Revista de I+D TECNOLÓGICO
Portal de revistas: http://revistas.utp.ac.pa

Arquitectura de *software* para el desarrollo de videojuegos sobre el motor de juego Unity 3D

Software architecture for the development of videogames on the game engine Unity 3D

Andy Hernández Paez [1*], Javier Alejandro Domínguez Falcón [2], Alejandro Andrés Pi Cruz[3]

[1,2,3] *Centro de Entornos Interactivos 3D, Vertex, Universidad de las Ciencias Informáticas. Carretera a San Antonio de los Baños, Km 2½, Torrens, La Lisa, La Habana, Cuba*

***Autor de correspondencia:** andyhp@uci.cu

RESUMEN– En los últimos años la arquitectura de *software* se ha consolidado como una disciplina que intenta contrarrestar los efectos negativos que pueden surgir durante el desarrollo de un producto informático, ocupando un rol significativo en la estrategia de negocio de una organización que basa sus operaciones en el *software*, volviéndose necesaria para todo tipo de desarrollo, incluyendo los videojuegos. La presente investigación tiene como objetivo desarrollar una arquitectura de *software* para videojuegos desarrollados sobre el motor de juego Unity 3D, que permita organizar y estructurar sus características funcionales básicas. A partir del estudio de arquitecturas usadas en videojuegos se agruparon las clases candidatas de la solución propuesta, identificándose los paquetes principales, dependencias entre ellos, patrones de diseño y buenas prácticas empleadas, concretando una arquitectura de *software* basada en la integración de los tipos de arquitectura: en capas y basada en componentes. Se desarrolla un prototipo funcional de un videojuego del género plataformas, empleando para describirlo elementos del diseño de videojuegos, especificación de mecanismos y las vistas propuestas por Robert Nord: conceptual, de módulos, de código y de implementación. La arquitectura propuesta fue validada a través de las técnicas de evaluación basadas en prototipos, en escenarios y en conjunto con la aplicación del método de Análisis de Acuerdos de Arquitectura de *Software*. Con la aplicación de esta técnica se identificaron los riesgos presentes en la arquitectura propuesta, teniendo en cuenta el comportamiento de atributos de calidad sobre la solución, según el modelo ISO/IEC 25010.

Palabras claves– *Arquitectura de software, ISO/IEC 25010, Unity 3D, videojuegos.*

ABSTRACT– In recent years, the software architecture has consolidated as a discipline that tries to counteract the negative effects that may arise during the development of a computer product, playing a significant role in the business strategy of an organization that bases its operations on software, becoming necessary for all kinds of development, including video games. The present research aims to develop a software architecture for video games developed on the game engine Unity 3D, which allows organizing and structuring its basic functional characteristics. Based on the study of architectures used in video games, the candidate classes of the proposed solution were grouped, identifying the main packages, dependencies among them, design patterns and good practices used, specifying a software architecture based on the integration of architecture types: layered and component based. It develops a functional prototype of a video game of the genre platforms, using to describe it elements of video game design, specification of mechanisms and the views proposed by Robert Nord: conceptual, modules, code and implementation. The proposed architecture was validated through evaluation techniques based on prototypes, in scenarios and in conjunction with the application of the Software Architecture Agreement Analysis method. The application of this technique identified the risks present in the proposed architecture, taking into account the behavior of quality attributes on the solution, according to the ISO / IEC 25010 model.

Keywords– *Software architecture, ISO / IEC 25010, Unity 3D, videogames.*

1. Introducción

El uso de las Tecnologías de Información y la Comunicación (TIC) en los últimos años ha permitido extender la informática a muchos sectores, ocupando un lugar creciente en la vida humana y en el sistema de la sociedad. Uno de los procesos que mayor impacto ha tenido en la sociedad es el de creación de videojuegos, que se vuelve más complejo con el tiempo producto del aumento de los requerimientos de los usuarios.

Los videojuegos son una vía de entretenimiento interactivo en el que uno o varios usuarios, mediante un dispositivo de entrada, se comunican con un sistema que posea imágenes de video. La plataforma en la que se desarrolle (o sistema) puede ser una computadora, consola, o incluso un celular. La interactividad usuario–sistema está dada por la capacidad del equipo de trabajo de planificar un sistema en el que el usuario se sienta cómodo y controle la situación [1].

Estos productos de *software* han evolucionado con increíble rapidez en los últimos años, convirtiéndose en una industria que cuenta con equipos de trabajo multidisciplinarios, al punto que para desarrollar un

Citación: A. Hernández, J. Domínguez y A. Cruz, "Arquitectura de software para el desarrollo de videojuegos sobre el motor de juego Unity 3D", *Revista de I+D Tecnológico*, vol. 14, n.° 1, pp. 54-64, Jun. 2018.
Tipo de artículo: Original. **Recibido:** 22 de febrero de 2018. **Recibido con correcciones:** 26 de febrero de 2018. **Aceptado:** 11 de abril de 2018.

videojuego es necesario, desde el concepto inicial hasta su versión final, el conocimiento de varias disciplinas y habilidades por parte del equipo de trabajo, tales como: programación, arquitectura de *software*, diseño y *marketing*. El desarrollo de productos de este tipo pasa por la creatividad y tiene en cuenta otros factores, como la estética o la temática en el instante de definir ciertos géneros [2].

Actualmente, se pueden encontrar en los mercados diferentes opciones de videojuegos, tales como: basados en licencias libres como comerciales que usan diferentes motores, los que se encargan de renderizado para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, *scripting*, animación, inteligencia artificial, redes, *streaming*, administración de memoria y un escenario gráfico [3].

Entre los motores de juego más utilizados se encuentra Unity 3D [4], que es empleado para realizar proyectos tanto 2D como 3D, dado que posee un entorno de desarrollo sencillo de manejar para los principiantes y suficientemente potente para los expertos, permitiendo crear fácilmente videojuegos y aplicaciones para diversas plataformas.

El diseño y desarrollo de un videojuego es demasiado complejo para que pueda ser abordado por completo sin utilizar una estructura lógica que describa los componentes que lo forman. Otro aspecto esencial para estructurar y desarrollar un videojuego con la calidad deseada es la Arquitectura de *software* (AS), la cual se encarga de definir de forma abstracta, los componentes de un sistema, sus interfaces y la comunicación entre ellos. Una AS, consiste en un conjunto de patrones y abstracciones coherentes que proporcionan soluciones a problemas de manera eficiente [5], así como se encarga de satisfacer determinados atributos de calidad, tales como: mantenibilidad, extensibilidad, flexibilidad e interacción con otros sistemas de información [6].

En la Universidad de Ciencias Informáticas (UCI) en colaboración con los Estudios de Animación del Instituto Cubano de Arte e Industria Cinematográfico (ICAIC) se realizan videojuegos de diferentes géneros. Para ello la universidad cuenta con el Centro de Entornos Interactivos 3D Vertex, el cual tiene como objetivo generar soluciones integrales, tecnologías, productos y servicios informáticos virtuales que cumplan con las necesidades y expectativas de los usuarios, con un alto valor agregado, resultado de un ciclo completo de I+D+I (Investigación-Desarrollo e Innovación) [7].

Este centro ha desarrollado varios productos utilizando el motor de juego Unity 3D, los cuales han sido liberados satisfactoriamente desde la Dirección de Calidad de la Universidad, algunos son: Súper Claria, Aventuras en la Manigua, Especies invasoras, La Neurona, Villa Tesoro y Caos Numérico. Después de la entrega de los productos antes mencionados se ha continuado con el desarrollo de nuevos videojuegos, pero se ha detectado, mediante consultas a expertos en el desarrollo, algunas limitaciones. La inexistencia de una estructura lógica basada en paquetes de la implementación dificulta el entendimiento de esta por parte del equipo de desarrollo y de los nuevos especialistas. Además, la poca representación y documentación que describe el funcionamiento de un videojuego que se desarrolle sobre el motor de juego Unity 3D, obstaculiza el proceso de su aprendizaje para los desarrolladores. Por otra parte, no se tiene referencia sobre el código fuente que se necesita para el desarrollo de nuevos videojuegos, lo que provoca desorganización en la implementación y diversidad en el modo que se implementa en los proyectos de este tipo. El análisis anterior conduce al siguiente problema de la investigación: ¿Cómo establecer una organización y estructuración funcional de las características básicas de un videojuego que se desarrolle sobre el motor de juego Unity 3D? El objetivo de esta investigación consiste en: Desarrollar una arquitectura de *software* para videojuegos implementados sobre el motor de juego Unity 3D, que permita organizar y estructurar sus características funcionales básicas.

2. Materiales y métodos

Como métodos científicos de investigación para la obtención de información sobre el proceso de desarrollo de videojuegos centrado en la arquitectura se emplearon, dentro de los teóricos: el histórico-lógico para el análisis y las tendencias actuales de las arquitecturas de *software*; el analítico-sintético para extraer y analizar la información sobre las principales arquitecturas de *software* usadas en el Centro de Entornos Interactivos 3D, Vertex. Dentro de los métodos empíricos se utilizaron: consultas de fuentes de información para el análisis de fuentes bibliográficas y

criterios de personal involucrado directamente con el proceso de desarrollo de videojuegos en el centro Vertex; la observación para observar los resultados obtenidos en la caracterización e identificación de los principales tipos de arquitecturas de *software* y proponer la solución

2.1 Arquitectura de *software*

Kruchten, en Rational Unified Process plantea: "La arquitectura de *software* representa la estructura o las estructuras del sistema, que consta de componentes de *software*, las propiedades visibles externamente y las relaciones entre ellas" [8].

Por otro lado, Robert Nord plantea [9]: "Arquitectura de *software* es sobre tomar decisiones estructurales fundamentales que son costosos de cambiar una vez implementado. Opciones de arquitectura de *software*, también llamados decisiones arquitectónicas, incluyen opciones estructurales específicas de posibilidades en el diseño de *software*. Por ejemplo, los sistemas que controlaban la lanzadera de espacio vehículo de lanzamiento tenía el requisito de ser muy rápido y muy confiable. Por lo tanto, una adecuada computación en tiempo real lengua tendría que ser elegido. Además, para satisfacer la necesidad de fiabilidad la elección podría realizarse para tener múltiples redundantes e independientemente producido copias del programa y ejecutar estas copias en independiente *hardware* y contrastar resultados".

2.2 Conceptualización de videojuegos

Un videojuego o juego de video es un *software* creado para el entretenimiento en general y basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta dicho videojuego; este dispositivo electrónico puede ser una computadora, una máquina arcade, una videoconsola, un dispositivo de mano (un teléfono móvil, por ejemplo) los cuales son conocidos como "plataformas" [1].

Aunque, usualmente el término "video" en la palabra "videojuego" se refiere en sí a un visualizador de gráficos, hoy en día se utiliza para hacer referencia a cualquier tipo de visualizador. Se entiende por videojuegos todo tipo de juego digital interactivo, con independencia de su soporte [2].

2.3 Arquitecturas de *software* utilizadas para el desarrollo de videojuegos

Un ejemplo que se podría citar es la Arquitectura para videojuegos serios con aspectos culturales, publicada por Ricardo Emmanuel Gutiérrez Hernández, Francisco Álvarez Rodríguez y Jaime Muñoz [10], de la Universidad Autónoma de Aguascalientes en México, que consiste en una arquitectura en seis Capas, las cuales son: interfaz de usuario, escenario, objetos juego, decoración, aplicación y contexto cultural.

Para la propuesta de solución no se utilizará como referencia esta arquitectura, debido a que los videojuegos desarrollados en el centro Vertex no se centran en contextos culturales, sin embargo, el estudio de la organización de sus componentes brinda una mayor visión de cómo estructurar un sistema en capas, definiendo qué almacena cada una de manera más organizada.

Otro ejemplo cercano al contexto que se analiza es la tesis que lleva por título: "Diseño y Desarrollo de un Prototipo Básico de un Videojuego de Plataformas en 2D" publicada por Carlota Esteban Cazalla [11]. Para esta tesis se utilizó como motor de juego Unity 3D y una arquitectura Modelo-Vista-Controlador (MVC).

El patrón MVC permite separar los datos de la lógica del videojuego. Sin embargo, desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas. Si el modelo experimenta cambios frecuentes, podrían desbordarse las vistas con una lluvia de requerimientos de actualización, lo que trae consigo que no cumpla con el atributo de calidad mantenibilidad en proyectos con cambios frecuentes en los requisitos como suelen ser los videojuegos. Teniendo en cuenta estas limitantes, dicha arquitectura es utilizada como referencia solamente para concebir la estructuración y organización de los componentes del sistema en capas.

2.3.1 Arquitecturas de *software* utilizadas para el desarrollo de videojuegos en el centro Vertex

En el centro Vertex se han desarrollado videojuegos de diferentes géneros, contando con equipos de diversos programadores y arquitectos de *software*, que en la mayoría de los casos no coinciden, por lo que sus criterios a la hora de definir una AS para el videojuego difieren. Esto, sumado a que en muchos casos no se contaba con un estudio del arte referente al tema, ha traído consigo desorganización y dio origen a la

necesidad de proponer una AS que satisfaga la situación problemática antes descrita [12].

2.3.2 AS del videojuego Especies invasoras

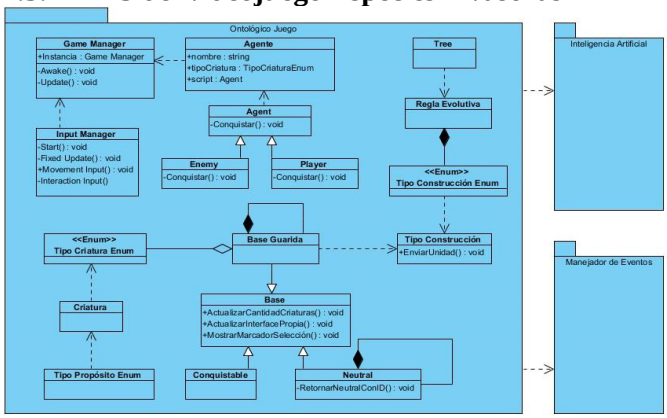

Figura 1. AS del videojuego Especies Invasoras.

En el videojuego Especies Invasoras se utilizó una arquitectura en tres Capas representada como se muestra en la figura 1 [12], funcionando de la siguiente manera:

Ontológico Juego: En esta capa se encuentra la lógica del videojuego, junto a las clases principales, cuya comunicación se evidencia de la manera planteada en la figura 1. En el Game Manager se implementó el patrón de diseño singleton, de manera que se tiene en el videojuego una única instancia de dicha clase para usarse cuando se requiera. Se puede apreciar también cómo heredan atributos y métodos las clases Conquistable, Neutral y Base Guardia de Base y Enemy y Player de Agent respectivamente.

Manejador de Eventos: En esta capa se implementó el patrón de diseño Event Handler, estando suscritas algunas de las clases del juego al manejador de eventos y controlando los mismos (el orden entre ellos) mediante una lista, debido a que el videojuego contaba con demasiados estados como para ser controlado con eficiencia mediante consultas simples.

Inteligencia Artificial: En esta capa se implementó la lógica de la inteligencia artificial y el comportamiento tanto de los personajes como los enemigos.

De la AS usada en el videojuego "Especies Invasoras", buenas prácticas observadas que se pueden incorporar a la propuesta de solución son:

- La herencia de algunas de sus clases, que garantiza reutilización y extensibilidad.
- El polimorfismo observado en métodos como Conquistar(), que permite separar los elementos que cambian de los que no lo hacen. El método mencionado se implementa de diversas formas, facilitando la ampliación, el mantenimiento y la reutilización del código.
- El uso del patrón de diseño singleton en el Game Manager, que garantiza que haya una sola instancia de dicha clase, reduciendo la carga en memoria de datos y optimizando el código.
- El uso del manejador de eventos (Event Handler), lo que posibilita optimizar el control de acciones en un juego que tiene demasiados estados.

2.3.3 AS del videojuego Caos Numérico

Como solución a la estructura del videojuego Caos Numérico se usó una arquitectura por Capas [12], definida de la forma que se muestra en la figura 2.

En la capa principal se encuentran los managers (Game Manager, Data Manager, Audio Manager, Level Manager, Load Manager), cada uno de ellos tiene un patrón Singleton (una única instancia) implementado para comunicarse entre ellos y con la capa inferior.

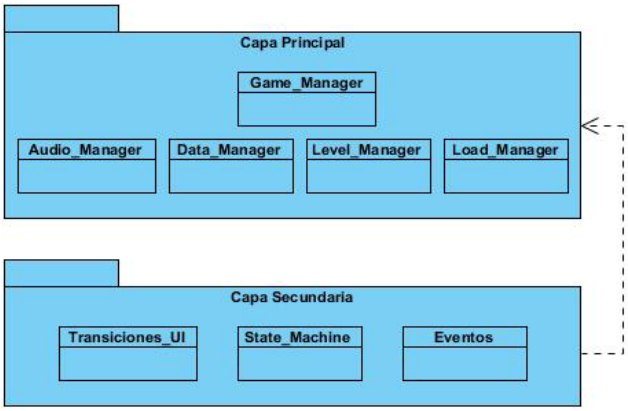

Figura 2. AS del videojuego Caos Numérico.

En la capa secundaria se encuentran las clases:

- **Transiciones UI:** Se encarga de las transiciones de la interfaz de usuario.
- **State Machine:** Se encarga de los diferentes estados que puede tener la escena principal, que es en la que se implementó la máquina de estados.
- **Eventos:** Usa tipo de datos delegate o delegado (propio de C#), representando métodos con una lista de parámetros determinada y un tipo de valor devuelto. Se usa para llamar a los métodos a través de la instancia del delegado.

De la AS usada en el videojuego "Caos Numérico", buenas prácticas observadas que se pueden incorporar a la propuesta de solución son:

- El uso del tipo de datos delegado para llamar a los métodos a través de instancias de ese tipo.
- El uso del patrón de diseño Singleton en cada uno de los *managers* (Game Manager, Audio Manager, Data Manager, Level Manager, Load Manager).
- La organización y estructuración del sistema en Capas.

A pesar de estas buenas prácticas cabe señalar que en la capa principal pudo implementarse el patrón de diseño fachada en el Game Manager. El objetivo de usarlo sería mantener una sola instancia de dicha clase y no de todas las que se accederá desde el Game Manager (Audio Manager, Data Manager, Level Manager, Load Manager).

Por otro lado, se toma como referencia para la propuesta de solución la posibilidad de estructurar los componentes de un producto base en diferentes capas según las arquitecturas estudiadas. De igual forma, la relación con las capas inmediatas que requieren y proporcionan determinados servicios para el control de los eventos contribuye a la extensibilidad de las soluciones que se desarrollen sobre el dominio de aplicación de videojuegos.

3. Resultados y discusión

A partir del estudio realizado se propone, una arquitectura que combina los patrones arquitectónicos: en Capas y basado en componentes, para estructurar los diferentes elementos necesarios en un videojuego. Los componentes de cada Capa se comunican con componentes de otras Capas a través de interfaces definidas o instancias de clases (en el caso de las clases se comunican con el Game Manager únicamente dado que es el que provee una fachada para las clases de la Capa Principal para interactuar con las demás). En la figura 3 se observa la distribución de las Capas presentes en la arquitectura propuesta estructurada en 3 Capas.

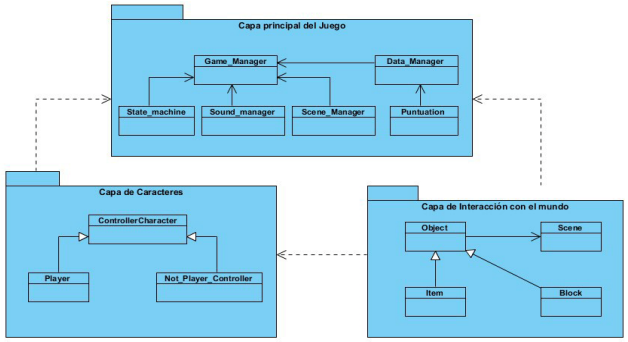

Figura 3. AS base para el desarrollo de videojuegos.

Según Pressman, desde un punto de vista orientado a objetos, un componente es un conjunto de clases que colaboran [5], por lo que en cada script de la AS propuesta se definen las clases requeridas para definirlo como un componente. Para cada uno de los componentes de la AS propuesta se definen las clases (con los atributos y operaciones apropiadas). Cada clase dentro de un componente se elabora por completo para que incluya todos los métodos y acciones relevantes para su implementación. Estos componentes fueron construidos teniendo en cuenta lo reutilizable, involucran interfaces, las funciones que realizan, así como la comunicación y colaboración que requieren. Como parte de la solución, también se definen todas las interfaces que permiten que las clases se comuniquen y colaboren con otras clases dependiendo de la Capa en la que se encuentren.

Las Capas están compuestas de la siguiente manera:

Capa Principal del Juego: En esta capa se encuentra el controlador principal del juego (Game Manager), la máquina de estados (State Machine), el sonido (Sound Manager), datos (Data Manager), escenas (Scene Manager System) y Puntuación.

- **Game Manager:** Es el controlador principal del sistema. Se implementa como un gameObject de Unity que se encuentra en todas las escenas del videojuego. Tiene asociados el resto de los managers, almacenando una instancia de cada uno de ellos. En caso de que se necesite usar puntuaciones para el videojuego, los datos se guardan en hashtables o tablas hash, que se encarga de asociar datos con valores y puede ser usada en la AS propuesta para almacenar grandes cantidades de información si es necesario. Si se desea acceder al

controlador principal del juego se usa un patrón de diseño singleton que se implementó en el mismo.

- **State Machine:** Se encarga de definir los estados del videojuego. Guarda los mismos en un enum que posee todos los diferentes estados que puede asumir el videojuego en un momento dado.
- **Sound Manager:** Controla el sonido del juego y particularmente de cada elemento o acción que active un sonido. Guarda los tipos de sonidos (dos tipos de sonido, o sea, dos listas: FX para Effects y BGM para Background) en listas usando estructuras (struct), formadas por tres elementos: un AudioClip, un AudioSource para poder reproducir el AudioClip y un enum que guarda los nombres de los sonidos para diferenciarlos a la hora de agregarlos al videojuego.
- **Data Manager:** Maneja los datos del juego. Se encarga de cargar, guardar el estado del videojuego o la puntuación.
- **Scene Manager:** Maneja las escenas del juego y los cambios entre ellas.
- **Puntuación:** Controla la puntuación del juego. En caso de no existir se puede eliminar esta clase de la arquitectura.

Capa de Caracteres: En esta capa se encuentran los caracteres (ControladorPersonaje), diferenciándose entre ellos por ser jugables (Player) o no jugables (NPC). Si se desea agregar otro tipo de personaje que intervenga en el videojuego se incorpora en esta capa.

Capa de Interacción con el mundo: En esta capa se encuentran el escenario y los objetos que pertenecen al mismo. Si existen otros tipos de objetos que modifiquen el videojuego se agregan a esta capa.

3.1 Restricciones arquitectónicas

- La arquitectura debe garantizar que los productos que se desarrollen sean multiplataforma (Android, Windows, Ubuntu).
- La arquitectura debe permitir la actualización, modificación o incorporación de compontes de forma natural.
- Las prestaciones de *hardware* dependerán de los requerimientos no funcionales de los productos que se desarrollen utilizando la arquitectura propuesta.
- Se debe usar como motor de videojuegos Unity 3D.

3.2 Vistas arquitectónicas

Para describir la arquitectura de *software* propuesta sobre el prototipo funcional desarrollado se utilizaron las vistas arquitectónicas que propone Robert Nord [9] y las buenas prácticas de ingeniería de *software* propuestas por el marco de trabajo ingenieril para el proceso de desarrollo de videojuegos utilizado en el centro Vertex [13]:

- Vista conceptual
- Vista de módulos
- Vista de código
- Vista de ejecución.

3.2.1 Vista conceptual

En esta vista se describe el sistema en términos de sus elementos principales de diseño y las relaciones entre estos según el dominio como se visualiza en la figura 4. Esta vista es independiente de las decisiones de implementación [14].

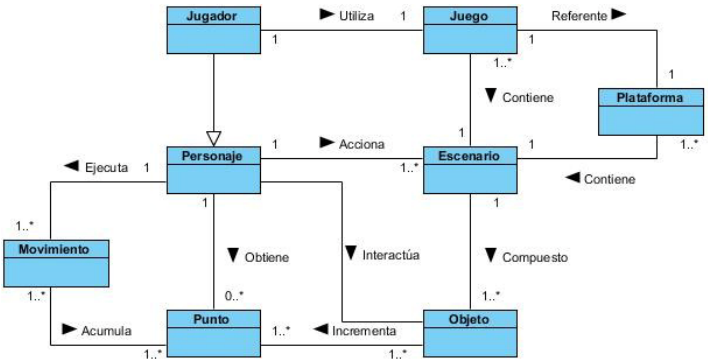

Figura 4. Modelo conceptual de la solución propuesta.

3.2.2 Vista de módulos

En esta vista se captura la descomposición funcional y las Capas del sistema. El sistema es descompuesto lógicamente en subsistemas, módulos, y unidades abstractas. Cada capa representa las distintas interfaces de comunicación permitidas entre los módulos [14]. Las dependencias entre paquetes resumen dependencias entre los elementos internos a ellos como se muestra en la figura 5 [11].

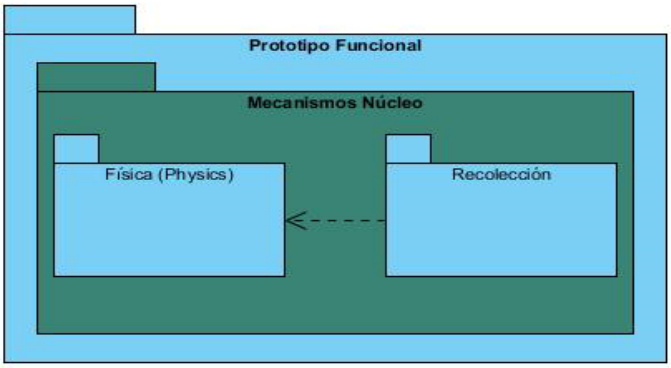

Figura 5. Diagrama de paquetes de la solución propuesta.

Para la representación de la interacción de las funcionalidades de la solución se utilizaron los diagramas de transición de estados para representar el comportamiento de objetos que se relacionan en procedimientos ejecutados del videojuego en tiempo real, tal y como se muestra en la figura 6.

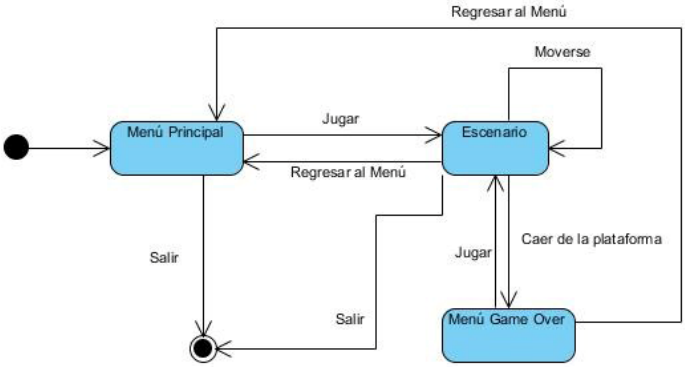

Figura 6. Diagrama de transición de estados: Mecanismo de Física.

3.2.3 Vista de código

En esta vista se organiza el código fuente representando la estructura y organización del prototipo funcional. Además, se puede apreciar la distribución de los componentes en cada una de las Capas.

La figura 7 visualiza desde una vista más cercana al código, cómo el prototipo funcional es dividido en componentes y muestra las relaciones de dependencias entre estos componentes.

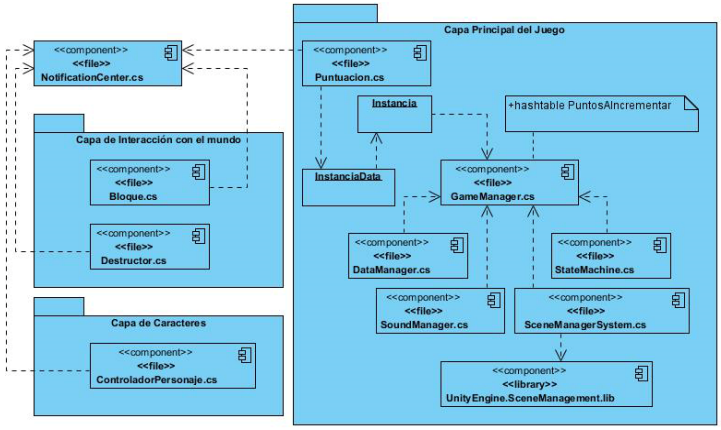

Figura 7. Diagrama de componentes de la solución propuesta.

3.2.4 Vista de ejecución

En la vista de ejecución se describe la estructura dinámica del videojuego en términos de sus elementos en tiempo de ejecución como se muestra en la figura 8. Algunos de los aspectos que se consideran en esta vista son: el desempeño y el entorno de ejecución [14].

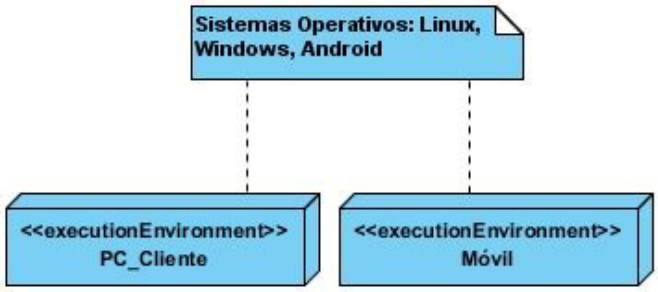

Figura 8. Diagrama de despliegue.

3.3 Evaluación de la arquitectura de *software* propuesta

El método de Análisis de Acuerdos de Arquitectura de *Software* (ATAM) está compuesto por nueve pasos divididos en cuatro fases, los cuales se adaptarán de acuerdo a los requerimientos del proyecto [15]. Después de la descripción de la solución propuesta y el análisis realizado, se procede a evaluar la arquitectura propuesta empleando este método.

Se seleccionó el método ATAM para la validación de la arquitectura propuesta, debido a que permite obtener los requerimientos de los atributos de calidad que son importantes para lograr cumplir las metas del contexto base que se desarrolla. La expresión concreta de estas metas son los escenarios de los atributos de

calidad que aparecen en las hojas del árbol de utilidad [16].

Los escenarios aplicados y las propuestas se unen cuando se pregunta cómo estos son soportados por la arquitectura. El resultado es un conjunto de decisiones arquitectónicas. Si estas decisiones son potencialmente problemáticas o especialmente importantes, o si afectan a más de un atributo, entonces deben ser registradas como riesgos. Por tanto, el método ATAM permite identificar los riesgos arquitectónicos que potencialmente prohíben a una organización conquistar las metas del contexto [16]; mientras que el esfuerzo dedicado a contrarrestar estos eventos garantiza una estructura lógica y robusta de las funcionalidades bases de un producto de *software* de calidad.

Árbol de utilidad: Los atributos presentes en el árbol de utilidad [16], usados para evaluar la arquitectura mediante el método ATAM, en conjunto con técnicas basadas en escenarios y prototipo, son los más relevantes a comprobar sobre el dominio de aplicación desarrollado (videojuegos), dado que aportan interpretaciones consistentes, pertinentes, facilitando el análisis de su comportamiento desde el núcleo arquitectónico hasta los horizontes de la arquitectura relacionados con las funcionalidades más básicas del videojuego (prototipo funcional). Las especificaciones de atributos pertenecen al modelo de calidad ISO/IEC 25010 [17] y sus descripciones se muestran en la tabla 1.

Tabla 1. Árbol de utilidad

Atributo	Subatributo	Escenario	Prioridad
Adecuación Funcional	Corrección Funcional	El videojuego debe proveer resultados correctos con el nivel de precisión requerido.	Alta
Eficiencia	Comporta-miento Temporal	Los tiempos de respuesta y procesamiento del videojuego con respecto a cada una de las acciones del usuario deben ser cortos.	Alta
	Utilización de recursos	Las cantidades y tipos de recursos utilizados cuando el videojuego lleva a cabo su función deben ser bajos.	Alta
Compatibili-dad	Coexistencia	El videojuego debe poder coexistir con otro *software* independiente, en un entorno común, compartiendo recursos comunes sin detrimento.	Media
	Interoperabili-dad	El videojuego debe poder intercambiar información y utilizar la información intercambiada con otro sistema.	Media
Usabilidad	Estética	La interfaz de usuario debe agradar y satisfacer la interacción con el usuario.	Media
	Aprendizaje	El usuario debe comprender el sistema con facilidad.	Media
	Operabilidad (Jugabilidad)	El sistema debe satisfacer la experiencia del jugador.	Media
Fiabilidad	Tolerancia	El sistema opera según lo previsto en presencia de fallos *hardware* o *software.*	Alta
	Disponibilidad	El sistema está	Alta

		disponible cuando se requiere su uso.	
Seguridad	Confidencialidad	El sistema protege contra el acceso a datos e información de personas no autorizadas, ya sea accidental o deliberadamente.	Media
	Integridad	El sistema previene accesos o modificaciones no autorizados a sus datos.	Alta
Mantenibilidad	Modularidad	Al realizar un cambio en un componente del sistema debe tener un impacto mínimo en los demás.	Alta
	Reusabilidad	Los elementos del videojuego deben ser reusables en otros desarrollos.	Alta
	Capacidad de ser modificado	El sistema puede ser modificado de forma efectiva y eficiente sin introducir defectos o degradar el desempeño.	Alta
Portabilidad	Adaptabilidad	El producto puede ser adaptado de forma efectiva y eficiente a diferentes entornos	Media
		determinados de *hardware*, *software*, operacionales o de uso.	
	Facilidad de instalación	El producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.	Media

3.4 Evaluación basada en escenarios con presencia de riesgos sobre la propuesta

Tabla 2. Descripción del Escenario Confidencialidad del atributo Seguridad

Atributo de Calidad	**Seguridad**
Subatributos/Sub-característica	Confidencialidad.
Objetivo	El sistema protege contra el acceso a datos e información de personas no autorizadas, ya sea accidental o deliberadamente.
Origen	Usuarios del sistema
Artefacto	Videojuego.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. Se accede al sistema sin autorización.	
Se accede al sistema sin autenticarse.	El sistema no posee autenticación de usuarios.
Medida de respuesta	
Desplegar el sistema.	

Tabla 3. Descripción del Escenario Integridad del atributo Seguridad

Atributo de Calidad	**Seguridad**
Subatributos/Sub-característica	Integridad.
Objetivo	El sistema previene accesos o modificaciones no autorizados a sus datos.
Origen	Usuarios del sistema.
Artefacto	Videojuego.
Entorno	El sistema desplegado.
Estímulo	**Respuesta: Flujo de eventos (Escenarios)**
1. Se accede al	

sistema sin autorización y se modifican las puntuaciones.	
Se accede al sistema sin autenticarse.	El sistema no posee autenticación de usuarios, por lo que se pueden modificar las puntuaciones.
Medida de respuesta	
Modificar datos del sistema.	

A partir de la información obtenida tras la aplicación del método ATAM como se muestra en las tablas 2 y 3, se identificaron dos riesgos en la arquitectura propuesta como se presentan en la tabla 4. Estos riesgos están asociados a los escenarios Seguridad: Integridad y Confidencialidad.

Tabla 4. Riesgos identificados

Escenario	Riesgo
Seguridad: El sistema protege contra el acceso a datos e información de personas no autorizadas, ya sea accidental o deliberadamente.	El sistema no posee autenticación de usuarios, por lo que cualquier usuario puede acceder al mismo.
Seguridad: El sistema previene accesos o modificaciones no autorizados a sus datos.	Comportamiento Temporal.

Además, como parte del proceso de validación de la arquitectura, para comprobar la calidad de la misma se utilizaron las siguientes métricas de calidad [18]:

- **Métrica de Complejidad estructural (S_i):** Para el cálculo de esta métrica se tuvo en cuenta el número de módulos subordinados a cada módulo del prototipo funcional, obteniéndose como valor: $S_i = 6$.
- **Métrica de Complejidad de datos (D_i):** Para la determinación de esta métrica se tuvieron en cuenta las variables de entrada y salida por cada uno de los módulos que comprende el videojuego desarrollado sobre la arquitectura propuesta, originándose como resultado: $D_i = 31$.
- **Métrica de Complejidad del sistema (C_i):** Para el cálculo de esta métrica se agruparon los valores obtenidos por S_i y D_i. Teniendo en cuenta esta adición se obtuvo un valor: $C_i = 37$.

A medida que crecen los valores de complejidad, crece la complejidad arquitectónica del sistema. Teniendo en cuenta que el prototipo funcional desarrollado sobre el motor de juego Unity 3D, se corresponde con un videojuego del género de plataformas, el mismo permitió comprobar la utilidad de la arquitectura en un escenario sencillo de este tipo con valor de $C_i = 37$. Sin embargo, muchos de los atributos de calidad evaluados arrojaron como resultados que la arquitectura de *software* propuesta es adaptable, reutilizable y modificable. Por tanto, en otro escenario de desarrollo el valor de C_i puede variar sin que cambien las propiedades básicas de la solución arquitectónica propuesta.

- **Métrica de Complejidad a nivel de componentes:** Para la determinación de esta métrica se diseñó el grafo de flujo para cada una de las operaciones contenidas en los componentes implementados. Posteriormente se calculó la complejidad ciclomática para cada grafo de flujo. Esto arrojó un valor promedio de 4 puntos, para un total de once componentes implementados de la solución con sus respectivas operaciones. Este valor puntual indica que se disponen a lo sumo de 4 caminos lógicos linealmente independientes para la ejecución de escenarios en sus métodos. Esto demuestra el grado de optimización de las funcionalidades desarrolladas como parte de la arquitectura base para el desarrollo de videojuegos. Además, este resultado denota la aplicabilidad en la solución arquitectónica propuesta de otros atributos de calidad, tales como: extensibilidad, fiabilidad y mantenibilidad.

3.5 Evaluación basada en prototipo

Para evaluar la AS fue implementado un prototipo funcional de un videojuego de plataformas según se muestra en la figura 9, en el que, luego de tener una comprensión mejor del mismo mediante las vistas arquitectónicas modeladas, se puede apreciar la interacción de los componentes del prototipo y con la arquitectura propuesta como solución a cada uno de los mecanismos [19] del videojuego.

Figura 9. Escena de juego del prototipo funcional.

Para ejecutar cada uno de los estados el personaje se vale del componente animator de Unity, contando con los siguientes scripts para completar las acciones:

ControladorPersonaje: Se encarga de determinar en qué estado se encuentra el personaje y realizar la acción correspondiente. Se encuentra suscrito al NotificationCenter (Observer) para posibilitar que en el momento que empiece a correr y, por consiguiente, se generen los elementos y objetos del videojuego.

Puntuación: Controla la puntuación ejecutando las acciones de incrementar 1 punto si el personaje se sitúa sobre un bloque o 5 puntos en caso de que acumule un objeto. Suscrito también al NotificationCenter, accede a las tablas de puntos contenidas en el GameManager, así como al DataManager para obtener la puntuación máxima guardada ahí y al método de guardar para sobrescribirla en caso de que se acumule una puntuación máxima superior. Para interactuar entre escenas se usa el SceneManager, que se encarga de los cambios de escenas, siendo implementado ahí los métodos que se encargan de dicho cambio entre escenas.

4. Conclusiones

Con la realización de esta investigación, se define y se valida una arquitectura de *software* para el desarrollo de videojuegos, probada sobre un prototipo funcional de un videojuego del género de plataformas, la cual permite reutilizar componentes, facilitar la base para la implementación y el uso de recursos, contribuyendo de este modo al desarrollo base de futuros videojuegos. La combinación de los patrones arquitectónicos: arquitectura en tres Capas y en componentes, permitió desarrollar una AS que satisface los atributos de calidad: reusabilidad, mantenibilidad, extensibilidad y eficiencia, que eran pautas a tener en cuenta a la hora de proponer una solución. El uso de las vistas arquitectónicas propuestas por Robert Nord, permitió tener una mayor visión de la propuesta de solución implementada en el prototipo funcional. Las evaluaciones realizadas, mediante el método ATAM: basadas en escenarios y prototipo, arrojaron la presencia de riesgos en el prototipo funcional y sobre la arquitectura propuesta, los cuales son considerados para futuros desarrollos de videojuegos.

5. Referencias

[1] Stack, P. History of video game consoles. Time Magazine website 2005; Available from: http://www.time.com/time/covers/1101050523/console_timeline. [Ene. 19, 2018].

[2] Greenslade, A. Gamespeak: A glossary of Gaming Terms. 2006; Available from: https://archive.is/TbSs#selection-147.12-147.29. [Ene. 20, 2018].

[3] Ward, J. What is a Game Engine?. 2008; Available from: https://www.gamecareerguide.com/features/529/what_is_a_game_.php. [Ene. 20, 2018].

[4] Technologies, U. Motores Gráficos. 2011; Available from: http://www.mat.ub.edu/futurs_ub/activitats/Matefest/2011/triptics/motoresgraficos.pdf. [Ene. 21, 2018].

[5] Pressman, R.S. "Ingeniería de software. Un enfoque práctico". Séptima Edición. 2010.

[6] Camacho, E., Fabio Cardeso, Gabriel Nuñez. "Arquitecturas de software. Guía de estudio". 2004.

[7] GESPRO 16.05. Herramienta de Gestión de Proyectos. 2017. Project Management Suite. Universidad de las Ciencias Básicas. Available from: https://gespro.vertex.prod.uci.cu. [Ene. 21, 2018].

[8] Kruchten, P. "The Rational Unified Process". Addison Wesley Longman, 2003.

[9] Nord, R.L. "Un modelo general de diseño de arquitectura de software derivado de cinco enfoques industriales". 2007.

[10] Gutiérrez, H., Ricardo Emmanuel; Francisco J. Álvarez, Jaime Muñoz-Arteaga. Arquitectura de Software para Juegos Serios con Aspectos Culturales: Caso de Estudio en un Videojuego para Fórmulas Temperatura. 2013; Available from: https://www.researchgate.net/publication/236162869_Arquitectura_de_Software_para_Juegos_Serios_con_Aspectos_Culturales_Caso_de_Estudio_en_un_Videojuego_para_Formulas_Temperatura. [Ene. 21, 2018].

[11] Cazalla, C.E. "Diseño y desarrollo de un prototipo básico de un videojuego plataformas en 2D". 2014.

[12] GitLab. GitLab Community Edition. 2017; Available from: https://gitlab.prod.uci.cu/users/sign_in. [Ene. 21, 2018].

[13] Hernández, P.A. "Marco de trabajo ingenieril para el proceso de desarrollo de videojuegos". RACCIS 7(1), 13-26, Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de Software. 2017.

[14] Reynoso, C.y.K., Nicolás. "Estilos y Patrones en la Estrategia de Arquitectura de Microsoft". 2004.
[15] Delgado, A., Alberto Castro, Martín Germán. "Evaluación de Arquitecturas de Software con ATAM". Universidad de la República, Facultad de Ingeniería, Instituto de Computación. Uruguay. 2007
[16] Clements, P.K., R & Klein, M. "Evaluating software architectures: Methods and case studies". Boston: Addison-Wesley. 2001.
[17] ISO/25010. ISO/IEC 25010. 2011 Available from: http://iso25000.com/index.php/normas-iso-25000/iso-25010. [Ene. 21, 2018].
[18] Pallares, E.Y. y .P.V., Johana Andrea. "Métricas del Modelo del Diseño". 2012; Available from: http://ing-software3.blogspot.com/2012/11/metricas-del-modelo-del-diseno.html. [Mar. 22, 2018].
[19] Adams, J.D.E. "Game Mechanics. Advanced Game Design". 2012.