espacio de búsqueda —para el caso de la navegación en un mapa— en celdas de tamaño más o menos uniforme, y que se agregan a un *grafo de navegación*, de forma que recorrer el grafo equivale a navegar el espacio representado en el mapa.

El algoritmo A* mantiene dos listas: una, llamada *lista abierta*, que contiene los nodos y que puede ser un siguiente paso en el camino; y otra, llamada *lista cerrada*, que son nodos que ya se han considerado y descartado. Además, opera en un ciclo que establece que mientras que no se alcance el nodo destino y existan nodos para ser considerados, se elige el nodo adyacente más probable según los valores proporcionados por la heurística. Cada vecino es visto como un posible próximo paso. A medida que cada lugar es descartado como una opción, es removido de la lista abierta y se añade a la lista cerrada. Eventualmente se alcanzará el nodo destino o no quedarán más nodos por visitar, lo que significa que no existe una ruta. El algoritmo se describe enseguida:

- En la inicialización se agrega el nodo de inicio a la lista abierta y se deja vacía la lista cerrada.
- Mientras existan posibles pasos a seguir en la lista abierta y no se ha encontrado el nodo destino:
 - a. Seleccionar el siguiente paso más probable (basados tanto en la heurística como en los costos de ruta).
 - b. Eliminarlo de la lista abierta y agregarlo a la cerrada.
 - c. Generar las heurísticas para cada vecino de la etapa:
 - Calcular el costo para llegar al vecino.
 - Si el costo es menor que el costo conocido a ese lugar, entonces se quita de la lista abierta o cerrada (ya que ahora se ha encontrado una ruta mejor).
 - Si la ubicación no está en la lista abierta o cerrada, hay que guardar los costos para esa ubicación y agregarlo a la lista abierta (esto significa que será considerado en la próxima búsqueda).
 - Registrar cómo se llega a este lugar.

El ciclo termina cuando ya se ha encontrado una ruta hacia el destino o ya no hay pasos que seguir. En caso de localizar una ruta es importante hacer un *backtrace* con los registros de cómo se llegó a cada punto, con la finalidad de determinar la ruta.

Creando la ilusión de comportamiento inteligente

Predecir el comportamiento de otras entidades presentes en nuestro ambiente es fundamental para la propia supervivencia. Según su origen, existen dos grandes clasificaciones del comportamiento: los comportamientos reactivos, los cuales se generan a partir de la percepción del ambiente; y los cognitivos, que se generan a partir del conocimiento y la intención del actor.

Para la mayoría de los videojuegos casuales, un comportamiento reactivo es suficiente. El hecho de efectuar una animación al contacto con un objeto del ambiente se considera un comportamiento de este tipo; por ejemplo, cuando el jugador es alcanzado por un oponente y se termina el juego. Pero, a pesar de lo anterior, para juegos más complejos es necesario utilizar comportamientos que reaccionen a cambios en el ambiente, tomando en cuenta la intencionalidad del actor. La creación de comportamientos realistas requieren además la representación y manipulación de la información percibida (Sao y Terzopoulos).

Planificación de acciones basada en metas

La técnica GOAP (*Goal-Oriented Action Planning*) propuesta por Jeff Orkin (2003, 2004) y popularizada en el juego F.E.A.R.¹ (Orkin, 2006) es una de las más utilizadas para juegos complejos. Esta consiste en crear un grafo que represente el juego, donde cada nodo es un estado del mundo o *world state* y los arcos representan las acciones necesarias para llevar al mundo de un estado a otro. Un mecanismo de búsqueda informada —tal como A^* — se aplica en el grafo para determinar la mejor secuencia de acciones para llevar al mundo al estado deseado o *meta*. Por lo tanto, la ruta que efectivamente recorre el grafo desde el estado inicial al estado deseado es el mejor plan para alcanzar la meta.

A saber, una meta es una condición o estado del mundo al que el actor pretende llegar; un ejemplo para alcanzarla podría ser *asegurar el área*, que implicaría que el actor debe eliminar cualquier amenaza dentro de su capacidad, es decir, utilizando solamente las acciones disponibles.

¹ Para más información, véase <https://www.whatisfear.com/>

Las acciones son, como su nombre lo indica, acciones atómicas que el actor puede tomar dentro del ambiente, tales como abrir una puerta, cargar o disparar un arma, cambiar de posición del cuerpo, dar un solo paso, entre otras. Es importante que las acciones sean atómicas y sus animaciones tengan aproximadamente la misma duración. De esta manera se consigue un comportamiento fluido que da la ilusión de que todo sucede al mismo tiempo.

Multitudes y comportamiento grupal

A partir del trabajo seminal de Reynolds (raig, 1987), con el desarrollo de parvadas de aves artificiales, se ha avanzado mucho en la simulación de multitudes a través de la agregación de comportamientos originados en la percepción y acción de cada individuo. A este fenómeno se le conoce como *emergencia*.

Paralelismo e inteligencia artificial distribuida

Para finalizar es necesario señalar que algunos sistemas de inteligencia artificial trabajan mejor *offline*, es decir que tienen tiempo de realizar su procesamiento tras bambalinas, lo cual resulta particularmente útil cuando existen espacios de búsqueda muy grandes. Pero, ¿qué pasa cuando es necesario tomar una decisión en tiempo real? Es aquí donde técnicas ya conocidas como el paralelismo y la programación multihilos entran en acción.

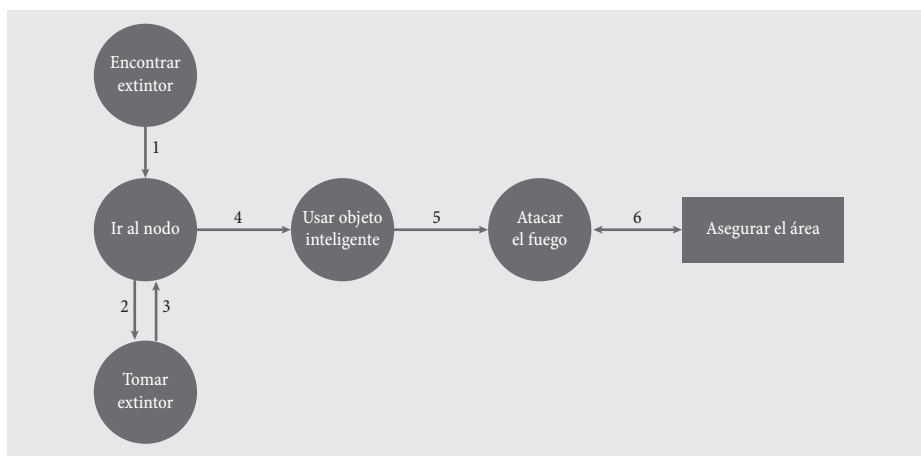


Figura 4.5. El plan generado por un motor de inteligencia artificial enfocado en combate de incendios

Existen problemas que por su propia naturaleza son buenos candidatos para la paralelización mediante la ya conocida técnica de “divide y vencerás”, que consiste fundamentalmente en encontrar aquellas partes del problema que son independientes unas de otras, o mejor aún, que son recursivas.

Ya que se identifican estas partes se puede proceder a codificar una solución que aproveche estas características, siendo la más común el uso de multihilos. Cabe señalar que esta técnica utiliza código idéntico que, al ejecutarse en paralelo con diferentes datos, devuelve una parte de la solución total.

Esta técnica es además popular porque puede funcionar en sistemas de escritorio de un solo CPU, más comunes en escuelas y oficinas. Su funcionamiento se debe a que cada hilo solicita tiempo de procesador al *scheduler* o planificador del sistema operativo, que lo asigna de manera independiente, generando la ilusión de procesamiento paralelo en tiempo real a través de la rápida multiplexación de estos hilos en un solo procesador.

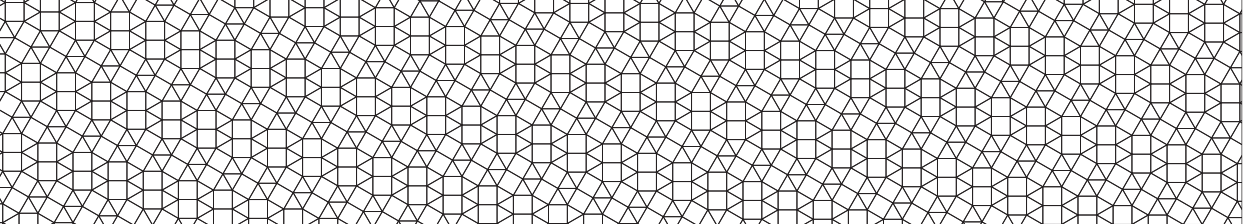
Conclusiones

A lo largo de este capítulo se han descrito diferentes técnicas de inteligencia artificial y su aplicación al desarrollo de videojuegos.

Si bien las técnicas de inteligencia artificial pueden ser intimidantes a primera vista y su implementación y relación con el ambiente gráfico no son una tarea trivial, esto no debería desanimar al lector para su exploración y explotación. Una vez dominadas e implementadas estas técnicas, las tareas exhaustivas de expandir y recorrer árboles de búsqueda, buscar rutas óptimas en el ambiente gráfico y generar planes que lleven al juego de un estado a otro, se realizan de manera algorítmica, lo que hace que para el usuario parezca algo instantáneo.

Técnicas avanzadas de inteligencia artificial pueden ser utilizadas para volver el juego más atractivo. Por ejemplo, el aprendizaje automático puede generar retos apropiados a la habilidad de cada jugador, aumentando el valor de *replay* del juego. La generación procedural de contenido o PCG mediante algoritmos genéticos o sistemas-L puede contribuir a la creación de juegos infinitos, donde cada partida o nivel del mismo es completamente aleatoria, aumentando el interés del jugador en seguir jugando.

Aunque lo más llamativo de un videojuego puede radicar en el aspecto gráfico y de narrativa, la inteligencia artificial subyacente contribuye en gran medida al realismo y a mejorar la jugabilidad y, por consecuencia, la experiencia del jugador.



CAPÍTULO 5

Integración de bases de datos en videojuegos

LAURA TORRES LÓPEZ

Introducción

Los elementos gráficos descritos en los capítulos previos, tales como ambientes virtuales, personajes y su comportamiento, darán lugar a la generación de información, la cual debe ser organizada y manipulada de manera que propicie coherencia y persistencia durante la ejecución del videojuego, principalmente con el fin de que el usuario perciba una secuencia lógica que tenga sentido y que le permita continuar interesado en el mismo.

En los sistemas de información los datos se organizan tradicionalmente mediante la tecnología de bases de datos, la cual parte de fundamentos fuertemente probados que, con las debidas consideraciones en las exigentes características de los videojuegos, pueden ser adaptados para ser el soporte de información adecuado.

Fundamentos

Las estructuras de datos y su acceso de manera eficiente

Los videojuegos se componen de diferentes estructuras que contienen datos de manera organizada, de forma tal que en su conjunto sea útiles para su ejecución. Las escenas, por ejemplo, están compuestas de objetos que tienen una representación 3D. Asimismo, dichos objetos pueden ser estáticos o dinámicos. Los estáticos son aquellos objetos que no tendrán cambios importantes en el tiempo, como los

edificios, la vegetación y parte de la conformación del paisaje. Los dinámicos, por su parte, están comprendidos por objetos con una dinámica particular que cambia en función de las acciones y del tiempo tales como, los personajes artificiales o los automóviles. También los datos de eventos en el mismo sistema, como puede ser el marcador del jugador, la cantidad de poderes u objetos que está cargando consigo, requieren de una representación mediante una estructura de datos.

En la actualidad, gracias a las grandes capacidades de los equipos de procesamiento y a las tarjetas gráficas, los escenarios y los personajes han ido adquiriendo un alto nivel de complejidad, lo que da mayor realismo al juego. Cabe decir que para lograrlo se requieren formas eficientes de acceder y filtrar datos, con el propósito de optimizar el rendimiento de los recursos de procesamiento y memoria. En un videojuego con una escena que se lleva a cabo en una ciudad, no se cargan todos los objetos que la componen, sino solo una parte; así, según el desplazamiento del jugador, se irán cargando dinámicamente las otras partes de la misma.

En consecuencia, se puede asegurar que las bases de datos son de gran utilidad ante grandes volúmenes de información en un videojuego, ya que logran mantener una organización en estos. Son una parte esencial de la programación de juegos. Establecen la forma más eficiente de almacenamiento; en la medida que se ejecuten rápidamente, será posible incluir mejores características (Pentor, 2002).

En función de lo anterior, los videojuegos se clasifican en *persistentes* y *no persistentes*. Los persistentes son aquellos que almacenan la información generada y el estado del juego. Estos tienen mucho en común con las bases de datos tradicionales, puesto que juegan un papel central en el desarrollo del juego; un ejemplo de este tipo son los videojuegos masivos multiusuario (White *et al.*, 2007). De manera ilustrativa, se puede imaginar un juego que propone una batalla entre miles de jugadores, incluyendo personajes artificiales, en donde es necesario que la representación de estos, así como las modificaciones que operen sobre la escena del juego, se vayan guardando y actualizando en las terminales en red de todos los participantes, de forma que la batalla cuente con un escenario inicial en cierto estado, y al final termine con uno completamente diferente. En este sentido, guardar estos estados es lo que se denomina *persistencia*; con ello todos los participantes pueden estar en continua conexión y desconexión de red con la escena, y siempre será necesario tener el último estado o el vigente de la simulación de la batalla.

Los videojuegos no persistentes son aquellos en donde el jugador no puede guardar la información generada durante el juego; si sale de este se pierde su estado y se asignarán los valores iniciales si vuelve a entrar (White *et al.*, 2007). Sin embargo, aun en estos casos, si la escena del juego es de gran complejidad, se requiere de un manejo y filtrado intensivo de los objetos que lo componen, así como de los eventos, lo que conlleva nuevamente a buscar una solución sobre las bases de datos.

Bases de datos relacionales como una opción para la gestión de la información en videojuegos

Los sistemas manejadores de bases de datos (DBMS —Database Management System—) pueden describirse como el *software* que proporciona los servicios básicos requeridos para la organización y mantenimiento de la información, entre los cuales se encuentran el movimiento de datos, el manejo de transacciones, el soporte de lenguaje de consulta, el control de accesos y las tareas de recuperación de información (Oppel, 2004). Adicionalmente, hay que señalar que un sistema de bases de datos distribuido (DDBS) administra una o más bases de datos relacionadas lógicamente, a través de una red (Bonifati *et al.*, 2008).

En cuanto al lenguaje de las bases de datos relacionales, estas generalmente utilizan SQL (Structured Query Language) como el lenguaje universal de consulta, el cual ha tenido una gran aceptación como estándar en diferentes manejadores. El motor de búsqueda de un DBMS es el encargado de responder a las peticiones formuladas en SQL. En el caso de XML, este es un lenguaje estándar para representación e intercambio de datos estructurados en aplicaciones web (Groff, Weinberh y Oppel, 2009). Otros DBMS son Oracle, DB2 de IBM y MySQL.

Aunado a lo anterior, SUN desarrolló JDBC (Java Database Connectivity) para realizar conexiones entre los desarrollos hechos en Java y las bases de datos que manejan SQL y J2EE para las aplicaciones web; sin embargo, este no es el único lenguaje. En el caso de ambientes como Unity, las bases de datos pueden ser invocadas por otros lenguajes como JavaScript, C#, Python, etcétera. Simplemente se trata de contar con una librería que tenga implementadas las funciones para poder conectarse a la base de datos y operar mediante el lenguaje SQL o alguna variante del mismo, y a los accesos lectura, escritura, borrado y modificación de la información contenida del videojuego en cuestión.

Formas de estructurar los datos en los videojuegos

La estructura básica de un videojuego consiste en un solo dispositivo independiente, esto es, que no está conectado en red con otros, como puede ocurrir con una consola o computadora personal, en donde el usuario puede almacenar, o no, el estado del ambiente en una base de datos local (véase figura 5.1).

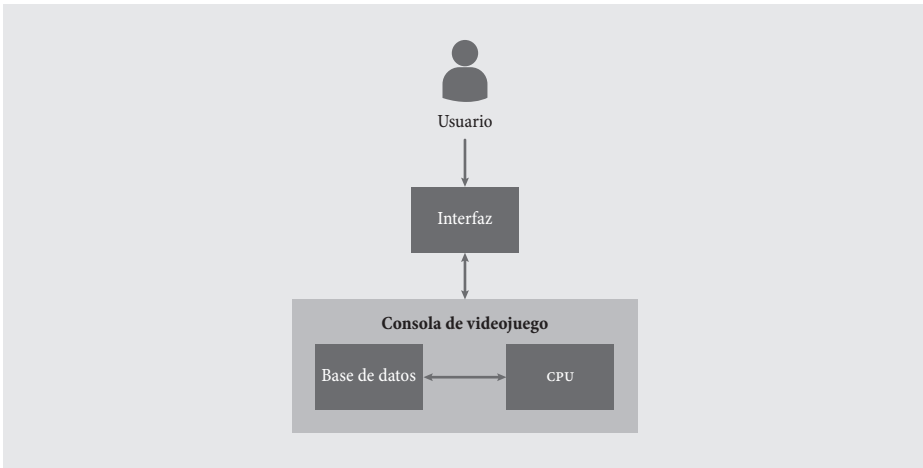


Figura 5.1. Videojuego en una consola independiente

Los videojuegos interactivos en línea, es decir, los que son accedidos por varios jugadores a la vez, requerirán operar y almacenar las modificaciones de la escena derivadas de la interacción entre los mismos. De acuerdo con la plataforma de la red utilizada, la mayoría de los juegos en línea se basan en el modelo cliente-servidor. El servidor es responsable de la interacción entre los múltiples objetos del mundo virtual, así como la comunicación con los clientes. Asimismo, los requerimientos computacionales del servidor son proporcionales al tamaño del mundo y el número de clientes conectados concurrentemente (Kumar *et al.*, 2008) (véase figura 5.2).

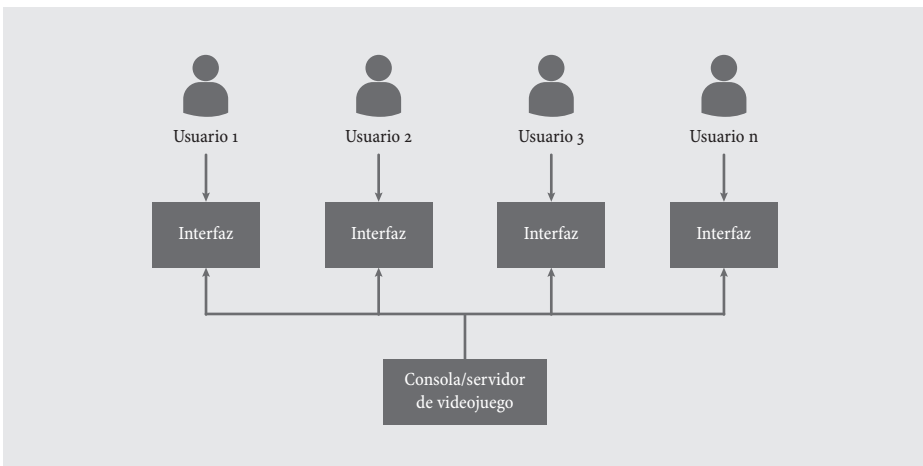


Figura 5.2. Videojuego en línea

Por su parte, dependiendo de la cantidad de usuarios y los requerimientos de recursos en el servidor para el videojuego, la estructura de la figura 5.2 puede repartirse para incrementar su capacidad, lo que deriva en una estructura cliente-servidor distribuida, como se observa en la figura 5.3

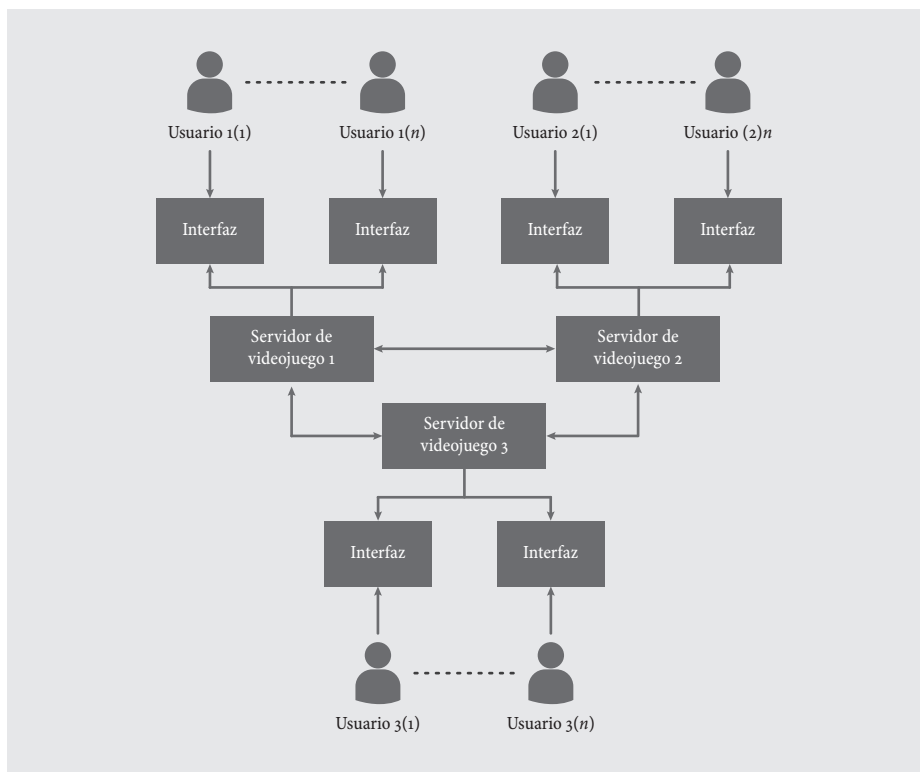


Figura 5.3. Videojuego en línea con una estructura distribuida

Una muestra de la estructura anterior se puede encontrar en *Second Life*, una aplicación donde los usuarios corren un programa-cliente que se conecta con un servidor central; este último utiliza tres clústers principales de computadoras. En primer lugar, el servidor de datos maneja una base de datos central, una de accesos, una de inventario y una de búsquedas. En segundo, hay una colección de servidores que simula el mundo virtual; estos están actualizándose todo el tiempo. Finalmente, hay un pequeño número de computadoras que realiza las funciones de autenticación de usuarios al entrar y envío de mensajes durante las sesiones (Kumar *et al.*, 2008). Cabe aclarar que si bien *Second Life* no es un videojuego en sí, en general tiene las características de los juegos, al contar con escenas llenas de objetos 3D y personajes con interacciones que cambian el estado de las mismas.

La distribución del sistema en diferentes servidores permite aprovechar los recursos que cada uno de ellos proporciona. Así pues, con base en estas ventajas se han considerado los sistemas *peer to peer* (P2P) como una opción interesante para el desarrollo de videojuegos, debido a la posibilidad de la descentralización completa tanto de la aplicación como de la información almacenada durante la ejecución del videojuego (Martínez y Larios, 2009; Ng *et al.*, 2003; Schiele *et al.*, 2007).

En este sentido, P2P es una arquitectura distribuida entre los clientes y no requiere de un servidor central para funcionar. El trabajo del servidor es repartido entre los clientes; cada computadora funciona como un servidor y cliente a la vez. Así, el ancho de banda utilizado se reduce dramáticamente debido a que los *peers* establecen comunicación directa entre ellos (Cabani, Pécuchet y Itmi, 2007; El Rhalibi y Merabti, 2005; Ng *et al.*, 2003). La figura 5.4 describe una estructura con estas características.

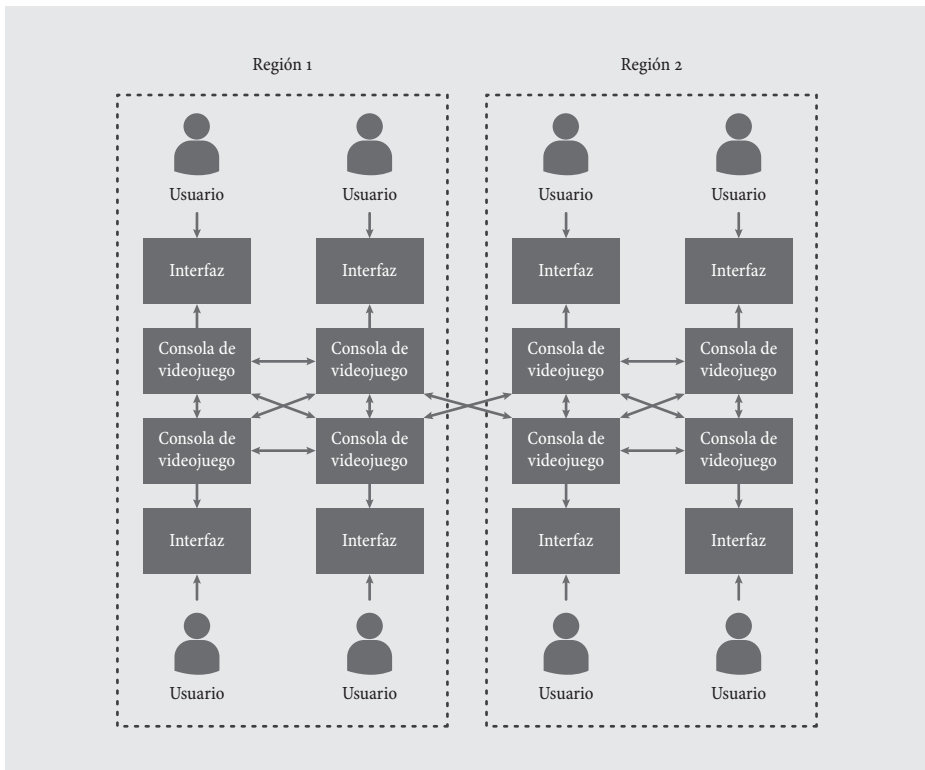


Figura 5.4. Videojuego en línea con una estructura P2P

Es por lo anterior que se vuelve importante conocer e identificar las diferentes arquitecturas de organización de estructuras de datos, así como la comunicación entre ellas requerida en los videojuegos. Con un poco de experiencia, esto debe permitir identificar las ventajas y desventajas de cada arquitectura, para elegir la opción más adecuada a los requerimientos y enfoque del proyecto de videojuego que se pretenda llevar a cabo.

Organización de la información de videojuegos en una base de datos

La información que se almacenará en la base de datos deberá incluir por lo menos los siguientes elementos:

- *Usuarios*. Se almacena la información de los usuarios registrados, de su perfil, de su contacto y de su actividad dentro del juego, además de los datos que el juego requiera para identificar y controlar su acceso.
- *Avatar*. La representación del jugador dentro del juego comúnmente es llamada avatar. En el sentido gráfico, un avatar puede ser una persona o un personaje que será el protagonista o héroe en el videojuego. La información que se requiere estará en relación con el objeto representado, su ubicación y propiedades.
- *Ítems*. Estos equivalen a los objetos estáticos y dinámicos, su ubicación, sus cambios en el estado y sus propiedades. Se relacionan con uno o varios usuarios, quienes serán los propietarios de los mismos.
- *Regiones*. Son los diferentes escenarios, que corresponden a las regiones en las que se divide la aplicación. Abarca toda la información relacionada con los elementos que la componen, la ubicación y las propiedades, tales como fronteras con otras regiones.

Los archivos que contienen la descripción gráfica, tanto de los avatares como de los objetos, escenarios y sus propiedades, pueden ser almacenados en repositorios de archivos, mientras que en la base de datos solamente se guardará la referencia de su ubicación, propiciando una disminución en el espacio requerido.

Problemas que hay que gestionar mediante la base de datos

Una vez identificadas las estructuras sobre las que un videojuego puede ser desarrollado, ahora se describen algunos de los problemas en los que, en el caso de los juegos persistentes, el manejador de bases de datos juega un papel importante para su solución:

- *Escalabilidad.* Se dice que un sistema es escalable si permanece eficaz cuando hay un incremento significativo en el número de recursos y/o de usuarios (Dollimore, Couloris y Kindberg, 2005). En una estructura cliente-servidor la escalabilidad estará limitada por los recursos de el(los) servidor(es). Adicionalmente, la capacidad del manejador de la base de datos para responder a las peticiones tendrá una mayor demanda conforme aumenten las consultas y actualizaciones. En una estructura distribuida, como es P2P, conforme crece el número de *peers* también aumenta la capacidad de los recursos, ya que cada nuevo dispositivo que se conecte a la red aportará los propios; sin embargo, el problema es la complejidad tanto en la administración de la aplicación como en el control de cambios en el estado y objetos, pues ambos deberán actualizarse en las diferentes bases de datos de cada cliente y ser propagados a los demás integrantes de la red.
- *Coherencia.* Un ambiente masivo multiusuario requiere manejar adecuadamente la interactividad para lograr una respuesta coherente, lo cual significa que los jugadores deberán percibir siempre el efecto de un evento de forma idéntica y al mismo tiempo, además de considerar las técnicas necesarias para identificar cuándo se requieren compensaciones para satisfacer las expectativas de los jugadores (Schiele *et al.*, 2007). Los cambios en el estado del ambiente y los objetos que el o los usuarios muevan o modifiquen, serán almacenados en la base de datos con el fin de asegurar que los demás usuarios, eventualmente, se enterarán de los cambios; el tiempo que transcurra desde que una actualización es llevada a cabo por un jugador y que sea reflejada por los diferentes ambientes del resto de los dispositivos, indicará la capacidad de la aplicación para manejar la coherencia del videojuego.
- *Persistencia.* Los perfiles, objetos y cambios en el mundo permanecen en el entorno virtual entre las activaciones de los procesos, es decir, entre la salida y entrada de un jugador (Dollimore, Couloris y Kindberg, 2005; Schiele, 2007). Esta información se almacena en la base de datos de tal forma que tenga un acceso rápido y efectivo, asegurando la actualización de dichos datos. Conservar la información una vez que es almacenada permitirá que los usuarios perciban el ambiente estable, aun cuando no se encuentren conectados. En este sentido, la persistencia es una característica inherente a las bases de datos, desde el formato tradicional en el que son conceptualizadas.
- *Concurrencia.* Equivale a la administración de cambios realizados sobre un mismo objeto al mismo tiempo, mediante técnicas de sincronización o serialización, con el fin de percibirlos de manera consistente (Dollimore, Couloris y Kindberg, 2005; Genovali y Ricci, 2008). Así, la administración de la concurrencia es otra característica inseparable de los sistemas tradicionales de bases de datos; no obstante, en el ámbito de los videojuegos la complejidad aumenta debido a la interacción que puede llegar a demandar una gran cantidad de accesos con requerimientos de tiempos de respuesta mínimos.

Con base en los problemas descritos previamente, es necesario considerar dos aspectos importantes; primero, que los juegos masivos multiusuarios (MMOGs) involucran desde mil hasta un millón de usuarios de forma simultánea, y segundo, que una interfaz gráfica requiere de 30 cuadros por segundo con el fin de que se perciba real para un humano (Burdea y Coiffet, 2003). De esto se deriva, en un cálculo conservador, que el manejador de la base de datos tendría que responder hasta 30 millones de accesos por segundo, ya sea para consulta o actualización de la información.

En suma, los problemas descritos son complejos, pero existen algunas propuestas de solución que permiten tomar ventajas con estrategias, mismas que se expresan enseguida.

Estrategias para soportar grandes cantidades de objetos en las escenas principales de los videojuegos

Desde el punto de vista de la escalabilidad, descrita anteriormente, conforme vaya aumentando el número de usuarios concurrentes es imprescindible aplicar estrategias que permitan mantener al sistema funcionando de manera estable y coherente. Algunas de estas estrategias son el manejo de regiones, el control de versiones de objetos de la escena del videojuego y su relación con las bases de datos, y las consideraciones de seguridad de la información en ellos.

Manejo de regiones

Se utiliza para la propagación de los cambios relevantes mediante la agrupación de los usuarios que comparten la misma área virtual. Consiste en la distribución de los datos almacenados y los archivos de los objetos 3D en los *peers*¹ localizados en la misma región y el replicado de dicha información para lograr la actualización constante de la interfaz gráfica (El Rhalibi y Merabti, 2005).

Además de esto, los usuarios tienen la posibilidad de cambiar de una región a otra y deben de percibir la actualización del ambiente de manera inmediata. Como referencia de lo anterior se puede hacer mención del funcionamiento de las antenas de telefonía celular, en las que conforme el usuario está en movimiento,

¹ Se denomina *peer* a cada uno de los usuarios participantes en un juego con una dirección IP en internet, desde la cual envía y recibe información en el sistema de comunicación establecido del videojuego.

la antena que le proporcionará el servicio será dependiendo del área de cobertura en la que se encuentre físicamente.

Control de versiones de objetos de la escena del videojuego y su relación con las bases de datos

Los objetos que integran el ambiente virtual están almacenados mediante archivos en repositorios; la referencia de su ubicación se almacena en la base de datos. En el momento en que los objetos son modificados, ya sea en alguna(s) de sus propiedades o su ubicación, se actualizará la base de datos local del usuario causante del cambio y se propagará a todos aquellos que comparten la misma región, de forma que la versión más actualizada estará almacenada y será informada a los jugadores más cercanos.

El resto de los jugadores tendrá acceso a la versión anterior guardada del objeto en cuestión, la cual será actualizada de forma gradual en el ambiente de cada uno de ellos, en caso de mantenerse dentro de su región; o de forma inmediata, si alguno de ellos se mueve hacia la región del usuario que originó el cambio.

En síntesis, el control de versiones permitirá identificar de forma segura la necesidad de actualizar la información relacionada con cada uno de los objetos, optimizando la propagación de mensajes con base en el manejo de regiones.

Consideraciones de seguridad de la información en los videojuegos

En una estructura cliente-servidor en la que la base de datos se ubica físicamente en uno o más servidores, el propio DBSM contiene las funciones necesarias para el control de acceso a la información, así como para el respaldo y recuperación de la misma en caso de algún incidente (Oppel, 2004). En una arquitectura P2P se requerirán estrategias de seguridad, probablemente basadas en el manejo de regiones.

Caso de estudio de un ambiente para videojuegos serios

Descripción del proyecto

El proyecto DVRmedia2 es un sistema masivo multiusuario de realidad virtual construido sobre una arquitectura P2P. Actualmente se encuentra en desarrollo un prototipo funcional para la simulación de situaciones de emergencia que requieran la participación de policías, bomberos y/o paramédicos, con fines de entrenamiento (VdeG, 2009).

La plataforma DVRmedia2 es un caso de estudio ubicado en el contexto de los videojuegos serios, los cuales se enfocan en el área de educación, entrenamiento y desarrollo de habilidades, entre otras.²

Capas conceptuales

Como se muestra en la figura 5.5, en la primera capa se ubica la interfaz gráfica, en la cual se involucran miles o quizá millones de usuarios concurrentes, que eventualmente requerirán acceso a la base de datos, tal como se dijo previamente. Por su parte, la distribución de los datos permitirá que la información relevante sea persistente durante el tiempo en que el juego se encuentre activo. Adicionalmente, los cambios deben ser actualizados tan rápido como sea posible mientras el sistema es lo suficientemente escalable para satisfacer las expectativas de los usuarios (Schiele *et al.*, 2007).

² Una de las páginas de internet que promociona proyectos de este tipo es Serious Games Initiative (Iniciativa de Juegos Serios). Para más información, véase <http://seriousgames.org/index.html>

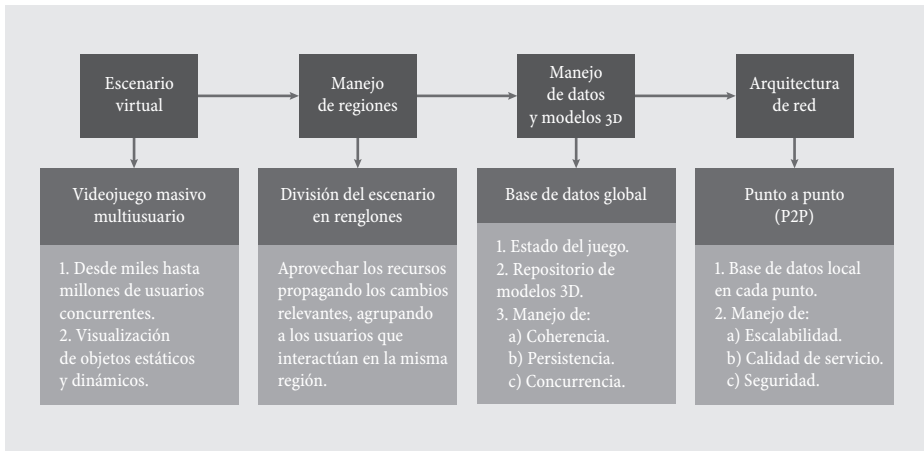


Figura 5.5. Capas conceptuales de un ambiente virtual P2P

Se pretende que el manejo de regiones, ubicado en la segunda capa, sea una estrategia lo bastante eficiente para aprovechar los recursos disponibles en cada uno de los dispositivos conectados, además de realizar de manera adecuada la propagación de los cambios relevantes. En relación con la base de datos, la implementación de esta estrategia requerirá de un adecuado control de versiones para asegurar las actualizaciones.

Enseguida, la tercera capa integra la información en una base de datos global, esto conceptualmente hablando, ya que físicamente los datos se distribuirán entre los equipos que conforman regiones, como lo indica la última capa, propiciando que los accesos sean, a su vez, descentralizados, por lo que el problema, entonces, se divide involucrando solamente a los usuarios internos de una región.

En otras palabras, la plataforma de red P2P sobre la que se desarrolla la aplicación integra todo el soporte de comunicación para el ambiente virtual; el manejo de regiones para reducir el tráfico en la red se lleva a cabo mediante los principios del diagrama de Voronoi, utilizando la librería de red VAST³ (Martínez y Larios, 2009).

Propuesta de base de datos P2P

Un sistema de base de datos P2P (PDBS) se concibe como una colección de repositorios locales autónomos que interactúan en un estilo P2P y que ofrecen acceso transparente a uno o más servicios de administración de datos (Boncz *et al.*, 2007; Bonifati *et al.*, 2008).

³ Para más información, véase <http://vast.sourceforge.net>

PDBS es una combinación de las tecnologías P2P y de las bases de datos tradicionales con ventajas sin paralelo en algunas ocasiones específicas en las que la tecnología de integración tradicional dificulta la implementación (Liu y Zhao, 2009).

Un ejemplo de lo anterior es Freenet, un sistema de almacenamiento distribuido que opera como una red P2P autoorganizada, que aprovecha espacio en disco no utilizado a través de cientos de miles de nodos potenciales para crear un sistema colaborativo virtual de archivos. Es importante decir que cada participante provee a la red de espacio para almacenamiento en disco. Para añadir nuevos archivos, un usuario envía a la red un mensaje de inserción que contiene el archivo y su identificador global único de asignación (GUID), el cual hace que el archivo sea almacenado en algún conjunto de nodos. Por su parte, durante el periodo de vida del archivo, este migrará o será replicado en otros nodos; para recuperarlo, el usuario envía un mensaje de petición que contiene el GUID. Cuando la petición es recibida por alguno de los nodos en los que el archivo está almacenado, este envía la información de regreso al que originó la petición (Clarke *et al.*, 2002).

Selección de estrategias

En el caso particular de la base de datos para DVRMedia2, hay que señalar que inició con una propuesta centralizada, y que se distribuirá gradualmente hasta llegar a un esquema P2P puro, es decir, sin la utilización de servidores ni de información centralizada. Cabe mencionar que las diferentes etapas serán evaluadas para determinar el nivel de distribución de la base de datos que permite el mejor desempeño de la aplicación.

Los archivos que contengan información requerida por la interfaz gráfica, como las representaciones humanas —comúnmente llamadas avatares—, los objetos, escenarios y sus propiedades, se almacenarán externamente a la base de datos, donde se registrará su ubicación para su localización cuando sean requeridos. Partiendo de que cada uno de los *peers* es a la vez cliente y servidor, el envío de los archivos del repositorio al cliente se llevará a cabo mediante funciones de *streaming* con el lenguaje de programación Java. El control de versiones soportado por la base de datos facilitará que las actualizaciones y cambios, tanto en los objetos como en los estados de los mismos, se mantengan adecuadamente.

Adicionalmente, el manejo de regiones se aplicará con un enfoque en la base de datos, lo que propiciará la propagación de los cambios más relevantes para el usuario de acuerdo con su ubicación. Esto significa que los cambios que el jugador pueda percibir dentro de la región en la que se encuentra se realizarán primero, permitiendo que los cambios en otras regiones se actualicen con algún retraso.

Opciones prácticas para el desarrollo de videojuegos

Algunas opciones para el desarrollo de videojuegos son las siguientes:

- *OpenSimulator*.⁴ Puede ser utilizado para crear un ambiente virtual. Al ser código abierto proporciona todas las herramientas necesarias para un desarrollo completo.
- *Proyecto Darkstar*.⁵ Esta opción, de SUN Microsystems, ofrece una plataforma de *software* para realizar juegos en línea, mundos virtuales y redes sociales.
- *Gamex*.⁶ Es de Cornell University. Proporciona código abierto y APIs para el desarrollo de videojuegos con lenguaje C++; contiene documentación, tutoriales y demos.

Conclusiones

Desde un punto de vista académico, el desarrollo de videojuegos se enfoca en mayor medida en la interfaz gráfica; sin embargo, es necesario conocer y comprender la arquitectura de un videojuego, con el fin de proponer el soporte que una base de datos puede proporcionar al desarrollo y desempeño del mismo (White *et al.*, 2007). El reto será encontrar oportunidades en las que esta base de datos permita mejorar y soportar las aplicaciones que presentan un alto grado de interacción, como lo son los juegos masivos multiusuario.

Para el caso de estudio *DVRmedia2*, el desafío será la integración de la base de datos con la aplicación desarrollada actualmente, contando con la definición del esquema adecuado. En el sitio *OpenSimulator*⁷ se puede encontrar el código abierto de un mundo virtual 3D similar al de *Second Life*;⁸ el esquema de base de datos utilizado para esta aplicación se tomó como referencia para proponer el que se muestra en la figura 5.6.

⁴ <http://opensimulator.org/>

⁵ <http://www.projectdarkstar.com/>

⁶ <http://gdiac.cis.cornell.edu/GameX/index.htm>

⁷ Para más información, véase <http://opensimulator.org/wiki/Database:Documentation>

⁸ Para más información, véase <http://secondlife.com/>

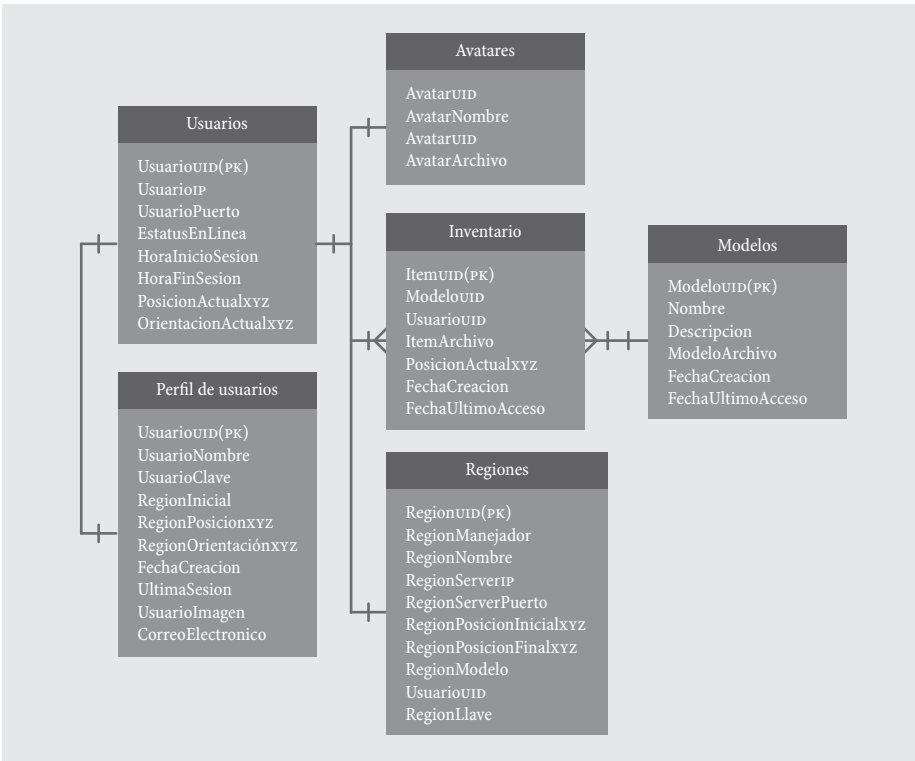


Figura 5.6. Esquema de la base de datos centralizada para DVRMedia2

Dicha integración deberá propiciar la solución de los problemas mencionados anteriormente (escalabilidad, coherencia, persistencia y concurrencia), todo ello sin afectar el tiempo de respuesta y basándose en la tecnología aplicada tanto en la comunicación como en la interfaz gráfica. La seguridad de la información se considerará desde las etapas iniciales de la implementación de la base de datos.

Finalmente, es necesario lograr que el sistema tome control del ambiente y de los oponentes con los que el jugador puede interactuar. Esto se logra a través de un motor de inteligencia artificial que toma las decisiones y acciones de algunos personajes independientes del control de los usuarios; el soporte que la base de datos pueda proporcionar será fundamental para el desempeño de la aplicación (véase siguiente capítulo).

CAPÍTULO 6

Estrategias de comunicación en red para videojuegos

MARTHA PATRICIA MARTÍNEZ VARGAS

VÍCTOR MANUEL LARIOS ROSILLO

Introducción

Si bien se considera que un primer nivel de interactividad en videojuegos equivale a que un usuario establezca una relación directamente con un videojuego, un segundo nivel corresponde al hecho de poder conectar a varios jugadores mediante la red en un mismo escenario para que colaboren o compitan entre ellos.

En el presente capítulo se aborda este segundo nivel a través de tres temas generales: los fundamentos de las aplicaciones masivas, la importancia de la conectividad en red y las tecnologías y tendencias actuales del videojuego en red. Cabe mencionar que todo lo anterior plantea una serie de problemas que se deben gestionar para que estos videojuegos puedan funcionar, además de que es necesario considerar los diferentes tipos de arquitecturas y los protocolos de comunicación a emplear.

A manera de ilustrar el estado del arte en comunicaciones y filtrado para videojuegos, se presenta la librería VAST como facilitadora en la conexión de los jugadores en red, la cual proporciona los beneficios de las arquitecturas punto a punto o *peer to peer* (P2P). Y para finalizar, se trata un ejemplo práctico de la conexión de multiusuarios con arquitectura P2P.

Fundamentos

Importancia de la red en relación con los videojuegos

En los recientes años, las aplicaciones masivas de multiusuarios de ambientes virtuales (MMVES) han incrementado su popularidad (Huang, Hu y Jiang, 2008). Estos ambientes virtuales se pueden entender como escenarios de videojuegos en 3D, mismos que han brindado al usuario una gran interactividad y noción de competitividad. Diversas aplicaciones aprovechan esta conectividad en red para mejorar el entretenimiento o generar sistemas colaborativos mediante juegos serios (Torguet, 2009).

Asimismo, estos juegos, con participantes en línea, disponen de arquitecturas y sistemas que soportan simultáneamente miles de usuarios, tales como *Second Life* y *World of Warcraft*, los cuales se tratarán más adelante.

Las tecnologías y tendencias actuales en videojuegos en red

Entre los casos más relevantes de aplicaciones masivas de usuarios, se pueden citar *Second Life*, *Fantasy Westward Journey-East* y *Eve Online*. Este tipo de aplicaciones proporciona a los usuarios un ambiente de escenarios que permiten intercomunicación e interacción en tiempo real con otros usuarios, lo que las hace atractivas y altamente recreativas.

En 2013, por ejemplo, el sistema de red social de *Second Life* registró picos de 64 mil usuarios concurrentes, y el de *Eve Online* reportó más de 50 mil en el mismo año (MMOData.net, 2013a) (véase figura 6.1). Por su parte, el videojuego en línea *Fantasy Westward Journey-East* reportó, durante 2011, picos de usuarios concurrentes de 2.32 millones (cada servidor maneja solo 80 mil jugadores concurrentes en el mismo mundo virtual) (MMOData.net, 2013b). Es importante remarcar que para soportar grandes cantidades de usuarios simultáneos se requiere resolver los problemas que se presentan en cuanto a gestión, garantizar el buen funcionamiento del estado compartido que se genera, y operar los cambios en las escenas de todos los usuarios concurrentes.



Figura 6.1. Entorno de *Second Life*

Motor de videojuegos

Un motor de videojuegos es un *software* diseñado para soportar la ejecución de videojuegos o ambientes virtuales. Provee funciones para simplificar el desarrollo, entre las que se encuentran el *rendering* o renderizado para gráficos en 2D y 3D, la física, la detección de colisiones, la animación, el sonido, la inteligencia artificial, el editor de programación, el manejo de memoria, los hilos, las escenas gráficas, los tutoriales, la transmisión de datos y el soporte de red, entre otros (Ward, 2012). En el cuadro 6.1 se presentan varios motores de videojuegos, todos ellos con soporte masivo de red.

Cuadro 6.1. Motores relevantes de videojuegos con soporte masivo de usuarios

Motor de video	Lenguaje de programación	Aplicación gráfica	Última versión	Primera versión
Unity ¹ 3D	JavaScript, C#, Boo	dx9, OpenGL, HTML elementos canvas	3.5.6 — Septiembre, 2012	2005
Unreal ² Engine 4	UnrealScript (orientado a objetos)	dx9, dx11	4.0 — Julio, 2012	1998
Web Graphics Library (WebGL)	JavaScript (cálculo de luces y sombras), Java	OpenGL ES 2.0	1.0 — Febrero, 2011	2011
Ogre3D ³	C#, Boo, Visual Basic .NET, Delphi, ASP.NET	OpenGL, Direct3D	1.8.1— Septiembre, 2012	1999
Java 3D	Java	OpenGL, Direct3D, JOGL	1.5.2 — 2008	1998
Irrlicht ⁴	C#, VisualBasic, Delphi, Java	OpenGL 1.2, OpenGL 3.x, Direct3D 8.1, Direct3D 9.0	1.7.3 — Febrero, 2012	2011
OpenSceneGraph ⁵	Lua, Python	OpenGL	3.0.1 — Julio, 2011	1998
jMonkey Engine (jME) ⁶	Java	LWJGL, OpenGL 2, OpenGL 4	2.1 — Marzo, 2011	2003

Si se quiere más información sobre los motores de video mencionados, véanse, para Unity 3D, <http://www.esenthel.com/?id=compare>; para Unreal Engine 4, <http://www.unrealengine.com/en/features/>; para Ogre3D, <http://www.ogre3d.org/>; para Irrlicht, <http://irrlicht.sourceforge.net/backup/features.html#drivers>; para OpenSceneGraph, <http://devmaster.net/devdb/engines/openscenegraph>; y para jME, <http://jmonke>

Problemas a gestionar en sistemas de usuarios interconectados

El hecho de lograr soportar una gran cantidad de usuarios interconectados significa necesariamente la confrontación de diversos problemas, como lo son la escalabilidad del sistema y la seguridad en el ambiente virtual, que entre más grande sea la población de los usuarios que comparta la misma escena, implicará un mayor riesgo.

Escalabilidad

Este problema impacta en la capacidad del sistema para soportar el incremento de usuarios y no saturar los recursos que tiene disponibles. Es por esta razón que se buscan soluciones que permitan mantener el sistema estable ante el crecimiento

de la cantidad de usuarios interconectados (Frey *et al.*, 2008). Un ejemplo de un sistema escalable es aquel que soporta mayor número de usuarios con objetos dinámicos y continúa estando estable sin interrupción o reducción de la calidad de servicio de comunicación entre sus usuarios. Por calidad de servicio se entiende que las actualizaciones del estado de los usuarios con los que interactúa, así como las del mismo jugador, sean coherentes y permitan dar la impresión de que lo hace en tiempo real simultáneamente con todos. Una mala calidad de servicio se presenta cuando hay interrupción en la comunicación; el servidor principal de la escena da un tratamiento lento a las modificaciones y actualizaciones de estado de los objetos de la escena y de los cambios de los participantes, lo que conlleva a poca interactividad (Shun-Yun, 2006).

Para lograr que un sistema sea escalable, se debe tomar como regla que a mayor cantidad de usuarios y objetos virtuales, mayores deberán ser los recursos que los soporten. Los sistemas masivos de usuarios actuales tienen una arquitectura cliente-servidor o centralizada, donde a mayor número de usuarios, mayor número de servidores que soportan a los usuarios y sus recursos. No obstante, esta arquitectura tiene ventajas y desventajas, mismas que serán tratadas más adelante. De momento se puede decir que una desventaja importante es el alto costo de implementación de esta arquitectura, razón por la cual se propone la arquitectura P2P o distribuida.

Seguridad en la comunicación de videojuegos

La seguridad en los ambientes virtuales se puede clasificar en tres tipos: la seguridad de autenticación de los usuarios, la integridad de los datos y la integridad de los participantes. Para el primer caso, solo aquellos usuarios registrados previamente deberán acceder al sistema (Torguet, 2009) mediante una base de datos donde se valide al usuario que desee ingresar al ambiente. En cuanto a la integridad de los datos, estos se deben encriptar antes de ser enviados por la red y descryptados en su recepción, es decir, se debe prevenir la información de los usuarios y la confiabilidad del estado del juego (Jiang, Chiou y Hu, 2007). Por último y no menos importante, está la integridad de los participantes. Esta se logra mediante la definición de reglas entre los usuarios, pudiendo reportar a aquel que tenga un inadecuado comportamiento y afecte la interacción entre los miembros del ambiente virtual.

Una vez solucionados estos problemas satisfactoriamente, se puede decir que se llega a lograr un sistema estable y confiable tanto para los usuarios como para el sistema.

Tipos de arquitecturas de comunicación de juegos en red

Existen tres tipos de arquitecturas de comunicación de juegos en red, que son centralizada, descentralizada y distribuida (Chen y Muntz, 2006).

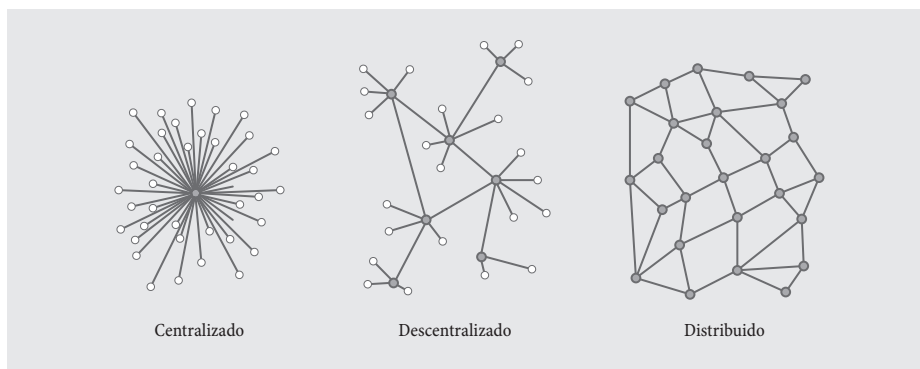


Figura 6.2. Tipos de arquitecturas de comunicación en red

En la arquitectura de red centralizada —conocida también como cliente-servidor—, la comunicación entre los clientes, así como el envío y recepción de los paquetes, se lleva a cabo por medio del servidor. Las principales desventajas de esta arquitectura son los cuellos de botella que pueden producir la dependencia de un punto central, o que el servidor quede fuera de servicio y el sistema deje de funcionar. Es por todo lo anterior que se concluye que mantener un sistema escalable resulta costoso en un esquema centralizado cliente-servidor (Singhal y Zyda, 1999).

Ahora bien, la arquitectura descentralizada —también denominada arquitectura multiservidor— tiene el mismo esquema de comunicación que una arquitectura cliente-servidor. Es posible tener varios usuarios compartiendo la misma área de interés, pero residiendo en diferentes servidores. En esta situación la comunicación es directa entre servidores, y posteriormente se realiza el envío a sus respectivos clientes. La ventaja de este esquema, comparado con la arquitectura cliente-servidor, es el mayor número de usuarios soportados y el incremento en la escalabilidad de la aplicación (Frey *et al.*, 2008).

Finalmente, una arquitectura distribuida o P2P es aquella en la que todos los participantes de la red realizan las mismas actividades o tareas; interactúan cooperativamente realizando al mismo tiempo los procesos de un cliente o un servidor. La gran ventaja de la arquitectura P2P es que explota el uso de los recursos de *hardware* y datos entre grandes cantidades de usuarios. Cada participante proporciona a la red sus recursos, generando un sistema robusto y con mayor grado de escalabilidad (Dollimore, Coulouris y Kindberg, 2005).

No obstante, a pesar de saber que la arquitectura distribuida es la más conveniente, la centralizada y descentralizada son las más empleadas en aplicaciones MMVES, lo que significa un alto costo en la manutención y soporte de un sistema escalable, y es en este sentido que surge la necesidad de trabajar con una arquitectura distribuida o P2P (Torguet, 2009). En el siguiente cuadro se resumen las ventajas y desventajas de emplear cada arquitectura de red.

Cuadro 6.2. Ventajas y desventajas de las arquitecturas de comunicación					
Arquitectura	Escalabilidad	Baja latencia	Simplicidad en la administración	Seguridad y recuperación de datos	Bajo costo en la implementación
Punto a punto	x	x			x
Cliente-servidor			x	x	
Multiservidor	x		x	x	

Protocolos de comunicación

Un protocolo es un conjunto de reglas bien definidas que permite la comunicación entre computadoras mediante la red, de tal manera que los equipos de cómputo de diversos fabricantes tienen un idioma en común para establecer dicha comunicación.

Al respecto, existen varios protocolos y clasificaciones, de los cuales el más estudiado es el modelo OSI,¹ que divide los datos en siete capas o niveles y los clasifica en dos categorías: transporte de datos y aplicación. La categoría de transporte de datos comprende los tres primeros niveles: físico, de enlace de datos y de red. Por su parte, la categoría de aplicaciones abarca los niveles de transporte, sesión, presentación y aplicación (Comer y Stevens, 1995). Vale la pena señalar que este tipo de aplicaciones se centran en el nivel cuatro, de transporte, que incluye los protocolos TCP y UDP, entre otros.

- *TCP (Protocolo de Control de Transmisión o Transmission Control Protocol)*. Es uno de los protocolos fundamentales de internet. Sus principales características son que asegura la recepción de los paquetes de datos, sin errores y en el mismo orden en que fueron enviados; proporciona un mecanismo para identificar los datos de diversas aplicaciones mediante el concepto de puertos; cuenta con un servicio de recuperación de paquetes (reenvía aquellos que no fueron recibidos). Para las aplicaciones de realidad virtual lo importante es obtener el paquete siguiente; todos aquellos que no fueron recibidos con anterioridad pierden la importancia hacerlo posteriormente. Entre sus desven-

¹ Sistemas de interconexión abiertos (*Open System Interconnection*).

Técnicas de transmisión de datos

Las técnicas de transmisión de datos son tres (Graves, 2012; Smed, Kaukoranta y Hakonen, 2002):

- *Unidifusión*. Es una comunicación directa entre el jugador que envía y el jugador que recibe; el control de los datos y el tráfico es P2P.
- *Multidifusión*. La comunicación es entre el jugador que envía y los múltiples jugadores receptores. La multidifusión provee una eficiente forma de transmitir la información para una gran cantidad de jugadores.
- *Radiodifusión*. La comunicación es entre el jugador que envía y todos los jugadores miembros de la red, lo que significa que cada jugador tiene que recibir y procesar los mensajes.

Estrategias para conectar en red a diversos usuarios de videojuegos

Los actuales videojuegos masivos en red fueron desarrollados con arquitecturas de red cliente-servidor o con un conjunto de servidores que forman un clúster. Este tipo de arquitecturas son costosas por el hecho de tener que soportar mayor número de usuarios, y su escalabilidad está limitada. Es por ello que surge la necesidad de desarrollar VAST, una librería de libre distribución, que permite organizar la comunicación para que cuente con un soporte P2P por medio de auras (Backhaus y Krause, 2007).

Auras

El concepto de aura o área de interés se representa con un círculo. En el círculo se integra el jugador y sus jugadores vecinos más próximos (Jiang, Huang y Hu, 2008). El envío de actualizaciones se realiza solo a los jugadores miembros del aura, reduciendo así el consumo de ancho de banda. Cabe mencionar que dicho consumo aumenta en aquellos casos que el aura se conforma por grandes cantidades de jugadores vecinos. En la figura 6.4 se muestra un aura integrada por el jugador y sus vecinos más próximos, los que recibirán las actualizaciones. Así, cada jugador tiene su propia aura, la cual es dinámica y se va creando o actualizando con la interacción de sus vecinos, y por lo tanto, al ser dinámica, los jugadores vecinos de mis vecinos son mis futuros vecinos a los que les actualizaré.



Figura 6.4. Aura o área de interés

Estrategias con VAST

Tal como se dijo previamente, VAST es una librería con arquitectura P2P que permite integrar a los jugadores al mismo ambiente. Al contar con la característica de áreas de interés, tiene una reducción de consumo de ancho de banda, ya que evita enviar las actualizaciones a todos los jugadores, identificando solamente a aquellos más cercanos para mandar el mensaje de actualización, el cual contiene coordenadas (x, y, z) y un identificador del usuario. En la identificación de los jugadores más próximos, VAST implementa el algoritmo de Voronoi (Aurenhammer, 1991). Dicho algoritmo consiste en dividir un plano de 2D en celdas. Cada celda representa a un jugador, integrando así el concepto de aura o área de interés (Backhaus y Krause, 2007).

Reducción de envío de paquetes de red

La reducción de envío de paquetes de red se logra con una estrategia llamada *dead reckoning* (DR). Esta estrategia fue desarrollada originalmente para la navegación, específicamente para estimar la posición actual de los barcos, dependiendo de una posición conocida inicial, así como la velocidad y el tiempo recorrido. Actualmente DR tiene varias aplicaciones, entre ellas están la determinación de la posición en que amanece y atardece; la predicción del lugar donde puede tocar tierra un huracán o del cuerpo celeste que podrá ser observado; y la evaluación de la eficacia de los dispositivos electrónicos de posicionamiento.

En la ciencia computacional, particularmente en aplicaciones de redes, DR es una herramienta para estimar no solo la posición presente, sino que también funciona para predecir la posición futura de objetos partiendo de un objeto real inicial (Smed y Hakonen, 2006; Aronson, 1997). Su reducción en el consumo de ancho de banda, como ya se señaló, es debido a que envía menos paquetes de

datos. Solo se envía un paquete cuando el objeto tiene un cambio en su trayectoria; la trayectoria se predice en los nodos remotos. No obstante, DR únicamente es una aproximación de la futura posición del objeto, pues existen varios factores que afectan la precisión, tales como los márgenes de maniobra, la posición actual o errores de aceleración (Bowditch, 1995).

En la figura 6.5 se ejemplifica la estrategia *dead reckoning*, donde se observa, del lado izquierdo, el usuario real que interactúa en el ambiente, y del lado derecho, la extrapolación del modelo real determinada por la trayectoria, siendo la actualización del objeto en un usuario remoto conectados en red (Singhal y Zyda, 1999).

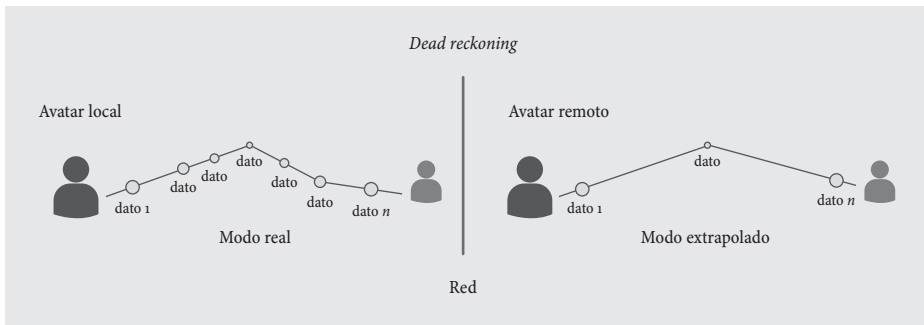


Figura 6.5. Estrategia *dead reckoning*

Implementación de un videojuego en red

Para integrar la red con el videojuego, el primer paso a realizar es descargar la librería VAST del sitio <http://vast.sourceforge.net/>. Es importante aclarar que en él existen dos distribuciones, una para C++ y otra para Java.²

Instrucciones de VAST:

1. Instalar Java Standard Edition, JDK (Java Development Kit)
2. Ejecutar el archivo “compile.bat” que se encuentra en la carpeta *test*. Esto generará el archivo “vast.jar” los archivos fuentes de vast, *class*”.
3. Integrar los archivos de código del videojuego con la librería VAST:
 - Para ello se requiere, primero, copiar los archivos del videojuego a la carpeta “src” de la carpeta de VAST.

² Este desarrollo se realizó con la distribución para Java y con el motor de gráficos JME (Java Monkey Engine).

- Enseguida se necesita un archivo para ejecutar el videojuego en red y solicitar a los jugadores el argumento de inicio, que es la dirección IP.³ Se tendrá que asignar a un jugador que realice la actividad de puerta de entrada o registro de los jugadores en red. Dicho jugador deberá ser el primero en ejecutar la aplicación; como dirección IP proporcionará -g. En la figura 6.6 se indica el ejemplo del código y se especifican las instrucciones más relevantes para la comunicación de ambientes virtuales.
- El siguiente paso se ejemplifica en la figura 6.7, que equivale a la creación de un mundo virtual con la inicialización de la comunicación y la recepción de las actualizaciones de los objetos virtuales.
- Finalmente, el resultado es el objeto virtual. En esta sección del código se debe hacer el envío de las actualizaciones. La información a actualizar son las posiciones x y z , la velocidad y el ángulo (véase figura 6.8).

³ Protocolo de internet.

Starter.java

```
1 package starterEngine;
2
3 import graphicEngine.World3D;
4 import com.jme.app.AbstractGame.ConfigShowMode;
5 //Networking
6 import java.net.InetSocketAddress;
7 import java.util.logging.Level;
8 import java.util.logging.Logger;
9
10 import networkEngine.vast.VASTNetworkUDP;
11
12 /**
13  * @author mzavala
14  *
15  */
16 public class Starter {
17
18     /**
19      * This class allows to run the virtual world,
20      * in addition to fetching all instantiated classes
21      */
22     public static void main(String[] args) {
23         Logger.getLogger("").setLevel(Level.WARNING);
24
25         InetSocketAddress isa = null;
26         boolean gateway = false;
27
28         if(args.length < 1){
29             System.err.println("Usage for gateway: java "+
30 Starter.class.getName()+" -g");
31             System.err.println("Usage for normal peers: java "+
32 Starter.class.getName()+" gateway_hostname");
33             System.exit(1);
34         }
35         if (args[0].equals("-g")){
36             isa = new InetSocketAddress("localhost",
37 VASTNetworkUDP.GATEWAY_PORT);
38             gateway = true;
39         }else{
40             isa = new InetSocketAddress(args[0],
41 VASTNetworkUDP.GATEWAY_PORT);
42         }
43     }
44 }
```

Figura 6.6. Archivo de ejecución de videojuegos en red, página 1

Starter.java

```
38
39         if(isa.isUnresolved()){
40             System.err.println("The gateway address " + args[0] +
"seems incorrect");
41             System.exit(1);
42         }
43     }
44
45     World3D world3D = new World3D(gateway, isa);
46     world3D.setConfigShowMode
(ConfigShowMode.AlwaysShow,World3D.class
47         .getClassLoader().getResource("images/
mmveTesisPortada2.jpg"));
48     world3D.start();
49 }
50 }
51
```

Page 2

Figura 6.6. Archivo de ejecución de videojuegos en red, página 2

World3D.java

```
1 package graphicEngine;
2
3 import java.util.HashMap;
4 import javax.swing.ImageIcon;
5
6 import com.jme.app.SimpleGame;
7 import com.jme.bounding.BoundingBox;
8 import com.jme.bounding.BoundingSphere;
9 import com.jme.image.Texture;
10 import com.jme.input.ChaseCamera;
11 import com.jme.input.KeyBindingManager;
12 import com.jme.input.KeyInput;
13 import com.jme.input.MouseInput;
14 import com.jme.input.ThirdPersonHandler;
15 import com.jme.light.DirectionalLight;
16 import com.jme.light.PointLight;
17 import com.jme.math.FastMath;
18 import com.jme.math.Quaternion;
19 import com.jme.math.Vector3f;
20 import com.jme.render器.ColorRGBA;
21 import com.jme.render器.Renderer;
22 import com.jme.render器.pass.RenderPass;
23 import com.jme.render器.pass.ShadowedRenderPass;
24 import com.jme.scene.Node;
25 import com.jme.scene.Skybox;
26 import com.jme.scene.shape.Box;
27 import com.jme.scene.state.CullState;
28 import com.jme.scene.state.TextureState;
29 import com.jme.scene.state.ZBufferState;
30 import com.jme.util.TextureManager;
31 import com.jme.util.resource.ResourceLocatorTool;//Used it to
    eliminated warnings
32 import com.jmex.terrain.TerrainPage;
33 import com.jmex.terrain.util.FaultFractalHeightMap;
34 import com.jmex.terrain.util.ProceduralTextureGenerator;
35
36
37 //Networking
38 import java.awt.Point;
39 import java.net.InetSocketAddress;
40 import java.net.SocketException;
```

Figura 6.7. Mundo 3D, página 1

World3D.java

```

41 import java.net.URL;
42 import java.util.Date;
43 import java.util.Enumeration;
44 import java.util.HashMap;
45 import java.util.Hashtable;
46 import java.util.logging.Level;
47 import java.util.logging.Logger;
48
49 //VAST
50 import networkEngine.vast.VASTCallback;
51 import networkEngine.vast.VASTID;
52 import networkEngine.vast.VASTNetworkUDP;
53 import networkEngine.vast.VAST_DCModel;
54
55 /**
56  * @author mzavala, mmartinez, ptorget, vlarios
57  *
58  */
59 @SuppressWarnings("unused")
60 public class World3D extends SimpleGame implements VASTCallback{
61
62     //private String id;
63     //private ChaseCamera chaser;
64     private ChaseCamera chaser;
65     // the terrain where the avatar will walk.
66     private TerrainPage page;
67     // the sky in the environment
68     private Skybox skybox;
69     // the buildings blocks
70     private Box building;
71     private Box building2;
72     //for the shadows
73     private Node occluders;
74     private Vector3f normal = new Vector3f();
75     private static ShadowedRenderPass sPass = new ShadowedRenderPass
76     ();
77     //Networking
78     private InetAddress ipGateway;
79     private boolean isGateway;
80

```

World3D.java

```

81  public static VASTNetworkUDP netlayer;
82  public static VAST_DCModel p2pmodel;
83  public static VASTID id;
84
85  //the local avatar
86  private Avatar myAvatar;
87
88  //other avatars
89  Hashtable<Integer, Avatar> avatar;
90
91  //used to compute the time spent in a draw loop
92  double lastTime = -1;
93
94  public World3D (boolean isGateway, InetAddress ipGateway){
95      this.isGateway = isGateway;
96      this.ipGateway = ipGateway;
97
98      //To create remote avatar
99      //a = new Avatar("Avatar " + id.getID());
100     avatar = new Hashtable<Integer, Avatar>();
101     initNetwork();
102
103     //shadow
104     stencilBits = 4;
105     // we need a minimum stencil buffer at least.
106 }
107
108 //init networking
109 public void initNetwork(){
110     try{
111         netlayer = new VASTNetworkUDP(isGateway, ipGateway);
112         //It is defined the maximum of allowed connections
113         p2pmodel = new VAST_DCModel(isGateway, 500, ipGateway);
114         p2pmodel.setNetwork(netlayer);
115         netlayer.setP2PModel(p2pmodel);
116         p2pmodel.setCallback(this);
117
118         //join the overlay at random position with an aoi of 50
119         float x = 0, y = 0;
120
121         Point pt = new Point();

```

World3D.java

```

122         pt.setLocation(x, y);
123         p2pmodel.join(500, pt, 0, 0);
124         //local_Avatar.translate(x, 0, y);
125     }catch(SocketException e){
126         e.printStackTrace();
127     }
128 }
129
130 /**
131  * builds the scene.
132  */
133 protected void simpleInitGame() {
134     display.setTitle("Network Virtual Environmet Prototype");
135     MouseInput.get().setCursorVisible(true);
136
137     buildCharacter(); //Building local_avatar
138     buildTerrain();
139     buildBuildings();
140     buildSky();
141     buildChaseCamera();
142     //buildInput();
143     setupOccluders();
144
145     rootNode.setRenderQueueMode(Renderer.QUEUE_OPAQUE);
146
147     //print and capture the object move
148     KeyBindingManager.getKeyBindingManager().set("Q",
149         KeyInput.KEY_Q);
150     KeyBindingManager.getKeyBindingManager().set("W",
151         KeyInput.KEY_W);
152     KeyBindingManager.getKeyBindingManager().set("X",
153         KeyInput.KEY_X);
154     KeyBindingManager.getKeyBindingManager().set("A",
155         KeyInput.KEY_A);
156     KeyBindingManager.getKeyBindingManager().set("S",
157         KeyInput.KEY_S);
158     KeyBindingManager.getKeyBindingManager().set("D",
159         KeyInput.KEY_D);
160
161
162 }

```

World3D.java

```

163
164     protected void simpleUpdate() {
165         //Compute the time spent between two calls
166         Date date = new Date();
167         double time = date.getTime() / 1000.0;
168         if(lastTime != -1 && myAvatar != null){
169             //System.out.println("NET UPDATE !");
170             //updates the local avatar
171             myAvatar.update(time - lastTime);
172             myAvatar.sendNetworkUpdate();
173             //process messages
174             p2pmodel.processMessage();
175
176             // update remote avatars
177             Enumeration<Avatar> enu = avatar.elements();
178             while(enu.hasMoreElements()){
179                 enu.nextElement().update(time - lastTime);
180             }
181             //keep the skybox around our eyes, so move it with the
camera
182             skybox.setLocalTranslation(cam.getLocation());
183
184             if (KeyBindingManager
185                 .getKeyBindingManager()
186                 .isValidCommand("Q", true)) {
187                 myAvatar.setVelocity(myAvatar.getVelocity()-0.5);
188             }
189             if (KeyBindingManager
190                 .getKeyBindingManager()
191                 .isValidCommand("W", true)) {
192                 myAvatar.setVelocity(myAvatar.getVelocity()+0.5);
193             }
194             if (KeyBindingManager
195                 .getKeyBindingManager()
196                 .isValidCommand("S", true)) {
197                 myAvatar.setVelocity(0);
198             }
199             if (KeyBindingManager
200                 .getKeyBindingManager()
201                 .isValidCommand("A", true)) {
202                 myAvatar.setOrientation(myAvatar.getOrientation()

```

World3D.java

```

+0.01f);
203         }
204         if (KeyBindingManager
205             .getKeyBindingManager()
206             .isValidCommand("X", true)) {
207             myAvatar.setVelocity(myAvatar.getVelocity()-0.5);
208         }
209         if (KeyBindingManager
210             .getKeyBindingManager()
211             .isValidCommand("D", true)) {
212             myAvatar.setOrientation(myAvatar.getOrientation
213         (-0.01f);
214         }
215     }
216
217     //input.update(tpf);
218     chaser.update(tpf);
219     float camMinHeight = page.getHeight(cam.getLocation()) + 2f;
220     if (!Float.isInfinite(camMinHeight) && !Float.isNaN
221         (camMinHeight)
222         && cam.getLocation().y <= camMinHeight) {
223         cam.getLocation().y = camMinHeight;
224         cam.update();
225     }
226
227     /**keep the avatar inside the camera's vision*/
228     /*
229     */
230     float characterMinHeight = page.getHeight(myAvatar
231         .getLocalTranslation().y+(BoundingBox)
232         myAvatar.getWorldBound().yExtent;
233     if (!Float.isInfinite(characterMinHeight) && !Float.isNaN
234         (characterMinHeight)) {
235         myAvatar.getLocalTranslation().y = characterMinHeight;
236     }
237     lastTime = time;
238 }

```

World3D.java

```

239 private void buildCharacter() {
240     /**
241      * Create my avatar instance to use the Avatar class.
242      */
243     //set the avatar attributes
244     myAvatar = new Avatar("Avatar Node");
245     rootNode.attachChild(myAvatar);
246     rootNode.updateGeometricState(0, true);
247     myAvatar.setRenderQueueMode(Renderer.QUEUE_OPAQUE);
248 }
249
250 /**
251  * TerrainPage is used to build a quad tree of terrain blocks
252  */
253 private void buildTerrain() {
254     /**
255      * Set up a basic, default light.
256      */
257     DirectionalLight dr = new DirectionalLight();
258     dr.setEnabled(true);
259     dr.setDiffuse(new ColorRGBA(1.0f, 1.0f, 1.0f, 1.0f));
260     dr.setAmbient(new ColorRGBA(.2f, .2f, .2f, .3f));
261     dr.setDirection(new Vector3f(0.5f, -0.4f, 0).normalizeLocal
262 ());
263     dr.setShadowCaster(true);
264
265     PointLight pl = new PointLight();
266     pl.setEnabled(true);
267     pl.setDiffuse(new ColorRGBA(.7f, .7f, .7f, 1.0f));
268     pl.setAmbient(new ColorRGBA(.25f, .25f, .25f, .25f));
269     pl.setLocation(new Vector3f(0,500,0));
270     pl.setShadowCaster(true);
271
272     DirectionalLight dr2 = new DirectionalLight();
273     dr2.setEnabled(true);
274     dr2.setDiffuse(new ColorRGBA(1.0f, 1.0f, 1.0f, 1.0f));
275     dr2.setAmbient(new ColorRGBA(.2f, .2f, .2f, .4f));
276     dr2.setDirection(new Vector3f(-0.2f, -0.3f, .
277 2f).normalizeLocal());
278     dr2.setShadowCaster(true);
279 }

```


World3D.java

```

278         CullState cs = display.getRenderer().createCullState();
279         cs.setCullFace(CullState.Face.Back);
280         cs.setEnabled(true);
281         rootNode.setRenderState(cs);
282
283         lightState.detachAll();
284         lightState.attach(dr);
285         lightState.attach(dr2);
286         lightState.attach(pl);
287         lightState.setGlobalAmbient(new ColorRGBA(0.6f, 0.6f, 0.6f,
1.0f));
288
289         FaultFractalHeightMap heightMap = new FaultFractalHeightMap
(257, 32, 0,
290             255, 0.55f);
291         Vector3f terrainScale = new Vector3f(10, 0, 10);
292         heightMap.setHeightScale(0.001f);
293         page = new TerrainPage("Terrain", 33, heightMap.getSize(),
294             terrainScale, heightMap.getHeightMap());
295
296         page.setDetailTexture(1, 16);
297         rootNode.attachChild(page);
298
299         ProceduralTextureGenerator pt = new
ProceduralTextureGenerator(
300             heightMap);
301         pt.addTexture(new ImageIcon(World3D.class.getClassLoader()
302             .getResource("images/grassb.png")), -128, 0, 128);
303         pt.addTexture(new ImageIcon(World3D.class.getClassLoader()
304             .getResource("images/dirt.jpg")), 0, 128, 255);
305         pt.addTexture(new ImageIcon(World3D.class.getClassLoader()
306             .getResource("images/highest.jpg")), 128, 255,
307             384);
308
309         pt.createTexture(512);
310
311         TextureState ts = display.getRenderer().createTextureState
();
312         ts.setEnabled(true);
313         Texture t1 = TextureManager.loadTexture(pt.getImageIcon
().getImage(),

```

World3D.java

```

314         Texture.MinificationFilter.Trilinear,
Texture.MagnificationFilter.Bilinear, true);
315         ts.setTexture(t1, 0);
316
317         Texture t2 = TextureManager.loadTexture(World3D.class
318             .getClassLoader()
319             .getResource("images/Detail.jpg"),
320             Texture.MinificationFilter.Trilinear,
Texture.MagnificationFilter.Bilinear);
321         ts.setTexture(t2, 1);
322         t2.setWrap(Texture.WrapMode.Repeat);
323
324         t1.setApply(Texture.ApplyMode.Combine);
325         t1.setCombineFuncRGB(Texture.CombinerFunctionRGB.Modulate);
326         t1.setCombineSrc0RGB(Texture.CombinerSource.CurrentTexture);
327         t1.setCombineOp0RGB(Texture.CombinerOperandRGB.SourceColor);
328         t1.setCombineSrc1RGB(Texture.CombinerSource.PrimaryColor);
329         t1.setCombineOp1RGB(Texture.CombinerOperandRGB.SourceColor);
330
331         t2.setApply(Texture.ApplyMode.Combine);
332         t2.setCombineFuncRGB(Texture.CombinerFunctionRGB.AddSigned);
333         t2.setCombineSrc0RGB(Texture.CombinerSource.CurrentTexture);
334         t2.setCombineOp0RGB(Texture.CombinerOperandRGB.SourceColor);
335         t2.setCombineSrc1RGB(Texture.CombinerSource.Previous);
336         t2.setCombineOp1RGB(Texture.CombinerOperandRGB.SourceColor);
337         rootNode.setRenderState(ts);
338     }
339
340     private void setupOccluders() {
341         TextureState ts = display.getRenderer().createTextureState
342         ();
343         ts.setEnabled(true);
344         /*ts.setTexture(
345             TextureManager.loadTexture(
346                 World3D.class.getClassLoader().getResource(
347                     "jmetest/data/texture/rust.jpg"),
348                 Texture.MinificationFilter.Trilinear,
349                 Texture.MagnificationFilter.Bilinear));
350         */
351         occluders = new Node("occs");
352         occluders.setRenderState(ts);

```

World3D.java

```

352     rootNode.attachChild(occluders);
353     for (int i = 0; i < 50; i++) {
354         Box b = new Box("box", new Vector3f(), 8, 50, 8);
355         b.setModelBound(new BoundingBox());
356         b.updateModelBound();
357         float x = (float) Math.random() * 2000 - 1000;
358         float z = (float) Math.random() * 2000 - 1000;
359         b.setLocalTranslation(new Vector3f(x, page.getHeight(x,
360             z)+50, z));
361         page.getSurfaceNormal(b.getLocalTranslation(), normal );
362         if (normal != null)
363             b.rotateUpTo(normal);
364         occluders.attachChild(b);
365     }
366     occluders.lock();
367 }
368 /**
369  * This part of the code it is only to see objects in the
370  * terrain
371  */
372 private void buildBuildings(){
373     //building1
374     building = new Box("building", new Vector3f(),5,8,5);
375     building.setLocalScale(5);
376     building.setLocalTranslation(0,20,20);
377     building.setModelBound(new BoundingBox());
378     building.updateModelBound();
379     TextureState ts = display.getRenderer().createTextureState
380 ();
381     ts.setEnabled(true);
382     ts.setTexture(TextureManager.loadTexture
383 (World3D.class.getClassLoader().getResource(
384     "images/Building1.jpg"),
385     Texture.MinificationFilter.BilinearNearestMipMap,
386     Texture.MagnificationFilter.Bilinear));
387     building.setRenderState(ts);
388     //rootNode.attachChild(building);
389 }

```

World3D.java

```
389
390
391 /**
392  * Skybox object with all the proper textures.
393  */
394 private void buildSky() {
395     skybox = new Skybox("skybox", 20, 20, 20);
396
397     //we want to keep the skybox around our eyes,
398     //so move it with the camera
399     skybox.setLocalTranslation(cam.getLocation());
400
401     Texture north = TextureManager.loadTexture(
402         World3D.class.getClassLoader().getResource(
403             "images/cloud.gif"),
404         Texture.MinificationFilter.BilinearNearestMipMap,
405         Texture.MagnificationFilter.Bilinear);
406     Texture south = TextureManager.loadTexture(
407         World3D.class.getClassLoader().getResource(
408             "images/cloud.gif"),
409         Texture.MinificationFilter.BilinearNearestMipMap,
410         Texture.MagnificationFilter.Bilinear);
411     Texture east = TextureManager.loadTexture(
412         World3D.class.getClassLoader().getResource(
413             "images/cloud.gif"),
414         Texture.MinificationFilter.BilinearNearestMipMap,
415         Texture.MagnificationFilter.Bilinear);
416     Texture west = TextureManager.loadTexture(
417         World3D.class.getClassLoader().getResource(
418             "images/cloud.gif"),
419         Texture.MinificationFilter.BilinearNearestMipMap,
420         Texture.MagnificationFilter.Bilinear);
421     Texture up = TextureManager.loadTexture(
422         World3D.class.getClassLoader().getResource(
423             "images/cloud.gif"),
424         Texture.MinificationFilter.BilinearNearestMipMap,
425         Texture.MagnificationFilter.Bilinear);
426     Texture down = TextureManager.loadTexture(
427         World3D.class.getClassLoader().getResource(
428             "images/cloud.gif"),
429         Texture.MinificationFilter.BilinearNearestMipMap,
```

World3D.java

```

430         Texture.MagnificationFilter.Bilinear);
431
432         skybox.setTexture(Skybox.Face.North, north);
433         skybox.setTexture(Skybox.Face.West, west);
434         skybox.setTexture(Skybox.Face.South, south);
435         skybox.setTexture(Skybox.Face.East, east);
436         skybox.setTexture(Skybox.Face.Up, up);
437         skybox.setTexture(Skybox.Face.Down, down);
438         skybox.preloadTextures();
439
440         CullState cullState = display.getRenderer().createCullState
441         ();
442         cullState.setCullFace(CullState.Face.None);
443         cullState.setEnabled(true);
444         skybox.setRenderState(cullState);
445
446         ZBufferState zState = display.getRenderer
447         ().createZBufferState();
448         zState.setEnabled(false);
449         skybox.setRenderState(zState);
450         rootNode.attachChild(skybox);
451     }
452
453     public void createPeer(VASTID id, Point pt, float orientation,
454     float z, double velocity){
455         System.err.println("Discovered Avatar " + id);
456         Avatar a = avatar.get(id.getID());
457
458         if(a == null){
459             a = new Avatar("Avatar " + id.getID());
460
461             rootNode.attachChild(a);
462             rootNode.updateGeometricState(0, true);
463             a.setRenderQueueMode(Renderer.QUEUE_OPAQUE);
464
465             a.setPosition(pt.x/1000, z, pt.y/1000);
466             a.setOrientation(orientation);
467             a.setVelocity(velocity);
468             avatar.put(id.getID(), a);
469         } else {
470             // only update its position

```

World3D.java

```

468         a.setPosition(pt.x/1000, z/1000, pt.y/1000);
469         a.setOrientation(orientation);
470         a.setVelocity(velocity);
471         System.err.println("Update for Avatar " + id.getID() + "
to (" + pt.x + "," + pt.y + ") vel "+velocity);
472     }
473
474     System.err.println("End - Discovered Avatar " + id);
475
476 }
477
478
479 public void removePeer(VASTID id){
480     Avatar a =avatar.get(id.getID());
481
482     rootNode.detachChild(a);
483
484     avatar.remove(id.getID());
485     System.err.println("Removed Avatar " + id);
486
487 }
488
489 public void updatePeer(VASTID id, Point pt, float orientation,
float z, double velocity){
490     Avatar a = avatar.get(id.getID());
491
492     if(a != null){
493         a.setPosition(pt.x/1000, z/1000, pt.y/1000);
494         a.setOrientation(orientation);
495         a.setVelocity(velocity);
496         System.err.println("Update from Avatar " + id.getID() +
" with coordinates (x:" + pt.x + ", y" + pt.y + ") velocity :
"+velocity);
497     }else{
498         System.err.println("Update from unknown Avatar " +
id.getID() + " with coordinates(x:" + pt.x + ", y" + pt.y + ")");
499     }
500
501 }
502
503 /**

```

World3D.java

```

504     * set the basic parameters of the chase camera.
505     * We want to be behind the avatar and a little above it.
506     */
507     private void buildChaseCamera() {
508         Vector3f targetOffset = new Vector3f();
509         targetOffset.y = ((BoundingBox) myAvatar.getWorldBound
510 ()).yExtent * 1.5f;
511         chaser = new ChaseCamera(cam, myAvatar);
512         chaser.setTargetOffset(targetOffset);
513         chaser.getMouseLook().setMinRollOut(150);
514         chaser.setMaxDistance(300);
515     }
516     /**
517     * create our custom input handler.
518     */
519     private void buildInput() {
520         /*
521         HashMap<String, Object> handlerProps = new HashMap<String,
522 Object>();
523         handlerProps.put(ThirdPersonHandler.PROP_DOGRAADUAL, "true");
524         handlerProps.put(ThirdPersonHandler.PROP_TURNSPEED, ""+(.5f
525 * FastMath.PI));
526         handlerProps.put(ThirdPersonHandler.PROP_LOCKBACKWARDS,
527 "true");
528         handlerProps.put(ThirdPersonHandler.PROP_CAMERAALIGNEDMOVE,
529 "true");
530         handlerProps.put(ThirdPersonHandler.PROP_ROTATEONLY,
531 "true");
532         input = new ThirdPersonHandler(myAvatar, cam, handlerProps);
533         input.setActionSpeed(100f);
534         */
535     }
536     /**
537     * close the window and also exit the program.
538     */
539     protected void quit() {
540         super.quit();
541         System.exit(0);
542     }
543 }

```

Avatar.java

```

1 /**
4 package graphicEngine;
5
6 import java.awt.Point;
23
24
25 /**
26  * @author mmartinez
27  *
28  */
29 public class Avatar extends Node {
30     private static final Logger logger = Logger
31         .getLogger(Avatar.class.getName());
32
33     private static final long serialVersionUID = 1L;
34     /**private Spatial model;*/
35
36     TerrainPage page;
37     Node model = null;
38     String modelResource;
39
40     float oldX =0;
41     float oldY =0;
42     float oldZ =0;
43     float oldAngle =0;
44     double oldVelocity=0;
45
46     //Values for position, orientation and velocity
47     double currentAngle = 0;
48     double velocity= 0;
49     int x = 0;
50     int y = 0;
51     double lastTime = -1;
52     public static final double MAXVEL = 100.f;
53
54     * Basic constructor takes the model that represents the
graphical
60     public Avatar(String modelResource) { //
96
97     /**
98     * retrieves the model Node of this avatar.

```


Avatar.java

```

99      * @return the model Node of this avatar.
100     */
101     public Node getModel() {
102         return model;
103         //return model2;
104     }
105
106     public void setPosition(float x, float y, float z) {
107         this.setLocalTranslation(new Vector3f(x,y,z));
108     }
109
110     public void setOrientation(float angle) {
111         this.localRotation.fromAngles(0, angle, 0);
112     }
113
114     public float getOrientation() {
115         float[] angles= new float[3];
116         this.localRotation.toAngles(angles);
117         return angles[1];
118     }
119
120     public double getVelocity(){
121         return velocity;
122     }
123
124     public void setVelocity(double velocity){
125         this.velocity = velocity;
126         if(this.velocity > MAXVEL)
127             this.velocity = MAXVEL;
128         if(this.velocity < - MAXVEL)
129             this.velocity = - MAXVEL;
130     }
131
132
133
134     //update the Avatar with the velocity and the time
135     public void update(double time){
136         if(velocity != 0) {
137             float[] angles= new float[3];
138             this.localRotation.toAngles(angles);
139             currentAngle = angles[1];

```

Avatar.java

```

140         float dx = (float) (velocity * time * Math.sin
    (currentAngle));
141         float dy = (float) (velocity * time * Math.cos
    (currentAngle));
142         //System.out.println("dx = "+dx+" dy = "+dy);
143         setPosition(this.localTranslation.x+dx,
    this.localTranslation.y, this.localTranslation.z+dy);
144     }
145 }
146
147 /**
148  * The method verifies if there is an update send the change in
    the network.
149  */
150 public void sendNetworkUpdate(){
151     //if the avatar didn't move we don't send anything.
152     float[] angles= new float[3];
153     this.localRotation.toAngles(angles);
154
155     Point pt = new Point();
156     pt.setLocation(this.localTranslation.x*1000,
    this.localTranslation.z*1000);
157     // send message
158     World3D.p2pmodel.setPosition(pt, angles
    [1],this.localTranslation.y*1000, velocity);
159 }
160 }
161

```

Conclusiones

Los videojuegos brindan un ambiente de diversión y esparcimiento a los jugadores; cuando se integra una conectividad en red se maximiza el alcance del videojuego al tener un entorno multiusuario.

Hoy en día se encuentran muchas aplicaciones con soporte masivo de usuario, como los casos de *Second Life*, *Fantasy Westward Journey-East* y *Eve Online*, que reportan miles de usuarios concurrentes y ganancias. Dichas aplicaciones enfrentan la gran creciente demanda de usuarios, y con ellos los diversos problemas que se presentan, como lo es la gestión de garantizar el buen funcionamiento del estado compartido y la actualización de la escenas de los jugadores concurrentes. Para ello, existen los diversos motores de videojuegos con soporte masivo de usuarios que facilitan y agilizan el desarrollo; además, cuentan con una comunidad que brinda soporte.

Al diseñar un videojuego en red se deben contemplar diversos problemas que se derivan del soportar la interacción de grandes poblaciones de usuarios simultáneos, entre ellos se encuentran la escalabilidad del juego; específicamente el soportar la adición de más jugadores con interacción concurrente, la seguridad en la comunicación de datos y jugadores, la arquitectura de red que brinda el soporte de interconexión entre los jugadores, los protocolos de comunicación encargados de transmitir las actualizaciones del ambiente virtual entre los jugadores, así como la selección de las técnicas de transmisión de datos.

En la actualidad, los videojuegos masivos en red han sido desarrollados con arquitecturas de red cliente-servidor o con un conjunto de servidores que integran un clúster. Cabe mencionar que estas arquitecturas de red son costosas, ya que soportan un gran número de usuarios y, en consecuencia, se genera un videojuego con escalabilidad limitada. Al respecto, se propone emplear la librería VAST, una librería de libre distribución con soporte punto a punto, donde cada jugador que se integre al videojuego lo haga con sus recursos de cómputo, soportando de esta manera la demanda y escalabilidad requerida por los jugadores (Shao-Chen, Yu-Li y Shun-Yun, 2012). Otra aportación que brinda la librería VAST propuesta es el concepto de aura o área de interés, la cual se representa con un círculo que se compone por el jugador y sus jugadores vecinos más próximos, actualizando solo a los jugadores miembros del aura, y logrando una reducción de los requerimientos de ancho de banda. Esta aura es dinámica y se actualiza con la interacción de nuevos vecinos (Shun-Yun, Shao-Chen y Jehn Ruey, 2008).

Por otro lado, en la búsqueda por reducir los requerimientos de ancho de banda para soportar mayor número usuarios, proponemos la estrategia *dead reckoning*, lo que hace que se reduzca el envío de paquetes de red. *Dead reckoning* hace una estimación de la posición presente y futura del objeto 3D enviando

solamente actualizaciones cuando exista un cambio en la trayectoria del objeto real (Aronson, 2012; Bowditch, 1995).

Así, después de mostrar los fundamentos de los videojuegos en red y proponer herramientas para facilitar la conexión de diversos jugadores o usuarios, se puede concluir este capítulo con la reflexión de que al integrar en red los videojuegos se obtendrá un ambiente más interactivo entre los jugadores, lo que dará como resultado mayor diversión y logrará metas más grandes con la colaboración de multiusuarios.

Conclusiones generales

A lo largo de toda la obra se mostraron los diferentes aspectos tanto técnicos como de frontera requeridos en el ámbito de desarrollo de videojuegos serios. En todos los casos, el propósito de este texto ha sido, más que mostrar una cadena de producción con paquetes de *software* que con el paso del tiempo pueden hacerse obsoletos rápidamente, revelar los procesos de forma genérica, así como los principios fundamentados en 40 años de experiencias recopiladas de equipos de desarrollo de videojuegos en general, pero con especial énfasis en aquellos de carácter serio.

Para efectos prácticos de estas conclusiones, la obra se dividirá en tres partes. La primera (equivalente al capítulo uno) abarca el concepto y producción de videojuegos serios como tal; la segunda (capítulo dos), los procesos relevantes de desarrollo; y la tercera, correspondiente a la sección técnica (capítulos tres a seis), es la de trabajos de frontera.

Así pues, en la primera parte se definió el concepto de videojuegos serios como un área profesional multidisciplinaria donde se vuelve indispensable el trabajo en equipo organizado. Dada la exigencia actual de los proyectos, es difícil que una sola persona pueda abarcar tantas habilidades y áreas de conocimiento para poder elaborar por sí mismo un videojuego serio. Por lo tanto, se llegó a la conclusión de que tres son las áreas primordiales para llevar a cabo este proceso: el diseño, el arte y la programación. En el diseño entra la fase de redacción, ideas y gestión necesaria para conjuntar talentos y generar la producción del videojuego. En lo que respecta al arte, se incluyen los trabajos que materializarán de manera estática las ideas propuestas en el diseño del guión e historia, buscando, según se establezca, recrear las escenas y ambientes lo más realistas y detallados posibles, de acuerdo con los recursos disponibles. Finalmente, la programación permitirá que los modelos creados por el equipo de arte cobren vida, basados en las reglas establecidas previamente por el equipo de diseño. En suma, esta parte lleva todo el desarrollo de la lógica que dictará la dinámica de interacción de los usuarios con el juego serio. Adicionalmente, se dio una perspectiva de los diferentes casos en que se han aplicado los videojuegos serios, donde el factor educativo es uno de los principales, pero no el único que motiva su desarrollo y proporciona una demanda de mercado. Hoy en día, con la gran diversidad de dispositivos móviles con que se cuenta para interactuar con los videojuegos, entre los que están cámaras,

giroscopios, sensores de georeferenciación, acelerómetros y detección de gestos, entre otros medios, se vuelve para el usuario toda una experiencia que permite explorar nuevas oportunidades de desarrollo de videojuegos serios. Además, es muy importante considerar durante este proceso la gestión de derechos de autor en todos los equipos de trabajo, así como contar con metodologías de desarrollo y gestión de proyectos eficientes y ágiles, debido a la complejidad y necesidad de interacción entre estos equipos.

Por otro lado, en la segunda parte de este libro se identificaron todos los procesos importantes de desarrollo de todos los equipos de trabajo en una secuencia ordenada e ilustrada con *software* de carácter libre. Con algunos ejemplos y técnicas empleadas más comunes, enriquecidos con la experiencia propia de proyectos reales, se hace una secuencia de procesos, vocabulario y reflexiones que buscan dar la perspectiva técnica y la visión necesaria para quien quiera incursionar y gestionar proyectos tangibles de videojuegos serios.

Finalmente, la sección técnica se hizo con el objetivo de seguir ampliando la visión del lector, para que pueda tener claro hasta dónde la tecnología permite el desarrollo del videojuego y en qué punto se vuelve un área de oportunidad donde no hay una sola solución. Atendiendo esto, se siguió entonces una secuencia de las diferentes áreas que están presentes con el inicio de la creación de multitudes de personajes realistas. Cabe resaltar que lo anterior requiere de competencias y dominios de los gráficos por computadora, así como de técnicas no convencionales de programación computacional, como son los algoritmos evolutivos. Luego de eso se utilizaron técnicas similares para diseñar la inteligencia artificial en el videojuego, es decir, para crear un comportamiento a todos los personajes, o en otras palabras, concebir un cerebro artificial y mostrar una forma particular de modelarlos y crearlos.

Una vez explicados los gráficos por computadora para la creación de los personajes y de la inteligencia artificial para darles un comportamiento autónomo, se presentaron las bases de datos como medio para gestionar toda la información que sucede en el ambiente del videojuego serio y, sobre todo, para dar una coherencia a las acciones de todos los personajes participantes.

Ahora bien, en razón de que los ambientes y escenas de un videojuego pueden contener una tal cantidad de objetos presentes, es que se vuelve necesario identificar técnicas de manejo y gestión de concurrencia a través de las bases de datos. Lo interesante de esta sección reside en seguir ampliando la visión y ver hasta dónde las bases de datos convencionales son empleadas en una nueva dimensión de interacción de los videojuegos serios, donde varios usuarios pueden compartir el mismo espacio, modificarlo y generar en este diferentes acciones y operaciones que lo modifican, pero siempre con un soporte de sistema eficiente.

El último capítulo de esta sección trató los conceptos de cómo transmitir datos en redes de computadoras, complementando la visión y el interés de generar

videojuegos serios con múltiples usuarios, compartiendo un mismo ambiente de manera remota mediante una red de datos y estrategias de comunicación eficiente. Debido a que las redes no tienen un canal de comunicación con capacidades ilimitadas, cuando se tiene un número importante de usuarios, que puede ir desde una decena hasta millares, es necesario tener estrategias y protocolos de comunicación eficientes. Estos protocolos, además de una serie de estrategias, que hoy están en proceso experimental y con resultados muy prometedores, complementan la obra y da una de varias áreas de extensión en donde se pueden ofrecer servicios e innovación a esta industria de los videojuegos serios.

En sí, este libro marca la pauta de una forma global, pero a su vez puntual, para aquellos que desean incurrir en el desarrollo profesional de los videojuegos serios y las consideraciones necesarias para poder integrar equipos de trabajo que puedan ser exitosos y reconocidos en esta industria.

Bibliografía

- ABELSON, H., LEDEEN, K. Y LEWIS, H. (2008). *Blown to bits: Your life, liberty, and happiness after the digital explosion*. Upper Saddle River, NJ: Addison-Wesley.
- AHEARN, L. (2001). *3D game art f/x design*. Scottsdale, AZ: Coriolis.
- ARONSON, J. (1997, 19 de septiembre). *Dead reckoning: Latency hiding for networked games*. Recuperado de http://www.gamasutra.com/view/feature/131638/dead_reckoning_latency_hiding_for_.php
- ARONSON, J. (2012). *Dead reckoning: Latency hiding for networked games*. Recuperado de http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php
- AURENHAMMER, F. (1991, septiembre). Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 345-405.
- BADLER, N., BINDIGANAVALA, R., BOURNE, J., PALMER, M., SHI, J. Y SCHULER, W. (2000). A parameterized action representation for virtual human agents. En Cassell, J., Sullivan, J., Prevost, S. y Churchill, E. (eds.), *Embodied conversational agents* (pp. 256-284). Cambridge: MIT Press.
- BACKHAUS, H. Y KRAUSE, S. (2007). Voronoi-based adaptive scalable. En *Netgames '07. Proceedings of 6th Annual Workshop on Network and Systems Support for Games* (pp. 49-54). Melbourne, Australia.
- BASTIONI, M., RE, S. Y MISRA, S. (2008). Ideas and methods for modeling 3D human figures: The principal algorithms used by makehuman and their implementation in a new approach to parametric modeling. En *COMPUTE '08. Proceedings of the 1st Bangalore Annual Compute Conference* (pp. 1-6). Bangalore, India: ACM.
- BEN AZOUZ, Z., RIOUX, M. Y LEPAGE, R. (2002, septiembre). 3D description of the human body shape using Karhunen-Loève expansion. *International Journal of Information Technology*, 8(2), 92-100.
- BONCZ, P., BONIFATI, A., ILLARRAMENDI, A., JANACIK, P., KÖNIG-RIES, B., LEHNER, W., ... TÜRKER, C. (2007). Working group report on managing and integrating data in p2p databases. En Böttcher, S., Gruenwald, L., Marrón, P. y Pitoura, E. (eds.), *Scalable data management in evolving networks*. Dagstuhl Seminar 06431. International Conference and Research Center (IBFI), Schloss Dagstuhl, Germany.
- BONIFATI, A., CHRYSANTHIS, P., OUKSEL, A. Y SATTler, K. (2008, marzo). Distributed databases and peer-to-peer databases: Past and present. *SIGMOD Record*, 37(1), 5-11.

- BOULIC, R., CAPIN, T., HUANG, Z., KALRA, P., LINTERMANN, B., MAGNENATTHALMANN, N., ... THALMANN, D. (1995, agosto). The HUMANOID environment for interactive animation of multiple deformable human characters. *Eurographics '95*, 14(3), 337-348.
- BOURKE, A. (2003). *Painless legacy FoxPro applications*. Whitefish Bay, WI: Hentzenwerke Publishing.
- BOWDITCH, N. (1995). Dead reckoning. Definition and purpose. En *The American Practical Navigator*. No. 9. (pp. 99-104). Washington, DC: National Imagery and Mapping Agency.
- BRILLIANT, K. (2003). *Building a digital human*. Graphics Series. Hingham, MA: Charles River Media.
- BURDEA, G. Y COIFFET, P. (2003). *Virtual reality technology* (2ª ed.). USA: John Wiley and Sons.
- BYRNE, E. (2004). *Game level design*. Game Development Series. Hingham, MA: Charles River Media.
- CABANI, A., PÉCUCHE, J. E ITMI, M. (2007). Distributed multiagent simulation on P2P architecture. En Roberts, D., Georgius, K. y Abdulmotaleb, E. (eds.), *11th IEEE International Symposium on Distributed Simulation and Real-Time Applications* (pp. 76-79). Chania: IEEE.
- CHAUDHURI, S., HORN, D., HANRAHAN, P. AND KOLTUN, V. (2008). CSTR 2008-02. Distributed rendering of virtual worlds. CS *Tech reports*. Stanford University.
- CHEANG WONG, J.C. (2005, 10 de julio). Ley de Moore, nanotecnología y nanociencias: síntesis y modificación de nanopartículas mediante la implantación de iones. *Revista Digital Universitaria*, 6(7). Recuperado de http://www.revista.unam.mx/vol.6/num7/art65/jul_art65.pdf
- CHEN, J. (2002). *Guide to graphics software tools*. Secaucus, NJ: Springer-Verlag New York Inc.
- CHEN, A. Y MUNTZ, R. (2006). Peer clustering: A hybrid approach to distributed virtual environments. En *Netgames '06. Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games* (pp. 30-31). New York, NY: ACM.
- CLARKE, I., MILLER, S., HONG, T., SANDBERG, O. Y WILEY, B. (2002). Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1), 40-49.
- CLAYTON, A.C. (2003). *Introduction to level design for PC games*. Hingham, MA: Charles River Media.
- CODEXEMPLA (2010). *Lenguajes de script y JavaScript*. Recuperado de http://codexemplar.org/curso/curso_4_1.php
- COMER, D. Y STEVENS, D. (1995). *Internetworking with TCP/IP: Design, implementation, and internals*. Vol. II (3ª ed). Englewood Cliffs, NJ: Prentice Hall.
- CYBERWARE (2010). *Whole Body Color 3D Scanner*. Recuperado de <http://cyberware.com/products/scanners/wbx.html>
- DOLLIMORE, J., COULOURIS, G. Y KINDBERG, T. (2005). *Distributed systems. Concepts and design* (4ª ed.). Harlow, England: Addison-Wesley.

- CUNNINGHAM, S. (2006). *Computer graphics: Programming in OpenGL for visual communication*. Upper Saddle River, NJ: Prentice Hall.
- DAVIS, L.D. Y MITCHELL, M. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.
- ECMA (2006). *C# language specification* (4a ed.). Geneva: ECMA International.
- EL RHALIBI A. Y MERABTI, M. (2005). Agents-based modeling for a peer-to-peer MMOG architecture. *ACM Computers in Entertainment*, 3(2), 3.
- EVRY, H. (2004). *Beginning game graphics*. Boston, MA: Course Technology.
- FALLER, A., SCHÜNKE, M. Y SCHÜNKE, G. (2004). *The human body: An introduction to structure and function*. Stuttgart: NY: Thieme.
- FERZZOLA, M. (2006, noviembre). Second Life. Un mundo paralelo lleno de oportunidades. *Neoteo virtual magazine*, 3(3), 26-29.
- FLACHER, F. Y SIGAUD, O. (2005). BASC, a bottom-up approach to automated design of spatial coordination. En AAMAS '05. *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 1109-1110). Utrecht, Netherlands: ACM.
- FOGEL, L., OWENS, A. Y WALSH, M. (1966). *Artificial intelligence through simulated evolution*. New York: John Wiley and Sons.
- FRANSON, D. (2002). *2D artwork and 3D modeling for game artists*. Boston, MA: Course Technology.
- (2004). *Dark side of game texturing*. Independence, Kentucky: Course Technology.
- FRASCA, G. (2003). Simulation versus narrative. Introduction to ludology. En Wolf, M. y Perron, B., *The video game theory reader* (pp. 221-235). London, NY: Routledge.
- FREY, D., ROYAN, J., PIEGAY, R., KERMARREC, A.M., ANCEAUME, E. Y LE FESSANT, F. (2008). Solipsis: A decentralized architecture for virtual environments. *MMVE 2008, The 1st International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality*. Reno, Nevada: MMVE.
- GARCÍA, S. (2012). *Halo 4 es el juego más caro de Microsoft*. Recuperado de <http://www.hobbyconsolas.com/noticias/halo-4-es-juego-mas-carro-microsoft-44699>
- GARCÍA, C., LARIOS, V. Y LUGA, H. (2010, marzo). Crowd behavior modeling using high performance computing. En *ISUM 2010 1st International Supercomputing Conference in Mexico*. Guadalajara, Jalisco.
- GENOVALI, L. Y RICCI, L. (2008). Voronoi models for distributed virtual environments. En CONEXT '08. *Proceedings of the 2008 ACM CONEXT Conference* (pp. 1-2). New York, NY: ACM.
- GRAVES, A. (2005). *Broadcast*. Recuperado de <http://whatis.techtarget.com/definition/broadcast>.
- GROFF, J., WEINBERG, P. Y OPPEL, A. (2009). *SQL: The complete reference* (3a ed.). New York, NY: McGraw-Hill/Osborne Media.

- GUTIÉRREZ, D., FRISCHER, B., CEREZO, E., GOMEZ, A. Y SERON, F. (2006, marzo). Ai and virtual crowds: Populating the colosseum. *Cultural Heritage*, 8(2), 176-185.
- HERAS, P. DE, SCHERTENLEIB, S., MAÏM, J. Y THALMANN, D. (2005). Reviving the roman odeon of Aphrodisias: Dynamic animation and variety control of crowds in virtual heritage. En *Proceedings of 11th International Conference on Virtual Systems and Multimedia (vsmm)* (pp. 601-610). Ghent, Belgium: Archaeoligia.
- HOLLAND, J. (1962). Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3), 297-314.
- HUANG, G.Y., HU, S.Y. Y JIANG, J.R. (2008). Scalable reputation management for P2P MMOGS. *Int. J. of Advanced Media and Communication*, 2(4), 380-401.
- KROEMER, K., SNOOK, S., MEADOWS, S.Y DEUTSCH, S. (EDS.) (1988). *Ergonomic models of anthropometry, human biomechanics and operator-equipment interfaces. Proceedings of a workshop*. Washington, DC: CHF.
- HOURVITZ, L. (2009). *Computer animation festival*. Recuperado de http://www.siggraph.org/asia2009/for_presenters/caf/
- INTERNATIONAL GAME DEVELOPERS ASSOCIATION (IGDA) (2010). *The IGDA community*. Recuperado de <http://www.igda.org/>
- JENKINS, H. (2004). Game design as narrative architecture. En Harrigan, P. y Wardrip-Fruin, N., *First person: New media as story, performance, and game* (pp. 118-130). Cambridge, MA: MIT Press.
- JIANG, J.R., CHIOU, J.S. Y HU, S.Y. (2007). Enhancing neighborhood consistency for peer-to-peer distributed virtual environments. En *Proceedings 27th International Conference on Distributed Computing Systems Workshops*. Totonto: IEEE.
- JIANG, J.R., HUANG, Y.L Y HU, S.Y. (2008). Scalable AOI-cast for peer-to-peer networked virtual environments. En *Proceedings 28th International Conference on Distributed Computing Systems Workshops Cooperative Distributed Systems* (pp. 447-452). IEEE.
- KEIBLER, C. (2007). *Avatar personas of Second Life*. Technical report. Kansas City: People Path.
- KUMAR, S., CHHUGANI, J., KIM, C., KIM, D., NGUYEN, A., DUBEY, P., BIENIA, C. Y KIM, Y. (2008). Second Life and the new generation of virtual worlds. *Computer*, 41(9), 46-53.
- LAMSTER, M. (2000). *Architecture and film*. New York, NY: Princetown Architectural Press.
- LAPLANTE, P. Y COSTELLO, T. (2005). *CIO wisdom II: More best practices*. Upper Saddle River, NJ: Prentice Hall.
- LEY FEDERAL DEL DERECHO DE AUTOR (LFDA) (2014). *Diario Oficial de la Federación* (publicada el 24 de diciembre de 1996).
- LIU, D. Y ZHAO, H.L. (2009). Distributed relational data sharing based on P2P. En *Proceedings 2009 International Conference on New Trends in Information and Service Science* (pp. 378-383). Beijing: IEEE.
- LUEBKE, D., REEDY, M., COHEN, J., VARSHNEY, A., WATSON, B. Y HUEBNER, R. (2003). *Level of detail for 3D graphics*. San Francisco, CA: Morgan Kaufmann Publishers.

- MAKEHUMAN (2009) *MakeHuman -1.0- Pre-alpha. User's guide*. Recuperado de http://makehuman.googlecode.com/svn/releases/makehuman_1_0_o_alpha3a/docs/MH_Users_Guide.pdf
- MARTÍNEZ VARGAS, M.P. Y LARIOS ROSILLO, V.M. (2009). *Dvrmedia2 P2P networking strategies to support massive online multiuser virtual environments*. Universidad de Guadalajara: Guadalajara.
- MCDONNELL, R., LARKIN, M., HERNÁNDEZ, B., RUDOMIN, I. Y O'SULLIVAN, C. (2009). Eye-catching crowds: saliency based selective variation. *ACM Transactions on Graphics*, 28(3), 1-10.
- MICHAEL, D.R Y CHEN, S. (2005). *Serious games: Games that educate, train, and inform*. Boston, MA: Thomson Course Technology.
- MMODATA.NET (2013a). *Peak concurrent users on the largest shards (single game worlds)*. Recuperado de <http://users.telenet.be/mmodata/Charts/PCushard.png>
- (2013b). *Subscriptions (ss) and active accounts (AA) with a peak above 1m*. Recuperado de <http://users.telenet.be/mmodata/Charts/Subs-1.png>
- MONTES, R. (2007). *Halo 3 genera 170 millones de dólares en su primer día*. Recuperado de <http://www.viciojuegos.com/noticia/Halo-3-genera-170-millones-de-dolares-en-su-primer-dia/16759>
- NATIONAL LIBRARY OF MEDICINE (2012). *Asian continental ancestry group*. Recuperado de http://www.nlm.nih.gov/cgi/mesh/2014/MB_cgi?mode=&index=22009&field=all&HM=&II=&PA=&form=&input=
- NG, W.S., OOI, B.C., TAN, K.L. Y ZHOU, A. (2003). PeerDB: A P2P-based system for distributed data sharing. En *Proceedings 19th International Conference on Data Engineering 2003* (pp. 633-644). IEEE.
- OPPEL, J. (2004). *Databases demystified*. McGraw-Hill Companies.
- ORKIN, J. (2003). *Constraining Autonomous Character Behavior With Human Concepts*. En Rabin, S., *AI game programming wisdom 2*. Rockland, MA: Charles River Media.
- (2004). Symbolic representation of game world state: Toward real-time planning in games. En *Procedings of the AAAI Workshop on Challenges in Game Artificial Intelligence* (pp. 26-30).
- (2006). Three states and a plan: The A.I. of F.E.A.R. *Game Developers Conference*.
- PARRADO, J.A. (1999, 28 de junio). La alta tecnología y el lujo se dan la mano en el Valle del Silicio. *Diario del Navegante*. Recuperado de <http://www.elmundo.es/navegante/99/junio/28/ricos.html>
- PARDEW, L. (2004). *Game art for teens*. Course Technology PTR.
- PENTON, R. (2002). *Data structures for game programmers*. Richmond, TX: Muska and Lipman/Premier-Trade.
- PIPHO, E. Y LAMOTHE, A. (2002). *Focus on 3D models*. Game Development Series. Independence, KY: The Premier Press.
- RECHENBERG, I. (1965). *Evolutionary strategies for technical innovation*. Technical report. Technischen Universität Berlin.

- RUBIO, J. (2012). *La Xbox360 edición Halo 4 nos muestra su lado más fotogénico*. Recuperado de <http://www.vandal.net/noticia/70912/la-xbox-360-edicion-halo-4-nos-muestra-su-lado-mas-fotogenico/>
- SÁNCHEZ, A., DÍAZ, R. Y BOCK, P. (2005). A CLS hierarchy for the classification of images. En Gelbukh, A., Albornoz, A. de y Terashima, H., *Lecture notes in computer science* (pp. 369-378). Berlín: Springer.
- SCHIELE, G., SUSELBECK, R., WACKER, A., HAHNER, J., BECKER, C. Y WEIS, T. (2007). Requirements of peer-to-peer-based massively multiplayer online gaming. En *Proceedings of the 7th International Workshop on Global and Peer-to-Peer Computing, organized at the IEEE/ACM International Symposium on Cluster Computing and the Grid 2007* (pp. 773-782). Río de Janeiro: ACM.
- SCHIESEL, S. (2009). *Visiting new Mombasa? It's one dangerous burg*. Recuperado de http://www.nytimes.com/2009/09/24/arts/television/24halo.html?_r=2&
- SHAO-CHEN, C., YU-LI, H. Y SHUN-YUN, H. (2012). *Voronoi-based adaptive scalable transfer*. Recupedado de <http://vast.sourceforge.net/>
- SHUN-YUN, H. (2006). A case for 3D streaming on peer-to-peer networks. En *Proceedings of the Eleventh International Conference on 3D Web Technology* (pp. 57-63). New York, NY: ACM.
- SHUN-YUN, H., SHAO-CHEN, C. Y JEHN RUEY, J. (2008). Voronoi state management for Peer-to-Peer massively multiplayer online games. En *Consumer Communications and Networking Conference. CCNC 2008. 5th IEEE* (pp. 1134-1138). Las Vegas: IEEE.
- SINCLAIR, B. (2008). *MS: 17.7 million 360s sold*. Recuperado de <http://www.gamespot.com/articles/ms-177-million-360s-sold/1100-6184291/>
- SINGHAL, S. Y ZYDA. M. (1999). *Networked virtual environments. Design and implementation*. New York, NY: ACM Press Books.
- SINHA, P., BALAS, B., OSTROVSKY, Y. Y RUSSELL, R. (2006). Face recognition by humans: Nineteen results all computer vision researchers should know about. *IEEE Proceedings*, 94(11), 1948-1962.
- SKETCHUP (2013). *¿Qué hace de SketchUp un acierto?* Recuperado de <http://www.sketchup.com/es/products/sketchup-pro/features/3D-modeling>
- SMED, J. Y HAKONEN, H. (2006). *Algorithms and networking for computer games*. England: John Wiley & Sons.
- SMED, J., KAUKORANTA, T. Y HAKONEN, H. (2002). Aspects of networking in multiplayer computer games review. *The Electronic Library*, 20(2), 87-97.
- STAFF SPECIALIZED SYSTEMS CONSULTANTS (SSC) (2002). *Linux in the workplace*. San Francisco: No Starch Press, Incorporated.
- TAKAHASHI, S., KHAN, I., OKUDA, M. Y IKEHARA, M. (2008, abril). Real time 3D avatar modeling and transmission using cylinder mapping. *Procedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2007* (pp. 473-476). Honolulu, Hawaii.

- TALTON, J., GIBSON, D., YANG, L., HANRAHAN, P. Y KOLTUN, V. (2009). Collaborative mapping of a parametric design space. En *Proceedings of the 2nd Annual ACM SIGGRAPH Conference and Exhibition in Asia*. New York, NY: ACM Press.
- THALMANN, D., CETRE, R., ULICNY, B., HERAS, P. DE Y CLAVIEN, M. (2004). Creating a virtual audience for the heritage of ancient theaters and odea. En *Proceedings 10th International Conference on Virtual Systems and Multimedia*. Ogaki City, Japan.
- TORGUET, P. (2009). *Networked virtual environments origins and architectures*. Toulouse, France: Paul Sabatier University/IRIT/VORTEX.
- TOSCA, S. (2003). The quest problem in computer games. En *Tecnologies for Interactive Digital Storytelling and Entertainment* (pp. 69-81). Darmstadt: IRB Verlag.
- TREUILLE, A., COOPER, S. Y POPOVIC, Z. (2006, julio). Continuum crowds. *ACM Transaction Graph*, 25(3), 1160-1168.
- ULICNY, B., HERAS, P. DE Y THALMANN, D. (2004). Crowdbush: Interactive authoring of real-time crowd scenes. En Boulic, R. y Pai, D.K. (eds.), *Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (pp. 243-252). Switzerland: Eurographics Association.
- UNIVERSIDAD DE GUADALAJARA (UDEG) (2009). *DVRmedia2 project*. Guadalajara. Recuperado de <http://cisgi.cucea.udg.mx/dvrmedia/>
- VOLZ, A., BLUM, R., HÄBERLING, S. Y KHAKZAR, K. (2007). Automatic, body measurements based generation of individual avatars using highly adjustable linear transformation. En *ICDHM '07. Proceedings of the 1st International Conference on Digital Human Modeling* (pp. 453-459). Berlin, Heidelberg: Springer-Verlag.
- WARD, J. (2008). *What is a game engine?* Recuperado de http://www.gamecareerguide.com/features/529/what_is_a_game_.php?page=1
- WHITE, W., KOCH, C., GUPTA, N., GEHRKE, J. Y DEMERS, A. (2007). Database research opportunities in computer games. *SIGMOD Rec.*, 36(3), 7-13.
- ZERBST, S. (2004). *3D game engine programming*. Boston: Course Technology Cirsp.

Autores

VÍCTOR MANUEL LARIOS ROSILLO

Es ingeniero en electrónica por el Instituto Tecnológico y de Estudios Superiores de Occidente, con maestría y doctorado en Ciencias Computacionales por la Universidad de Tecnología de Compiègne en Francia. Actualmente es profesor investigador titular de la Universidad de Guadalajara (udeg) y, desde abril de 2013, director del Centro de Innovación en Ciudades Inteligentes del Centro Universitario de Ciencias Económico Administrativas (CUCEA) de la udeg. Cuenta con una trayectoria de trece años en esta Casa de Estudio, en la que ha sido líder y colaborador de varios proyectos internacionales orientados a ambientes virtuales, simulación de multitudes, cómputo paralelo y juegos serios. Es el fundador del programa del Doctorado en Tecnologías de Información (DTI) de la udeg adscrito a la fecha al Programa Nacional de Posgrado de Calidad (PNPC) del Consejo Nacional de Ciencia y Tecnología (Conacyt). Entre sus colaboraciones, desde hace ya más de ocho años, ha estado involucrado en proyectos con la industria, donde particularmente con IBM, Intel y HP, trabaja desde tres años en el tema de Ciudades Inteligentes junto con un equipo de estudiantes del Doctorado en Tecnologías de la Información del CUCEA. En 2011 se convierte en propulsor del primer Centro de Exploración de Soluciones para Ciudades Inteligentes IBM-UDG, donde se siguen impulsando proyectos a la fecha, con el objetivo de crear un foco de innovación e investigación en esta temática. En 2013 fue consultor y director de ciencia y tecnología en el proyecto de Guadalajara Ciudad Creativa Digital, del cual estableció las bases de los proyectos catalizadores para la infraestructura tecnológica y el desarrollo de la plataforma de ciudad inteligente en Guadalajara y su zona metropolitana. Es miembro Senior del Instituto de Ingeniería Eléctrica y Electrónica (IEEE) con 23 años de pertenencia, y líder del Piloto Mundial de Guadalajara Smart City con IEEE desde julio de 2013 a la fecha, donde ha respaldado la creación de una red de Living Labs con el gobierno del estado, la creación y lanzamiento de los *Massive Online Open Courses (MOOCs)* en Ciudades Inteligentes del IEEE y una serie de *white papers* referentes en el tema. Sus áreas de interés están enfocadas al tema de simulación con sistemas multiagentes, videojuegos serios, ciudades inteligentes, analítica y visualización de información. En todos estos temas ha graduado a cinco estudiantes de doctorado, más de una docena de estudiantes de maestría y licenciatura, y participado en diversos jurados de tesis nacionales e internacionales, contribuyendo con un acervo importante de publicaciones y comunicaciones científicas.

ÓSCAR FERNANDO GONZÁLEZ DELGADILLO

Es licenciado en Negocios Internacionales por la Universidad de Guadalajara, con maestría en Negocios Internacionales y Estudios Económicos por la misma institución. Su tema central ha estado enfocado a la innovación y la comprensión de los factores de

éxito de emprendedores en los sectores de alta tecnología como es el de los videojuegos. Durante tres años, dirigido por el doctor Víctor M. Larios Rosillo, desarrolló competencias técnicas en todas las áreas de los procesos de producción de los videojuegos, que culminó con los modelos de procesos presentados en este libro, incluyendo incluso hasta algunos aspectos y nociones de programación. Es autor de varios videojuegos serios que se desarrollaron para la prueba de conceptos y selección de herramientas de código abierto o en un concepto de licencia libre o sin costo, siempre con la filosofía de que en una pequeña empresa se pudiera contar con posibilidades de desarrollo en este sector de alta creatividad digital. En su tesis de maestría reprodujo la experiencia de empresas de emprendimiento e innovación en el sector creativo digital. Entre la experiencia internacional más notable con relación a esta obra destaca el haber competido y participado en la impartición de un curso a profesores de Ciencias Computacionales en Estados Unidos dentro del evento de ACM SIGSIE en 2010 y el trabajo presentado fue el que dio vida a este libro. Asimismo, ha fungido como apoyo en seguimiento de proyectos en el tema de ciudades inteligentes con la empresa IBM y la Universidad de Guadalajara. Recientemente colaboró en las oficinas de ProMéxico en Mumbai, India, promoviendo nuevos negocios y mercados a nivel internacional.

MARTHA ELENA ZAVALA VILLA

Es maestra en Tecnologías de Información por la Universidad de Guadalajara, donde obtuvo el grado en 2006 con el tema de tesis *Optimización de un sistema de representación de objetos en Java 3D y OpenGL para visualización de grandes conjuntos de objetos mediante algoritmos mixtos BSP y LOD*. Cuenta con un doctorado en Tecnologías de Información también por la UdeG, en el cual, bajo la dirección del doctor Víctor M. Larios Rosillo, desarrolló la tesis *Heterogeneous humanoid editor to simulate virtual crowds in different environments using genetic algorithms and distance functions like crowd generators*, que fue codirigida por el doctor e investigador del equipo VORTEX del centro de investigaciones IRIT, Hervé Luga, de la Universidad de Toulouse 1 en Francia. El objetivo de este trabajo fue el de facilitar y automatizar la generación de personajes para multitudes en ambientes virtuales 3D. El proyecto atiende la necesidad en el área de videojuegos, dado para disminuir tiempos de desarrollo en la autoría y creación de multitudes de personajes para videojuegos o películas que requieren de animaciones digitales de multitudes. Su trabajo está respaldado por publicaciones en congresos científicos con arbitraje nacional e internacional de expertos en el tema. Actualmente se desenvuelve como consultora independiente y como profesora de asignatura en las licenciaturas de Tecnologías de Información y Ciencias Computacionales de la Universidad de Guadalajara.

CÉSAR ALEJANDRO GARCÍA GARCÍA

Ingeniero en Sistemas Computacionales por el Instituto Tecnológico y de Estudios Superiores de Monterrey, con maestría y doctorado en Tecnologías de Información

por la Universidad de Guadalajara; obtuvo un grado similar de doctorado en la Universidad de Toulouse en Francia por un proyecto especial de doble titulación. Su tema de investigación para la tesis doctoral está centrado en la inteligencia artificial y el modelado del comportamiento de personas en situaciones de emergencia para reproducir, mediante microsimulación, el fenómeno que se genera durante la evacuación de multitudes de personas en ambientes virtuales y videojuegos serios, la cual fue dirigida por doctor Víctor M. Larios Rosillo de la Universidad de Guadalajara, en codirección con el doctor Hervé Luga de la Universidad de Toulouse 1, miembro del equipo VORTEX del centro de investigaciones IRIT de Francia. Con previa experiencia profesional como desarrollador en la industria sobre aplicaciones de servicio en la *Web* y la nube, así como en ambientes para asistir a personas en edificios inteligentes, desarrolla su proyecto de tesis dentro del marco del programa binacional de intercambio PCP-Conacyt México-Francia, donde realiza estancias en el seno del centro de investigaciones IRIT y la Universidad de Toulouse 1, además de hacer pruebas y desarrollos de transferencia tecnológica de sus simulaciones con impacto en otras áreas de interés para empresas como Intel sobre simulaciones que requieren de cómputo paralelo. Durante su proyecto de tesis ha generado varias publicaciones con arbitraje internacional y nacional tanto en conferencias científicas como en revistas reconocidas en el área de videojuegos serios en relación con su trabajo de modelado con aportaciones prometedoras. También es profesor de tiempo completo en la Ingeniería en Ciencias Computacionales de la Universidad de Guadalajara.

LAURA TORRES LÓPEZ

Es ingeniera en Sistemas Computacionales por el Instituto Tecnológico y de Estudios Superiores de Occidente, con maestría en Comercio Electrónico por el Instituto Tecnológico y de Estudios Superiores de Occidente y doctorado en Tecnologías de Información por la Universidad de Guadalajara. Tiene experiencia profesional probada de más de diez años como desarrolladora y arquitecta de bases de datos, aplicaciones *web*, sistemas con programación orientada a objetos y aplicaciones cliente-servidor que ha desarrollado e implementado para las áreas administrativas de la Universidad de Guadalajara. En su propuesta de tesis doctoral, inició revisando cómo en los ambientes virtuales y videojuegos con una gran cantidad de usuarios interconectados mediante una red, las bases de datos podrían dar servicio eficiente para soportar la coherencia de las escenas de los ambientes virtuales frente a los cambios y modificaciones concurrentes de cada usuario. Sin embargo, siendo un problema interesante para los ambientes virtuales y videojuegos como punto de partida, en el ámbito de las bases de datos relacionales no representaba grandes retos de investigación. De ahí que con el fenómeno de las redes sociales emergentes con información no estructurada en bases de datos convencionales, se inició el estudio de otros tipos de bases de datos que procesaran grandes volúmenes de información y con estructuras diferentes de las relacionales. En ese sentido, el tema evolucionó de problemas centrados en mundos

virtuales (que bien pueden estar centrados en videojuegos serios) a los problemas y retos que presenta la recolección de grandes volúmenes de información de fuentes de datos en la convergencia tecnológica que hoy se aplica para hacer ciudades más inteligentes. En estos ambientes, la técnica de MapReduce es la que resultó interesante para procesamiento en paralelo de grandes volúmenes de información que fueran leídos en un formato estándar de la industria y en dicho procesamiento paralelo. Debido al tema de ciudades inteligentes, su tesis fue dirigida por el doctor Víctor M. Larios Rosillo, y codirigida por el doctor Ignacio Silva Lepe, quien es investigador de tiempo completo en la ciudad de Nueva York en el prestigiado centro de investigaciones IBM T. J. Watson. Adicionalmente, imparte cursos en las áreas de ciencias computacionales e ingeniería y se desarrolla como consultora independiente.

MARTHA PATRICIA MARTÍNEZ VARGAS

Es licenciada en Sistemas de Información, con maestría en Tecnologías de Información y doctorado en Tecnologías de información, todos por la Universidad de Guadalajara. Su tesis doctoral, con el tema *Estrategias de comunicación Punto a Punto para soportar multi-usuarios en línea en ambientes virtuales*, estuvo bajo la dirección del doctor Víctor M. Larios Rosillo de la Universidad de Guadalajara y del doctor Patrice Torguet de la Universidad Paul Sabatier en Toulouse, quien es investigador del equipo VORTEX del IRIT en Francia. Entre su experiencia previa al doctorado, ha desarrollado aplicaciones para servicios en la Web y se ha especializado en los protocolos de comunicación, particularmente para videojuegos, en donde se necesita interconectar grandes cantidades de usuarios que requieren compartir un mismo ambiente virtual. Junto con el equipo de investigación en Francia y otro grupo de desarrollo en China, en los últimos años ha participado en el desarrollo de sistemas de comunicación con protocolos VAST, que emplean como principio un esquema de optimización basado en identificación de vecinos de red por mapas de VORONOI. Su trabajo se respalda con publicaciones nacionales e internacionales en congresos científicos. Adicionalmente, es profesora de tiempo completo del Departamento de Sistemas de Información de la Universidad de Guadalajara, donde imparte clases en la Licenciatura en Sistemas de Información.

MARÍA DEL ROCÍO MACIEL ARELLANO

Es licenciada en Derecho por la Universidad de Guadalajara, y maestra y doctora en Ciencias de la Educación por el Instituto Mexicano de Estudios Pedagógicos. Es profesora investigadora del Departamento de Sistemas de Información del CUCEA en la Universidad de Guadalajara desde hace más de 18 años, y actualmente coordinadora del Doctorado en Tecnologías de Información de esta institución. Tiene más de 15 años de impartir cursos a nivel licenciatura en diseño gráfico, multimedia y gestión de servicios en tecnologías de información. También ha dirigido la creación de un Diplomado en Multimedia y el desarrollo del programa de una Especialidad en Animación y Videojuegos. Sus estudios en Derecho le han proporcionado una visión holística de las

tecnologías de información. Asimismo, por su posgrado, tiene amplia experiencia en el tema de educación virtual mediante plataformas en línea y ha dirigido varios trabajos de investigación y tesis de posgrado en este sentido, que son un respaldo importante para los videojuegos serios. Actualmente participa activamente en el grupo de trabajo en conjunto con la IEEE para el desarrollo de la educación en Ciudades Inteligentes, donde varios de los objetos de aprendizaje de cursos están centrados en los principios de los videojuegos serios.

Producción de videojuegos serios
se terminó de imprimir en mayo de 2015
en los talleres de Acento Editores
Reforma 842, Col. Centro
44100 Guadalajara, Jalisco

En la formación de este libro se utilizaron las familias tipográficas Minion Pro, diseñada por Robert Slimbach, y Ronnia, diseñada por Veronika Burian y José Scaglione.