Universidad de las Ciencias Informáticas

Facultad 5

PAQUETE DE MECÁNICAS PARA EL DESARROLLO DE VIDEOJUEGOS DE TIPO ESTRATEGIA TÁCTICA SOBRE UNITY 3D

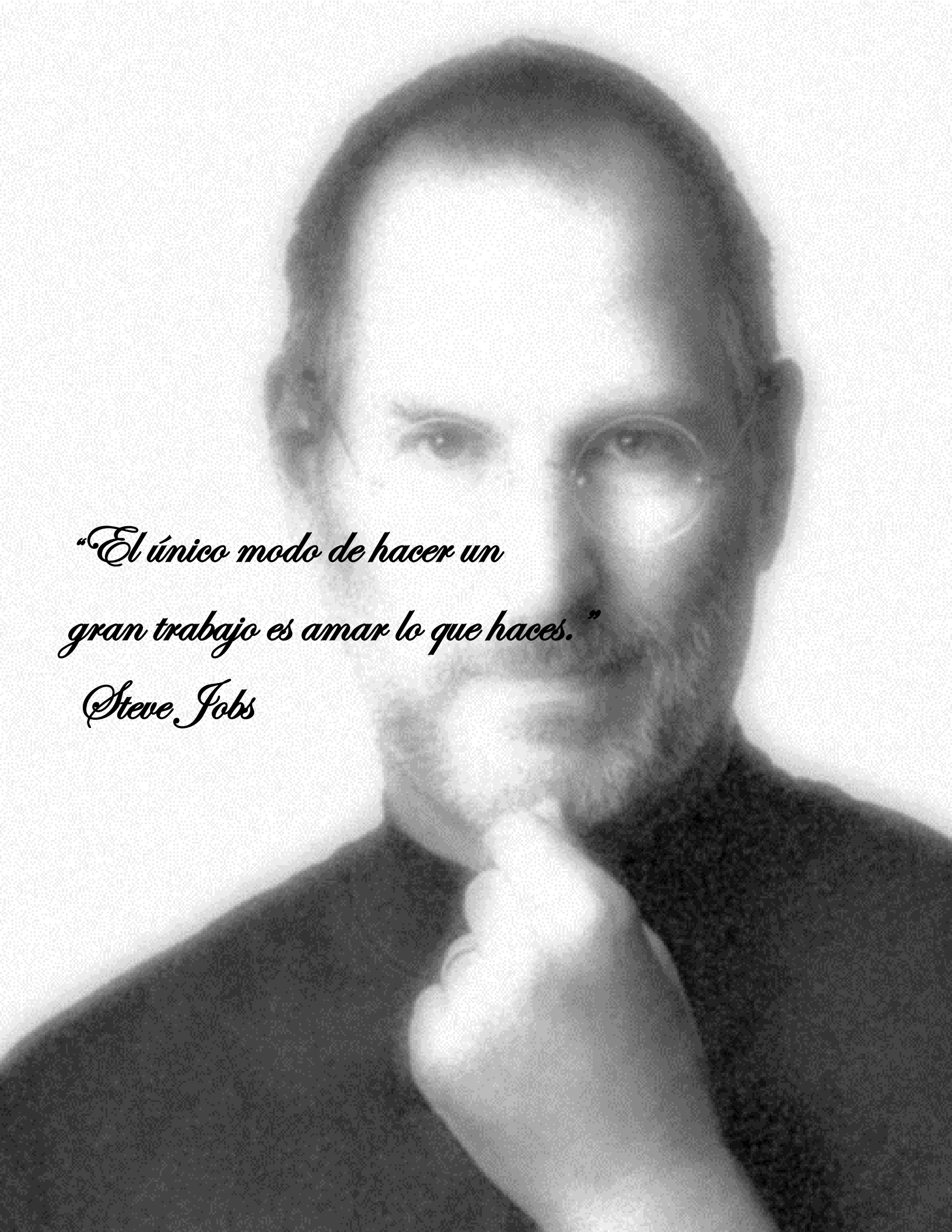
Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Dayron Suárez del Toro

Tutores: Dr.C Omar Correa Madrigal

Ing. Enelis Blanca Cuba Rondón

La Habana, 2016


“El único modo de hacer un
gran trabajo es amar lo que haces.”
Steve Jobs

DECLARACIÓN DE AUTORÍA

Declaro ser autor único de este trabajo y autorizo a los tutores Omar Correa Madrigal, Enelis Blanca Cuba Rondón y al Centro I+D+i Entornos Interactivos 3D (Vertex), de la Universidad de las Ciencias Informáticas, para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

_____

Dayron Suárez del Toro

Autor

_____

Dr.C Omar Correa Madrigal

Tutor

_____

Ing. Enelis Blanca Cuba Rondón

Tutor

DATOS DE CONTACTO

Generales de los tutores:

Nombre y apellidos: Dr.C Omar Correa Madrigal

Especialidad: Ciencias Técnicas

Síntesis del tutor:

- Actual director del centro Vertex.
- Profesor investigador en los temas de Visualización Gráfica y Realidad Virtual.
- Líder de la línea de investigación universitaria Visualización y Realidad Virtual desde el 2011.
- Profesor del Diplomado de Realidad Virtual.
- Forma parte del claustro de la Maestría de Computación Avanzada.
- Categoría docente: asistente.
- 9 años de experiencia.

Correo electrónico: ocorrea@uci.cu

Nombre y apellidos: Ing. Enelis Blanca Cuba Rondón

Especialidad: Ingeniería en Ciencias Informáticas

Síntesis del tutor:

- Especialista A, jefa del grupo de inteligencia empresarial del centro Vertex.
- Representante de los estudios que se realizan en la tecnología referente a los entornos virtuales en OpenSim.
- 4 años de experiencia.

Correo electrónico: ebcuba@uci.cu

DEDICATORIA

A mi madre.

A mi hijo.

A mi mujer Enelis Cuba.

A mi familia.

AGRADECIMIENTOS

A mi madre y mi padre por darme la vida, especialmente a mi madresita querida por su inmenso amor incondicional, dedicación y sacrificio para hacerme la persona que hoy soy.

A mi hijo por el amor que me ha transmitido siempre, aun sin conocerme.

A mi compañera de vida Enelis Cuba Rondón por su amor, tolerancia, apoyo, comprensión en los buenos y malos momentos y por darme uno de mis mayores tesoros.

A toda mi familia por confiar en mí, por su inmenso amor y por apoyarme siempre.

A todos mis amigos, especialmente a mis compañeros José Manuel, Enriquito, Laura, Ernesto y Javier Rodríguez por su incondicionalidad.

A mis tutores por su gran confianza y apoyo dentro y fuera del periodo de tesis.

A todos los profesores que contribuyeron en mi formación.

A la revolución cubana por darme la posibilidad de superarme.

RESUMEN

El videojuego es uno de los medios audiovisuales de mayor aceptación en la actualidad y se ha mantenido en constante evolución desde su origen en la década de 1940. Antiguamente para desarrollarlos era necesario poseer un gran dominio de la programación de bajo nivel. Hoy en día existen los llamados motores de videojuegos que proporcionan a los desarrolladores un conjunto de funcionalidades reutilizables que pueden emplear y así centrarse en los elementos propios del videojuego. En este trabajo se desarrolló un paquete con las principales mecánicas que componen a los videojuegos de tipo Estrategia Táctica, definidas mediante el estudio de tres ejemplos pioneros de este subgénero. El mismo fue realizado para el motor de videojuegos Unity 3D y guiado por la metodología de desarrollo de software XP, con el objetivo de brindar a los desarrolladores una base técnica reutilizable, flexible y extensible, para evitar su implementación en cada proyecto y así poder dedicar más tiempo al enriquecimiento del juego. Para validar que la solución cumpliera con los requerimientos definidos por el cliente, se aplicaron pruebas de aceptación al finalizar cada iteración y pruebas unitarias para asegurar el correcto funcionamiento del código. También se realizó un demo con el paquete para demostrar su reusabilidad, flexibilidad y extensibilidad. Por último fue probado el rendimiento de este en dicho demo. Lo cual demostró que se cumplió satisfactoriamente el objetivo del trabajo.

Palabras clave: Estrategia Táctica, mecánicas, paquete, Unity 3D, videojuego.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los niños desde edades tempranas empiezan a conocer el mundo mediante juegos que les permiten socializarse y relacionarse con su entorno. Jugar para ellos es una capacidad que logran naturalmente, por medio de la cual es posible representar sus sentimientos y vivencias. A lo largo de la vida, el ser humano ha utilizado los videojuegos como instrumento de entretenimiento, pues simulan mediante medios digitales muchos de estos juegos que el hombre emplea y además pueden incorporarles elementos de ficción o incluso representar espacios completamente fantásticos. Con ellos adquieren habilidades y destrezas que le permiten desempeñarse en su existencia, al facilitar su integración, su creatividad y favoreciendo el aprendizaje físico-motriz.

Los videojuegos y su proceso de desarrollo, desde su aparición hasta la actualidad, han mantenido una constante evolución en cuanto a: formas de visualización, modos de control, jugabilidad y concepción. Los primeros videojuegos no visualizaban más que puntos, líneas o figuras geométricas en una pantalla y se controlaban mediante el uso de simples potenciómetros o pulsadores (García 2014). Sin embargo, hoy en día, muestran gráficos 3D de alto realismo y los modos de control son tan abundantes como diversos, yendo desde teclado o mandos hasta reconocimiento de movimiento mediante el uso de cámaras. El aumento de la popularidad de los videojuegos a nivel mundial ha provocado un incremento en la variedad de géneros, temáticas y en la aplicabilidad en diferentes áreas; logrando la inmersión de los usuarios en entornos de acción, ficción, aventura, deporte; accesibles desde videoconsolas, ordenadores y dispositivos móviles. Los videojuegos internamente están conformados por diferentes mecánicas, las cuales se componen por conjuntos predeterminados de instrucciones, mecanismos o reglas que garantizan su correcto funcionamiento y le brindan los controles necesarios al usuario para interactuar con los elementos del juego.

En Cuba, uno de los centros pioneros en el desarrollo de videojuegos es la Universidad de las Ciencias Informáticas, específicamente el Centro de I+D+i Vertex, Entornos Interactivos 3D. En este se tiene como meta desarrollar videojuegos de diferentes géneros, aplicables al entretenimiento, la educación y la salud. Actualmente se proyecta el desarrollo de videojuegos de tipo Estrategia Táctica sobre el motor de videojuegos Unity 3D. La mayoría de los desarrolladores del centro no cuentan con suficiente experiencia en el trabajo con este motor, así como en las particularidades técnicas de este tipo de juego. Por otro lado, los videojuegos desarrollados con anterioridad, no aportan en este sentido ya que sus componentes tienen una alta dependencia con los elementos propios del juego, por ejemplo su relación con el argumento y

mecánicas específicas. Esto afecta su valor como componente reutilizable, que se refleja en su flexibilidad y extensibilidad ante un nuevo contexto de aplicación. Además, el análisis de distintos paquetes para Unity 3D permitió identificar que sus funcionalidades no se ajustan al tipo de juego a desarrollar y que también son privativos.

En este sentido se tiene como **problema de investigación**: ¿cómo contribuir al desarrollo de videojuegos de tipo Estrategia Táctica sobre la base de componentes reutilizables en Unity 3D? Y se define como **objeto de estudio** el desarrollo de videojuegos.

Para dar solución al problema se define como **objetivo** de la presente investigación: desarrollar un paquete de mecánicas reutilizables para videojuegos de tipo Estrategia Táctica sobre Unity 3D.

Por tanto, se establece como **campo de acción** las mecánicas de videojuegos de tipo Estrategia Táctica.

Para dar cumplimiento al objetivo general del trabajo se definieron como tareas de investigación las siguientes:

- Elaboración del marco teórico de la investigación a partir del estado del arte de las mecánicas de videojuegos de tipo Estrategia Táctica.
- Caracterización de paquetes para Unity 3D destinados al desarrollo de videojuegos de tipo Estrategia Táctica existentes tanto a nivel nacional como internacional, con el objetivo de identificar los principales elementos que los componen y seleccionar aquellas funcionalidades que puedan incluirse en la solución propuesta.
- Ejecución del levantamiento de requisitos funcionales y no funcionales de la solución.
- Desarrollo de los artefactos ingenieriles necesarios para los flujos de trabajo de Análisis y Diseño.
- Implementación de la solución.
- Desarrollo de pruebas para validar el cumplimiento de los requerimientos definidos.

Para la realización de la investigación y elaboración del presente trabajo se utilizarán varios **métodos científicos de investigación**, entre los cuales se pueden mencionar:

Métodos teóricos:

Histórico – Lógico: se empleó para la fundamentación y sistematización de los aspectos teóricos contemplados en el desarrollo de la investigación acerca de la evolución y las tendencias actuales del desarrollo de mecánicas de videojuegos y demás elementos relacionados con el contenido del trabajo.

Analítico – Sintético: se utilizó para analizar los videojuegos de tipo Estrategia Táctica existentes y arribar a conclusiones sobre sus características y mecánicas esenciales.

Métodos empíricos:

Observación: Se empleó como método referencial al observar el comportamiento de distintos videojuegos de Estrategia Táctica, para establecer una comparación y determinar las características y mecánicas comunes que poseen, con la integración a la solución propuesta por el autor.

A continuación, se muestra la estructura del presente trabajo, incluyendo una síntesis de los capítulos y secciones fundamentales:

Capítulo 1. Fundamentación teórica.
En este capítulo se realiza un estudio acerca de elementos necesarios para la definición de videojuegos, sus géneros y sus mecánicas. Se analizan tres videojuegos de Estrategia Táctica para establecer los mecanismos que debe presentar la solución. También se describen algunas soluciones homólogas existentes, así como metodología y herramientas a utilizar para el desarrollo de la solución.

Capítulo 2. Solución propuesta.
En este capítulo se definen los requisitos no funcionales y funcionales de la solución, estos últimos mediante las historias de usuario definidas por la metodología de desarrollo de software a utilizar. Se describe también la propuesta de solución, se realiza el plan de entregas y el de iteraciones, así como la arquitectura, la estructura de clases y los patrones empleados para el desarrollo de la misma.

Capítulo 3. Implementación y prueba.
En el capítulo se puede visualizar como se llevó a cabo la implementación del sistema, definiéndose diferentes tareas por iteraciones. Se realizaron pruebas unitarias llevadas a cabo por el programador para asegurar el correcto funcionamiento del código y de aceptación para verificar si la solución cumplía con las especificaciones del cliente. Se creó un demo sencillo para verificar la reusabilidad, flexibilidad y extensibilidad de la solución y por último se probó su rendimiento con la herramienta Profiler de Unity 3D.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

En este capítulo se definen los principales conceptos asociados al dominio del problema, como videojuegos, sus géneros y dentro de estos se destaca el de Estrategia Táctica, sus mecánicas principales y su proceso de desarrollo. Se efectúa el análisis de tres videojuegos clásicos de Estrategia Táctica para agrupar sus comportamientos principales en mecánicas que el autor define y se realiza un estudio del estado del arte sobre los paquetes existentes para crear videojuegos de estrategia con Unity 3D, además de describir las principales tecnologías, metodología y herramientas a emplear en el desarrollo de la solución.

1.1 Videojuegos

Primeramente, antes de hablar sobre videojuegos es necesario conocer qué se entiende por juego, debido a la estrecha relación que guardan estos términos.

Se puede entender por juego como: “todas las actividades que se realizan con fines recreativos o de diversión, que suponen el goce o el disfrute de quienes lo practican” (Google 2013). Mike Zyda (Zyda 2005), propone un juego como: “una prueba física o mental, llevada a cabo de acuerdo con unas reglas específicas, cuyo objetivo es divertir o recompensar al participante”.

Ya abordada la definición de juego se mostrarán varias definiciones de qué es un videojuego según los criterios propuestos por diferentes autores.

Dentro del mundo del entretenimiento electrónico, un videojuego normalmente se suele asociar a la evolución de uno o varios personajes principales, o entidades que pretenden alcanzar determinados objetivos en un mundo acotado. Normalmente este mundo puede estar compuesto por una serie de escenarios virtuales y tiene asociado un conjunto de reglas que determinan la interacción con el mismo, con el fin de plantearle retos al usuario que le garanticen emociones, entretenimiento, conocimiento y diversión (Fernández y Angelina 2012).

“Videojuego es aquel programa informático, normalmente asociado a un hardware específico, que recrea un ejercicio sometido a reglas, se debe lograr uno o varios objetivos, donde los jugadores pueden interactuar y tomar decisiones” (Gavaldà y Navarro 2008).

Según el diccionario de la Real Academia Española (RAE), videojuego es un *"dispositivo electrónico que permite, mediante mandos apropiados, simular juegos en las pantallas de un televisor o de un ordenador"* (Española 2015).

Los videojuegos incluyen uno o varios usuarios llamados *players* o jugadores, los cuales mantienen una interacción constante con varias interfaces, como pueden ser los *joysticks* o controles, teclado, *mouse* y un dispositivo de video. Pueden recrear situaciones y entornos virtuales de cualquier índole (fantasía, acción, lógica, deporte, terror), donde las reglas de la física pueden romperse y distorsionarse, mundos fantásticos o situaciones con un realismo extraordinario (Gómez y Ozete 2013).

Al analizar los criterios anteriormente planteados y teniendo en cuenta sus elementos comunes, se define para este trabajo un videojuego como: programa informático ejecutable en distintos dispositivos electrónicos (tales como: computadoras, teléfonos móviles y consolas), capaces de simular juegos mediante la recreación de diversos espacios y situaciones reales o de ficción. Permiten la interacción de los usuarios regida por reglas para cumplir objetivos o retos específicos y les aporta conjuntamente, emociones, entretenimiento, habilidades y conocimientos.

1.2 Géneros de videojuegos

Debido al surgimiento constante de nuevos estilos y mecanismos de videojuegos que no entraban en las definiciones ya existentes, se han ido creando nuevos géneros para su clasificación mientras otros caen en desuso y sólo se utilizan para clasificar videojuegos antiguos. Los más generales actualmente son los siguientes: Aventura, Deportivos, Disparo, Educativos, Lucha, Rompecabezas, Rol, Simulación, Estrategia, Carreras, Arcade, Lógica y nuevas tendencias (Díaz 2015). Luego de estudiadas las características de estos, se pudo identificar que el subgénero Estrategia Táctica del que se hará énfasis en este trabajo, forma parte del género Estrategia (Gavaldà y Navarro 2008).

1.2.1 Género Estrategia

Los videojuegos enmarcados por este género, se caracterizan por tener un escenario donde el jugador debe decidir la estrategia que seguirá para conseguir un objetivo específico. Normalmente, están relacionados con la estrategia militar, y las decisiones pueden ser: por turnos, donde los jugadores definen su estrategia en un orden (primero uno y después otro como en un juego de ajedrez o billar), o en tiempo real, donde los jugadores compiten también en rapidez.

Lo más característico en este tipo de videojuego es que se debe reflexionar sobre cuál va a ser la acción a tomar en función de una situación más o menos conocida. El jugador puede intuir los recursos del enemigo y conoce los suyos propios. En función de todo ello piensa una jugada y la lleva a cabo. Un clásico ejemplo de videojuego de estrategia perteneciente al subgénero Estrategia en Tiempo Real (RTS, Real Time Strategy, por sus siglas en inglés), es StartCraft, mostrado en el anexo 6 (Gavaldà y Navarro 2008).

1.2.2 Subgénero Estrategia Táctica

En estos videojuegos, la estrategia se realiza en un ámbito global y luego se tienen pequeñas acciones que son las tácticas, en las que, para llevarlas a cabo, existe conocimiento pleno de la disposición del enemigo en tiempo real para una acción concreta (Gavaldà y Navarro 2008). A diferencia de otros subgéneros de estrategia, en este no existe una administración a gran escala de la economía del juego, solo se controla un pequeño grupo de personajes que no varían en número. Sin embargo, en el subgénero RTS el usuario controla ejércitos enteros y puede realizar construcciones que le permiten expandirse en el terreno para ganar fuerzas y población. Esto le permite al jugador desarrollar el juego y realizar estrategias de manera diferente en cada partida. Por otro lado, en los videojuegos de Estrategia Táctica son pocas las posibilidades de variar la estrategia, estos son más lineales y se rigen por el hilo conductor de la historia propia del juego. Cada personaje posee características y habilidades diferentes, pero como grupo se complementan los unos a los otros para encarar las diversas situaciones que surgen durante cada misión. En estas, la presencia de cada personaje es de vital importancia y sería imposible lograr el objetivo final con la ausencia de alguno. Un claro ejemplo de este subgénero es la saga Commandos, de Pyro Studios (Gavaldà y Navarro 2008).
Otro de los términos que es necesario dominar para el desarrollo del trabajo, es el de mecánicas de videojuegos, pues ellas garantizan internamente el funcionamiento del videojuego.

1.3 Mecánicas de videojuegos

Lara Sánchez Coterón (Coterón 2012) propone en su tesis doctoral, que las mecánicas de videojuegos, desde la perspectiva del diseñador, dan lugar al comportamiento dinámico del sistema una vez que el videojuego está en ejecución, conduciendo a experiencias estéticas específicas para el jugador. Son la representación de los datos, las reglas, mecanismos, procesos y los algoritmos, internamente contenidos en un videojuego que definen cómo progresa y garantizan un correcto funcionamiento. Las mecánicas son acciones, comportamientos y control de mecanismos de los que dispone el jugador dentro del contexto del juego (Coterón 2012).

Las mecánicas también se encuentran clasificadas por categorías según la naturaleza de los mecanismos que las componen. A continuación pueden observarse cinco de estas categorías propuestas por Adams y Dormans (Adams y Dormans 2012), para agrupar mecánicas según su relación y función en el videojuego. Gran parte de las mecánicas contenidas en estas categorías están presentes en los videojuegos actuales.

Física: la física en los videojuegos se suele definir como la ciencia del movimiento y fuerza, en el mundo del juego (que pueden desafiar las leyes de la física del mundo real). Las mecánicas de física están presentes al mover personajes de un lugar a otro, saltar, conducir vehículos, al calcular la posición de un elemento en la escena, la dirección en la que este se mueve, así como la detección de colisiones con otros objetos.

Economía interna: son las mecánicas de las transacciones que involucran los elementos del videojuego que se recogen, se consumen o se negocian. La economía interna de estos típicamente abarca elementos fácilmente identificables como: recursos, dinero, energía y municiones. Sin embargo, la economía de un videojuego no se limita a elementos tangibles y concretos; también puede incluir abstracciones tales como la salud, la popularidad, poderes mágicos y habilidades.

Progresión: en muchos videojuegos, tradicionalmente, el jugador tiene que llegar a un lugar en particular para rescatar a alguien o para derrotar al principal malhechor y completar el nivel. En este tipo de videojuego, el progreso del jugador está estrechamente controlado por una serie de mecanismos que bloquean o desbloquean el acceso a ciertas áreas. Palancas, interruptores, y espadas mágicas que le permiten destruir ciertas puertas son ejemplos típicos de tales mecánicas de progresión.

Maniobra táctica: los videojuegos pueden tener mecánicas relacionadas con la colocación de unidades en el mapa con ventajas ofensivas o defensivas. La maniobra táctica es fundamental en la mayoría de los videojuegos de estrategia, pero también se encuentran en algunos de roles y simulación. Las mecánicas que rigen maniobras tácticas, típicamente especifican qué ventajas estratégicas cada tipo de unidad puede ganar al estar en cada posible ubicación. Estas aparecen en muchos juegos de mesa como el ajedrez, las damas chinas y en videojuegos de estrategia.

Interacción Social: estas mecánicas permiten la interacción social en el mundo del videojuego entre varios jugadores con fines e intereses comunes. Muchos videojuegos en línea incluyen la mecánica que

recompensa dar regalos, invitar a nuevos amigos a unirse y participar en otras interacciones sociales. Además, los videojuegos de estrategia podrían incluir reglas que rigen la formación y ruptura de las alianzas entre los jugadores.

Las mecánicas pueden ser definidas muy generales y englobar múltiples mecanismos o pueden ser subdivididas en otras de menor amplitud y más específicas, tan pequeñas como se decida en el proceso de desarrollo del videojuego.

En el siguiente apartado se realiza el análisis de los videojuegos: Commandos, Desperados y Robin Hood, los primeros del subgénero Estrategia Táctica en salir al mercado. El objetivo es identificar los comportamientos principales que presentan y agruparlos en mecánicas con nombres bien definidos, teniendo en cuenta los elementos conceptuales de las categorías mencionadas en esta sección. Las mecánicas identificadas durante este análisis se asumirán a lo largo del trabajo como las mecánicas elementales de los videojuegos de Estrategia Táctica que no deben faltar en la solución.

1.4 Mecánicas elementales de videojuegos de tipo Estrategia Táctica

En la actualidad es constante el surgimiento de nuevos videojuegos pertenecientes al subgénero Estrategia Táctica. Muchos tienden a emplear estilos de juego diferentes. Lo que los diferencian un poco en cuanto a características y jugabilidad de los primeros en surgir. Se pueden mencionar numerosos videojuegos de este tipo, tales como: Commandos, Men of War, Panzers Phase*,* Chicago 1930*,* Hidden & Dangerous, Jagged Alliance, Desperados, Silent Storm, Robin Hood: The Legend of Sherwood y Korea: Forgotten Conflict. La saga Commandos es el primero en salir al mercado y ejemplo clásico empleado para referirse a este subgénero (Gavaldà y Navarro 2008), aunque existen varios con características muy similares como: Robin Hood: The Legend of Sherwood, Korea: Forgotten Conflict, Desperados y Chicago 1930.

Commandos salió al mercado el 31 de julio de 1998, disponible para las plataformas Macintosh y Windows. El objetivo de este es dirigir a un grupo de comandos militares a lo largo de varios escenarios en un ambiente de la Segunda Guerra Mundial (Studios 1998). Este videojuego causó gran impacto y aceptación en todo el mundo por su novedoso estilo de juego, según publicaciones en diversos sitios web en Internet. Por tal motivo inspiró a varias empresas de este mercado a desarrollar videojuegos con estas características. En este epígrafe se analizarán sus comportamientos para tomarlos como referencia y definir las mecánicas

principales de los videojuegos de este subgénero. En la siguiente figura se puede observar una captura de pantalla del videojuego.

Figura 1. Captura de pantalla del videojuego Commandos.

En Commandos, la cámara del juego se encuentra situada sobre el terreno y enfoca gran parte de este desde una proyección isométrica. Se puede mover vertical y horizontalmente con las teclas de dirección del teclado o acercando el cursor a los bordes de la pantalla. Con las teclas "+" y "-" del área de teclas numéricas del teclado se puede manejar el enfoque de la cámara. Luego de analizado el comportamiento de la cámara y teniendo en cuenta que es un objeto que tiene una traslación en la escena (relacionado con la categoría de mecánica Física), se decide nombrar a la mecánica que maneje los mecanismos de la proyección, el control y el enfoque de la cámara, con el nombre de **Cámara**.

La interfaz en el videojuego presenta dos áreas principales. Una en la esquina superior izquierda, donde se encuentran retratos de cada personaje jugador (PC, *Person Character*, por sus siglas en inglés) usados para seleccionarlos e indicar los que están seleccionados, junto con su respectiva barra de salud. La otra área está situada en la esquina inferior derecha en forma de mochila. Esta contiene elementos que representan las acciones propias del PC. El resto de la interfaz presenta elementos con funcionalidades adicionales para facilitar el desempeño del juego como el mapa y la opción para tender en el suelo y poner de pie a los PC. Como la interfaz maneja la interacción entre el usuario y los PC y además controla

elementos de economía interna (por ejemplo, la barra de salud) el autor decide nombrar la mecánica que agrupa todos esos mecanismos con el nombre de **Interfaz del juego** que hace referencia a HUD, acrónimo del término inglés head-up display (presentación de información).

El usuario puede controlar las unidades seleccionándolas e indicándoles ejecutar determinada acción. La selección de estas se realiza de cuatro maneras:

- Dando clic en la escena sobre un PC (esta selección solo admite seleccionar uno a la vez).
- Dando clic sobre el retrato de algún PC en la interfaz. Con la combinación *"Ctrl"* + clic en estos retratos se pueden seleccionar simultáneamente más de un PC.
- Con las teclas del "1" al "7" del teclado alfanumérico correspondiente a cada PC.
- Manteniendo presionado y arrastrando el clic derecho, se genera un cuadro de selección y todos los PC en la escena que se encuentren dentro de este son seleccionados simultáneamente.

En los mecanismos de selección de Commandos se ve presente la física al interactuar con los PC para orientar un destino determinado en el terreno y al indicar que ejecute interactúe con un objeto determinado de la escena. También se aprecian elementos de maniobra táctica, pues con ellos se pueden posicionar los PC en lugares estratégicos, de tal manera que faciliten cumplir con los objetivos de las misiones del videojuego. Según lo anteriormente planteado y teniendo en cuenta el significado de selección definido por la Real Academia Española: "Acción y efecto de elegir a una o varias personas o cosas entre otras, separándolas de ellas y prefiriéndolas" (Española 2015), se decide nombrar **Selección** a la mecánica que engloba los mecanismos de selección y control de PC.

Los PC y los personajes no jugadores (NPC, del inglés *Non-Person Character*) en el juego pueden correr, caminar y arrastrarse hacia un destino determinado. En el caso de los PC se les puede indicar con un clic simple y solo correr con doble clic. La velocidad de la traslación de las unidades depende de la forma de locomoción que ejecuten. Basándose en el concepto definido por la Real Academia Española, se entiende por locomoción: *"traslación de un lugar a otro"* (Española 2015). Esta traslación puede ser de objetos, animales o seres humanos mediante un mecanismo de propulsión natural o extra, permitiéndoles cambiar de localización en el espacio. Los seres humanos presentan varias formas de locomoción, por ejemplo: caminar, saltar, correr, trotar, nadar, volar, rodar, arrastrase y gatear (Jáuregui 2013). Todas estas acciones están dentro de la categoría de mecánicas Física, pero para mayor comprensión del autor en la investigación

se decidió llamar en el presente trabajo **Locomoción,** a la mecánica que engloba todos estos mecanismos encargados de los comportamientos de traslación de unidades en el juego.

Por otro lado**,** al seleccionar un PC en Commandos, se muestran en la interfaz de juego las acciones que este puede realizar, pero si se selecciona más de uno, aparecen solo las comunes para todos. Las acciones se pueden activar dando clic sobre alguna o con la tecla que le corresponde. Para cancelar la que esté activa, bastará con presionar el clic derecho. Existen acciones que no dependen de recursos disponibles como agarrar y enterrarse. Y existen otras donde los recursos se pueden agotar o no, como disparar con una pistola de balas infinitas o disparar con un rifle con cantidad limitada. Esta mecánica tiene estrecha relación con la categoría Economía Interna debido a que controla recursos como armas y objetos y tiene en cuenta la disponibilidad de estos. Para identificarla en la presente investigación será conocida como **Acciones** y controlará los mecanismos encargados de la activación, visualización y ejecución de acciones de las unidades.

Otra de las peculiaridades que se destaca en este videojuego es el empleo de NPC. Como se mencionaba anteriormente, estos hacen uso de la locomoción y acciones para realizar comportamientos como: detección de personajes en un rango de visión determinado, persecución, patrullaje en grupo o individual, ataque y alarma. Este trabajo tratará solamente las mecánicas de interacción entre usuario y PC ya definidas y se tendrá en cuenta para futuras investigaciones, la implementación de la Inteligencia Artificial (IA) de los NPC, debido al tiempo establecido para la investigación.

Aunque se hayan definido las mecánicas principales que posee este subgénero a partir del análisis de Commandos, es necesario verificar si los mecanismos que presentan se mantienen en otros ejemplos de estos videojuegos.

1.4.1 Análisis de videojuegos Desperados y Robin Hood

En esta sección se realiza el análisis de las mecánicas de los videojuegos Desperados y Robin Hood: The Legend of Sherwood por ser los primeros sucesores de Comandos. Por último, se describen detalladamente todos los mecanismos que las conformarán.

Desperados*:* fue desarrollado por Spellbound Entertainment[1] y lanzado en 2001. Se desenvuelve en un ambiente del salvaje oeste (como se muestra en la figura 3) y para cumplir las misiones se cuenta con grupo de personajes con habilidades propias de su especialidad (Entertainment 2001).

Figura 2. Captura de pantalla del videojuego **Desperados*.***

Robin Hood:The Legend of Sherwood*:* es un videojuego de infiltración, desarrollado por la misma compañía en 2002. Es muy similar al anterior, el jugador controla un máximo de cinco personajes, en un entorno basado en las historias del protagonista Robin Hood (Entertainment 2002). En la figura 4 se puede observar una captura de pantalla de este videojuego.

[1] Compañía alemana de desarrollo de videojuegos.

***Figura 3. Captura de pantalla del videojuego* Robin Hood:The Legend of Sherwood.**

En la siguiente tabla se describen las mecánicas recurrentes de estos videojuegos, tomando como guía las definidas en el epígrafe anterior. Para indicar que las mecánicas son similares a las de Commandos se pondrá *"Si"*, en caso contrario *"No"* y si varía algún mecanismo o incorporan otros diferentes serán descritos.

Tabla 1. Descripción de las mecánicas de los videojuegos Desperados y Robin Hood.

	Videojuegos	
Mecánicas	Desperados	Robin Hood:The Legend of Sherwood
Cámara	Si El enfoque de la cámara también se puede manejar con la rueda del *mouse* o mediante una opción en la interfaz.	Si El enfoque de la cámara también se puede manejar con la rueda del *mouse* o mediante una opción en la interfaz.
Interfaz del juego	Si (varía solo la distribución de las áreas en la pantalla).	No Presenta solo un área principal en la que se encuentran también los retratos de los PC, pero con la particular característica de que las acciones de cada uno están justamente debajo

		de cada retrato y estas son visibles solo cuando el PC se seleccione, como se aprecia en la figura 4.
Selección	Si (varían las teclas). Además se pueden seleccionar todos los PC con la tecla A y deseleccionarlos con la D.	Si
Locomoción	Si	Si
Acciones	Si	Si

Como se puede observar en la tabla anterior, las mecánicas en los dos videojuegos generalmente mantienen las mismas características que las de Commandos, salvo algunas que incluyen otros mecanismos adicionales para mejorar la jugabilidad. En Robin Hood, la interfaz del juego presenta un área principal para mostrar los retratos de los PC y sus respectivas acciones. Pero estas se siguen visualizando al seleccionar los PC, solo que de forma diferente a la que se estila usar en este subgénero. Estos videojuegos se diferencian mayormente en cuanto a historia, diseño y controles, sin embargo, sus mecánicas como tal son muy similares.

A partir del análisis de las mecánicas de estos videojuegos se identificaron los mecanismos que no deben faltar en cada una. A continuación, son descritos y agrupados en las mecánicas a las que pertenece.

Cámara: debe estar en una proyección isométrica, permitiendo enfocar gran parte del terreno, lo que es vital para ejercer las maniobras tácticas. El movimiento de esta se realiza de dos maneras, mediante las teclas de dirección del teclado y acercando el cursor a los bordes de la pantalla. Para enfocar la cámara se emplean las teclas “+” y “-” del área numérica del teclado, la rueda del *mouse* o puede ser mediante una opción en la interfaz del juego, estos tres mecanismos pudieran emplearse en el videojuego para enriquecerlo o solo los que se desee.

Interfaz del juego: la interfaz del juego debe estar conformada por dos áreas principales que no pueden faltar, una con retratos de los PC que actúan en la misión y otra para mostrar las acciones de los PC seleccionados. Esta primera debe indicar cuál o cuáles PC actualmente están seleccionados y debe permitir seleccionarlos haciendo clic sobre ellos, además debe mostrar una barra de salud o cualquier indicador que

represente el porciento de salud de cada uno. La segunda visualiza las acciones pertenecientes al PC seleccionado y la disposición de cada una. Si se encuentra seleccionado un solo PC se deben mostrar todas sus acciones y si son más de uno, solo las acciones que sean comunes para todos. La interfaz del juego también puede emplear disímiles funcionalidades u opciones extras a las definidas para mejorar la jugabilidad, pero estas ya serían propias del videojuego que se quiera desarrollar.

Selección: la selección de los PC para su posterior control, puede ser individual, donde se selecciona exclusivamente uno o múltiple, donde se puede seleccionar más de uno simultáneamente. La selección individual se puede realizar dando clic sobre un PC en la escena, sobre uno de los retratos o simplemente presionando la tecla asociada a cada uno. La selección múltiple puede realizarse de cinco maneras: mediante un cuadro de selección, manteniendo una tecla presionada y conjuntamente dando clic sobre los PC o sobre los retratos en la interfaz y la última haciendo uso de una tecla determinada para seleccionarlos a todos a la vez.

Locomoción: las formas de locomoción que pueden optar las unidades son: arrastrarse, caminar y correr. En el caso de los PC, para indicarles que caminen hacia un destino, se debe dar un clic en el lugar deseado y estos caminarán hacia allá, de este mismo modo para que corran se indicará con doble clic y si estaban caminando o detenidos deben comenzar a correr. Para arrastrarse primero se les ordena que se tiendan en el suelo mediante una opción en la interfaz o una tecla definida y luego de esto, puede empezar a arrastrarse hacia una posición especificada con un clic. Si el o los PC seleccionados están arrastrándose o tendidos, pueden pasar a correr si se les indica. En el caso de que se desee que caminen es necesario indicarles primero que se levanten.

Acciones: cada PC debe poseer un conjunto de acciones que facilitan el cumplimiento de los objetivos de la misión. Al seleccionar un personaje, sus acciones se deben mostrar en la interfaz del juego junto a la disponibilidad en cuanto a recursos de cada una. Si solo está seleccionado un PC, se mostrarán todas sus acciones, pero si hay más de un PC seleccionado, solo se mostrarán las que sean comunes. La activación de las mismas se realiza mediante los íconos en la interfaz o con las teclas correspondientes y para cancelar la acción en uso se emplea el clic derecho.

Luego de definir los elementos que deben poseer las mecánicas establecidas en el epígrafe, se analizará el proceso de desarrollo de videojuegos, haciendo énfasis en la trascendencia de las mecánicas en el ciclo de vida de estos y así verificar, en qué parte aportaría la solución.

1.5 Proceso de desarrollo de videojuegos

Los videojuegos son considerados como un tipo más de *software*, sin embargo para su diseño y desarrollo no existe una metodología estándar que garantice la calidad (Díaz 2015). La realización de un videojuego no es una tarea simple y requiere de mecanismos y metodologías propias según sea el tipo de videojuego y la complejidad que presente. Generalmente las propias compañías son las que establecen cuál será su filosofía de trabajo a lo largo de su creación.

Diversos autores como Díaz (Díaz 2015) y Pereira (Pereira 2014), concuerdan en que los videojuegos, a lo largo de su ciclo de vida se pueden segmentar en tres fases rectoras: Pre-Producción, Producción y Post-producción. A continuación, se describirá brevemente en qué consiste cada una según el criterio de los autores antes mencionados y se hará énfasis principalmente en cómo se lleva a cabo el desarrollo de mecánicas en las distintas fases.

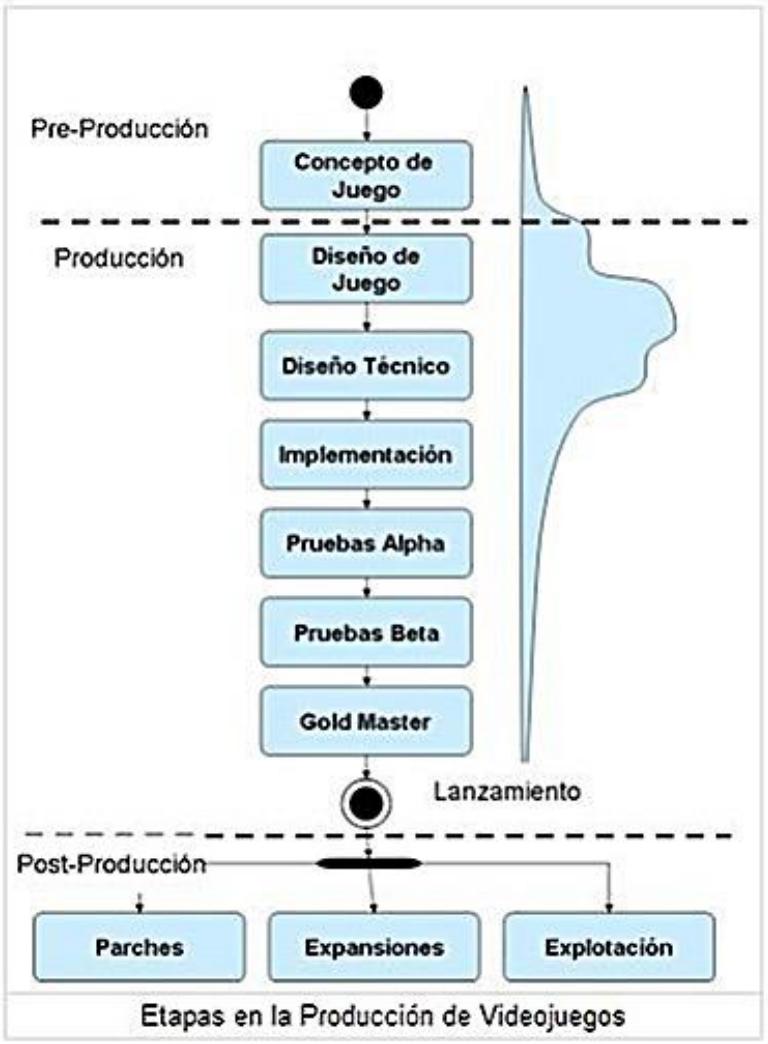


Figura 4. Etapas en la producción de videojuegos. Tomado de (Pereira, 2014).

Fase de Pre-Producción:

En esta fase inicial se define la concepción general del videojuego y los términos en los que se llevará a cabo su materialización. Se puntualizan diferentes aspectos que conforman el videojuego como: el género, la historia, se crean bocetos preliminares y se define el *gameplay* (Díaz 2015). El *gameplay* es un concepto amplio y difuso, que se define de manera diferente en cada videojuego. Se podría definir como la esencia, interactividad, grado o naturaleza del videojuego. Es aquí donde se define cómo se va a jugar, qué acciones se pueden realizar y cómo va a reaccionar el entorno del juego a las acciones del jugador a través del personaje. A su vez se establece cómo será la curva de aprendizaje del jugador. Todo esto sin entrar en

detalles gráficos, sonoros o de historia (Díaz 2015). Esta fase concluye con una primera versión del Documento de Diseño del Juego (*Game Design Document*) a manos del equipo creativo y que será fundamental para el diseño del videojuego y la producción en sí misma (Pereira 2014).

Fase de Producción:

Es la fase del proceso que más tiempo demanda y está compuesta por nueve sub-etapas: Diseño del Juego, Diseño Artístico (que engloba, la historia, sonidos, interfaces y gráficos) , Diseño Mecánico, Motor del Juego, Diseño Técnico, Implementación, Pruebas Alpha, Pruebas Beta y Gold Master (Díaz 2015). Donde se invierten los mayores esfuerzos y donde más personas colaborarán es en las dos primeras, pues al equipo de diseño inicial se le incorporará el resto de la plantilla asociada a la producción del videojuego. Concluidas todas las sub-etapas que componen la etapa de producción se tiene una primera versión del producto final (Díaz 2015).

Fase de Post-Producción:

El proceso vital del videojuego no termina con su puesta en el mercado. La etapa de post-producción genera un reporte cuyo propósito es describir en detalle todas las actividades que se realizaron satisfactoriamente y las que llegaron a causar problemas durante todo el proceso de desarrollo; junto con sugerencias para solucionar dichos problemas y no repetirlos en proyectos futuros (Díaz 2015).

Tras el análisis de las fases de este proceso, se identificó que las mecánicas de videojuegos están presentes en la fase de Pre-producción y Producción. En la primera se evidencia en la concepción del *gameplay.* Por otro lado, en la fase de Producción, el diseño de mecánicas marca las pautas de interacción con el videojuego, las normas internas y el tipo de comunicación que debe darse en caso de que el destino sea un entorno *on-line*[2]. En este punto los detalles cobran vida con el diseño de reacciones y comportamientos que darán funcionalidad a cada uno de los elementos y personajes del juego. Se comienza a diseñar la IA y los distintos mecanismos físicos de los elementos y del mundo donde se lleva a cabo (Pereira 2014).

La solución debe brindar las principales mecánicas presentes en videojuegos de Estrategia Táctica, para ser reutilizadas en la creación de cualquier videojuego de este tipo. Por tal razón, se evitaría la conceptualización de estas en la fase de Pre-producción y su implementación en la fase de Producción, lo

[2] Hace referencia a un estado de conectividad en una red de computadoras.

que contribuiría al proceso de desarrollo de videojuegos de este tipo en función a la reusabilidad, flexibilidad y extensibilidad.

Ya conocidas las mecánicas a desarrollar y el impacto que tienen en el proceso de desarrollo de videojuegos, es importante describir el motor de videojuegos (Unity3D) utilizado en el centro Vertex y sus potencialidades para la reutilización de mecánicas.

1.6 Unity 3D

Un motor de videojuego es una herramienta con funcionalidades reutilizables (implementadas previamente), que permiten automatizar determinadas tareas en el desarrollo de videojuegos. De este modo se evita lidiar con procesos de bajo nivel de gran complejidad. Este provee al videojuego de un motor para renderizar gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, *scripting*, animación, IA, redes, *streaming*, administración de memoria y un escenario gráfico (Fernández y Angelina 2012).

Para el desarrollo de la solución se usará el motor de videojuego Unity 3D en su versión 5.2. Está disponible para sistemas operativos, Windows y Macintosh y viene empaquetado como una herramienta para crear videojuegos, aplicaciones interactivas, visualizaciones y animaciones en 2D y 3D en tiempo real. Implementa contenido para múltiples plataformas como Windows, Mac OS, Linux, Xbox 360, Xbox One, PlayStation 3, PlayStation 4, PlayStation Vita, Wii, Wii U, Nintendo 3DS, iOs, Android y Windows Phone, además de contar con un *plugin web* para desarrollar videojuegos para navegador (Technologies 2011, 2015).

En Unity, las mecánicas de videojuegos son implementadas mediante *scripts* y compiladas usando una versión de *JavaScript* o *C#*. Además, está enfocado en los bloques de construcción (*assets*) y facilita su compresión y manipulación en tiempo de edición, para mejorar el rendimiento del videojuego. Los *assets* pueden ser: texturas, modelos 3D, archivos de audio, prefabricados (*prefabs)*, materiales, animaciones, *scripts* y cualquier contenido utilizado en la creación del videojuego. Permite empaquetar y exportar en un archivo con extensión propia del motor, colecciones de *assets* que conjuntamente conforman mecánicas, componentes o recursos, que se pueden importar en futuros proyectos de manera flexible (Technologies 2015). Esto potencia la reutilización, obteniéndose como consecuencia la aceleración y optimización del tiempo de desarrollo y minimiza el esfuerzo necesario para su realización.

Unity ofrece un editor visual compuesto por varias áreas para la construcción del contenido del juego. A continuación, se describe cada una de ellas:

1.6.1 Editor visual de Unity 3D

Está conformado por cinco áreas principales como se observa en la siguiente figura (Technologies 2015):

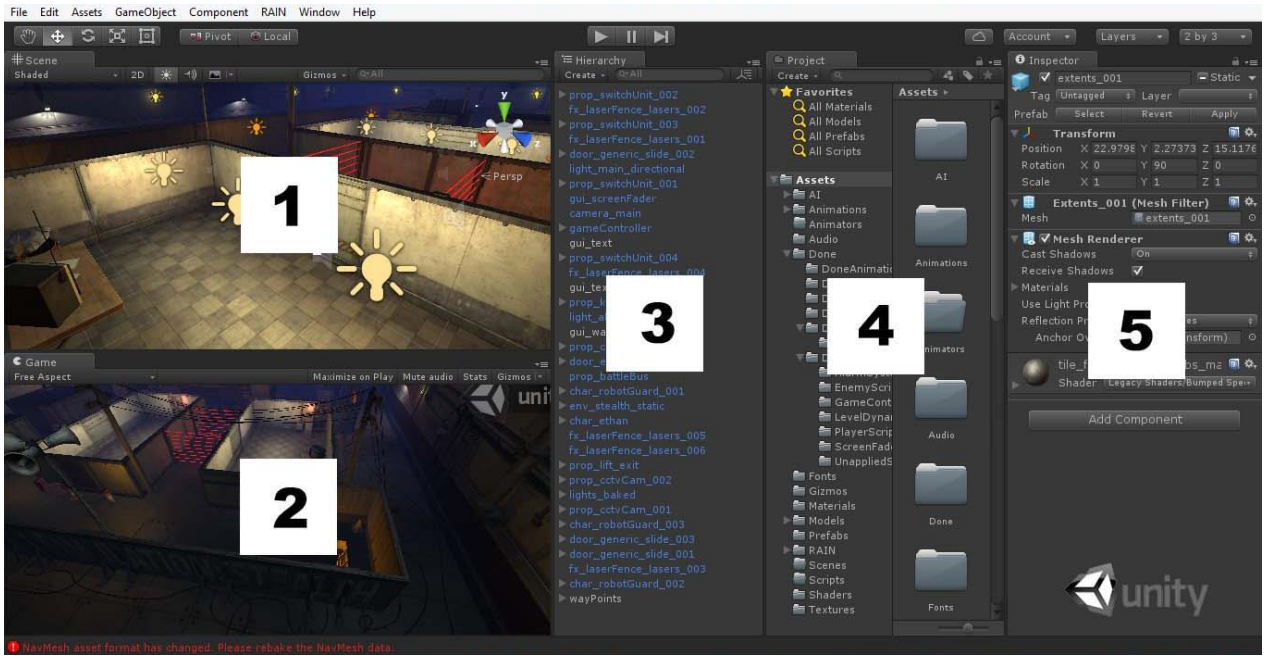


Figura 5. Captura de pantalla de la interfaz del editor de Unity 3D.

No 1. Ventana de escena: es el área de construcción de Unity donde se confecciona visualmente cada escena del juego.

No 2. Ventana de juego: en esta ventana se obtiene una previsualización del juego. Permite reproducirlo en cualquier momento del desarrollo para ir chequeando que funcione correctamente.

No 3. Ventana de jerarquía: esta ventana visualiza todos los objetos presentes en la escena actual.

No 4. Ventana de proyecto: permite manipular y visualizar el conjunto de *assets* que se emplea en el juego. En ella se pueden importar objetos 3D de distintas aplicaciones, texturas, audios, *scripts*, *prefabs,* escenas y animaciones, para usarlos en el videojuego.

No 5. Ventana de inspector: la ventana de inspector tiene varias utilidades, si se selecciona un objeto, entonces mostrará sus propiedades y podrán ser personalizadas. También permite configurar ciertas herramientas como la de terrenos si se tiene un terreno de Unity seleccionado.

Otra de las potencialidades que ofrece este motor es el sistema de mallas de navegación implementado, útil para trasladar unidades del juego hasta un destino determinado por el camino más óptimo y evadiendo obstáculos.

1.6.2 Mallas de navegación de Unity 3D

Una malla de la navegación (también conocida como *Navmesh*) es una representación simplificada de parte de la geometría del escenario del videojuego que los agentes (unidades del juego) usan para navegar por el entorno. Este proceso se conoce con el nombre de búsqueda de caminos (*pathfinding*). La generación de *Navmesh* en Unity es realizada por los diseñadores dentro de su editor, mientras el *pathfinding* lo llevan a cabo los agentes cuando el videojuego esta en tiempo de ejecución, basado en ese *Navmesh* previamente definido y para ello se emplea un algoritmo *A** (Technologies 2015).

En el anexo 7 se muestra un ejemplo de un escenario sencillo realizado en Unity 3D con malla de navegación generada. El cilindro representa a un agente que se puede desplazar única y exclusivamente por el área azul que sería la malla.

En conjunto con las mallas de navegación es necesario utilizar animaciones para simular las formas de locomoción que el personaje ejecute. Para el manejo y secuenciación de estas animaciones, Unity emplea máquinas de estado que son configuradas también de manera visual y se describen a continuación.

1.6.3 Control de animaciones en Unity 3D

Cada objeto en el transcurso del juego puede adoptar diferentes estados, por ejemplo, un personaje puede estar en estado inactivo, ejecutando la animación correspondiente a dicho estado y el usuario le orienta que debe caminar, este pasa al estado caminar y ejecuta su animación. Para controlar las animaciones y realizar transiciones al pasar de un estado a otro, Unity emplea las máquinas de estado antes mencionadas, lo que podría ser considerado como un tipo de diagrama de flujo o un sencillo programa escrito en un lenguaje de programación visual dentro del motor, ver figura 8. Donde los nodos representan los estados y los arcos (flechas) entre los nodos representan las transiciones. Estas máquinas de estado se guardan en *assets*

llamados *Animator Controller* y son editados en la ventana *Animator* de Unity 3D que se muestra a continuación.

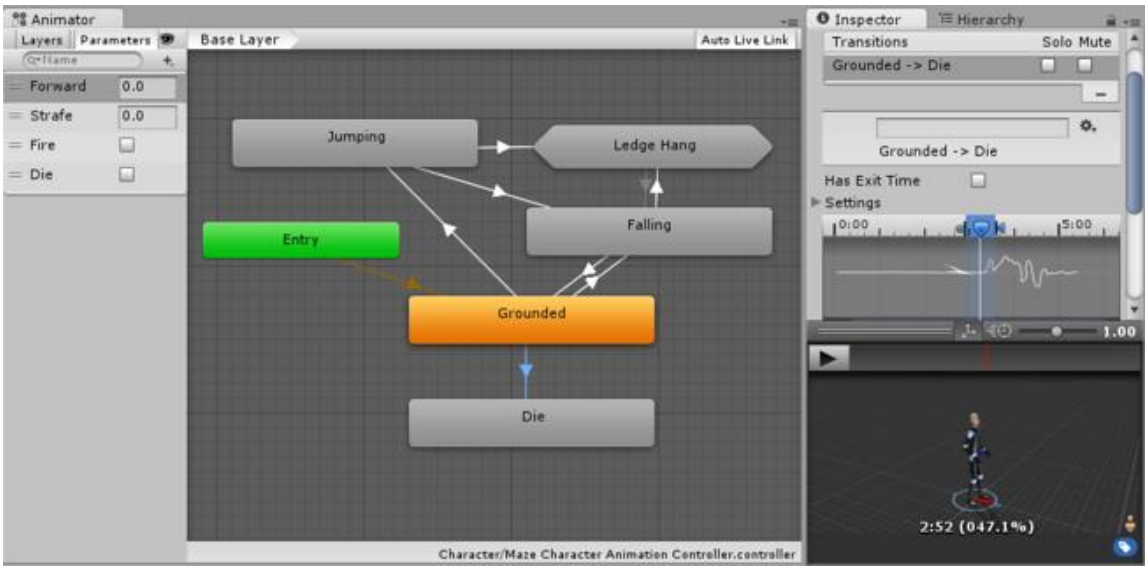


Figura 6. Típica vista de una máquina de estado de animaciones en la ventana Animator de Unity3D.

Para el motor Unity 3D existen múltiples paquetes compuestos por recursos. Algunos podrían servir para dar solución a varias mecánicas o incluso con ellos se podrían implementar videojuegos completos. A continuación, se describen tres paquetes que pudieran apoyar el desarrollo de la investigación.

1.7 Trabajos relacionados

El desarrollo de un videojuego es un proceso que puede tardar por disímiles factores y en muchas ocasiones existe la posibilidad de optimizar este tiempo. Por ejemplo, no siempre es necesario implementar desde un inicio todas las mecánicas que conforman al videojuego. Se pueden reutilizar recursos y funcionalidades predefinidas, que contribuyen a la disminución del esfuerzo. Durante la investigación se identificaron los siguientes paquetes por su similitud con el contexto de la investigación.

RTS Development Framework: contiene un conjunto de funcionalidades genéricas que posibilitan la creación de videojuegos RTS. Dentro de las mismas se destacan: cámara de Estrategia en Tiempo Real, mini-mapa, selección y mando de unidades (movimiento, ataque, parada), agrupación de unidades, construcción de estructuras y unidades con cola, pausa, cancelación y barras de vida, validación y rotación al colocar estructuras, recolección de recursos y sistema de interfaz de grafica de usuario. Además cuenta

con un código fuente completo en C# con extensos comentarios, dos escenas con ejemplos y un conjunto de prefabricados listos para usar (Unity Technologies 2016).

Fury Framework - RPG/RTS Creator: permite el desarrollo rápido de prototipos de videojuegos de rol (RPG, Role Playing Game, por sus siglas en inglés) y de Estrategia en Tiempo Real. Posee un conjunto de funcionalidades para la creación de habilidades personalizadas, campos de batalla multijugador, torres de defensa, artículos, IA para los NPC e inventarios. La documentación de todos los métodos y variables está implementado en C# (Technologies 2016a).

Generic Strategy Framework: brinda funcionalidades para crear videojuegos de estrategia por turnos y en tiempo real sobre la base de una rejilla de hexágonos de 1x1 hasta 100x100. Dispone de niebla de guerra para aplicar al terreno, de algoritmos para bordear obstáculos y encontrar el camino más corto, cámara de Estrategia en Tiempo Real, entre otros recursos (Technologies 2016b).

Con la búsqueda realizada se hallaron solo estos paquetes, concebidos generalmente para crear videojuegos RTS y RPG. Por lo tanto, no poseen la especificación necesaria para el tipo de videojuego referente en la investigación. Algunas de las mecánicas que presentan podrían ser de utilidad para la solución de este trabajo, sin embargo, su principal limitante es que para obtenerlos es necesario comprarlos.

Una vez demostrado de que no es posible emplear las soluciones encontradas, solo resta caracterizar la metodología, herramientas y lenguajes que guiarán el trabajo.

1.8 Descripción del marco de trabajo

Para el desarrollo de la solución, el cliente determinó que se utilizaría como metodología de desarrollo de software Extreme Programming (XP) y como Herramientas para Ingeniería de Software Asistida por Computadora (herramienta CASE), Visual Paradigm, por la experiencia de trabajo y los buenos resultados que se han obtenido en el centro con el uso de estos. También estableció como lenguaje de programación C#, debido a que es el empleado para desarrollar videojuegos con Unity en dicho centro, así como el Entorno de Desarrollo Integrado (IDE) Monodevelop. A continuación, se describen sus características distintivas.

Metodología XP

Las metodologías son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de software; están divididas en dos grupos, metodologías tradicionales o robustas y metodologías

ágiles o ligeras. La diferencia que existe entre ambos grupos es que las tradicionales son procesos muchos más controlados, con políticas y normas, tratando de buscar la calidad del software a través del orden y la documentación, sin embargo, las ágiles son procesos menos controlados y con pocos principios, que tratan de buscar la calidad del software a partir de la comunicación inmediata y directa entre las personas que intervienen en el proceso de desarrollo (Awad 2005).

XP es una de las metodologías ágiles que se diferencia de las metodologías tradicionales, principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Se basa en la simplicidad, la comunicación y el reciclado continuo de código y valora (Joskowicz 2008):

- Más a los individuos y las interacciones que a los procesos y las herramientas.
- A las aplicaciones que funcionan sobre la documentación exhaustiva.
- La colaboración del cliente sobre las negociaciones contractuales.
- La respuesta al cambio sobre el seguimiento de un plan.

Las características fundamentales de XP son (Pressman 2010):

- Pocos artefactos.
- Pocos Roles.
- Aplicable en grupos pequeños, menos de 10 integrantes y trabajando en el mismo sitio.
- Ideado para proyectos de corta duración.
- Desarrollo iterativo e incremental.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas.
- Programación en parejas: permite una mayor calidad del código escrito de esta manera es revisado y discutido mientras se escribe.
- El cliente forma parte del equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad.
- Es flexible a los cambios que puedan ocurrir en las funcionalidades del sistema.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar la legibilidad y el mantenimiento del mismo.
- Propiedad del código compartida.
- Simplicidad en el código.
- Entregas pequeñas del software a medida que se elabora.

Lenguaje de Modelado Unificado (UML, Unified Modeling Language, por sus siglas en inglés)

Se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema notacional destinado a los sistemas de modelado que utilizan conceptos orientados a objetos. Es el estándar mundial que utilizan los desarrolladores, autores y proveedores de Herramientas para Ingeniería de Software Asistida por Computación (CASE, *Computer Aided Software Engineering*, por sus siglas en inglés) (Group 2016).

Herramienta CASE

Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como: el proceso de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores (Pressman 2010).

Visual Paradigm

Es una herramienta que ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos (Pressman 2010). Dicha herramienta mediante la utilización de UML colabora con el desarrollo de la aplicación, desde la planificación, el análisis, el diseño y la generación de la documentación.

Monodevelop

Un IDE es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o utilizarse para varios. MonoDevelop es un IDE con su plataforma libre y gratuito, diseñado primordialmente para C# y otros lenguajes de programación. Incluye manejo de clases, ayuda incorporada, completamiento de código, soporte para proyectos y un depurador integrado desde la versión 2.2 (Larrateguy, Pividori y Sandrigo 2008). El motor de videojuego Unity 3D lo incorpora como herramienta en su instalación.

C#

Es uno de los lenguajes de programación que emplea Unity 3D para compilar los *scripts*, es orientado a objetos, lo cual facilita el trabajo ya que en esta herramienta todo componente o elemento del juego es un objeto o clase. Al empezar a programar, se pueden definir una o más clases dentro de un mismo espacio de nombres. Presenta un rango más amplio y definido de tipos de datos que otros lenguajes. Soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo (Seco 2001).

Consideraciones parciales

En el desarrollo de este capítulo se obtuvo un mejor dimensionamiento del problema a partir del estudio de los principales conceptos asociados a la solución. Con el análisis de los tres videojuegos pioneros en el subgénero Estrategia Táctica, se evidencia que las mecánicas principales (Cámara, Locomoción, Selección, Interfaz del juego y Acciones), poseen mecanismos muy semejantes, por lo que se agrupan para brindar más posibilidad de juego a los usuarios. Al estudiar los paquetes existentes para realizar videojuegos de estrategia con Unity 3D, no se encontró ninguno para videojuegos de Estrategia Táctica, por lo que sería de gran utilidad conformar un paquete para este tipo de videojuegos específicamente, que presente sus propias mecánicas y su propia arquitectura, con lo que se evitaría el tiempo de asimilación y uso de varios de estos.

CAPÍTULO 2. SOLUCIÓN PROPUESTA

Introducción

En el presente capítulo se describen las principales características del sistema propuesto, donde se ejecutan las fases de Exploración y Planificación definidas en la metodología de desarrollo de software XP, así como los diferentes artefactos generados en cada una de ellas. Se definen los requisitos no funcionales, se presenta la propuesta de solución del sistema y las historias de usuario expresando las necesidades de este. Finalmente se realiza una estimación de tiempo, se define la arquitectura del sistema y el diseño de clases.

2.1 Requisitos no funcionales del sistema

Los requisitos no funcionales según Sommerville (Sommerville 2005), *"son los que actúan para obligar la solución y se conocen a veces como apremios o requisitos de calidad"*. Se refieren a todos los requerimientos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento que debe poseer el sistema (Sommerville 2005).

Implementación

RNF1. Para el desarrollo del sistema debe emplearse como motor de videojuego Unity 3D.

RNF2. Como lenguaje de programación se empleará C#.

Usabilidad

RNF3. El sistema debe ser manipulado por personas con conocimiento previo en el manejo de la herramienta Unity 3D.

Software

RNF4. El sistema debe ser compatible con Unity 3D de la versión 5.2 en adelante.

Extensibilidad

RNF5. El sistema debe permitir incorporar nuevas funcionalidades para facilitar el crecimiento del mismo en el futuro.

2.2 Propuesta de solución

Como solución al problema de la investigación, se desarrollará un paquete para videojuegos de Estrategia Táctica en tiempo real (TRTS, Tactical Real Time Strategy, por sus siglas en inglés) para Unity 3D que

contendrá implementados los mecanismos de las mecánicas bases para el desarrollo de videojuegos de tipo Estrategia Táctica definidas con anterioridad (Cámara, Selección, Locomoción, Acciones e Interfaz del juego), ver figura 9. La solución debe permitir integrarlas en nuevos proyectos y configurar los parámetros de cada una según las necesidades del juego.

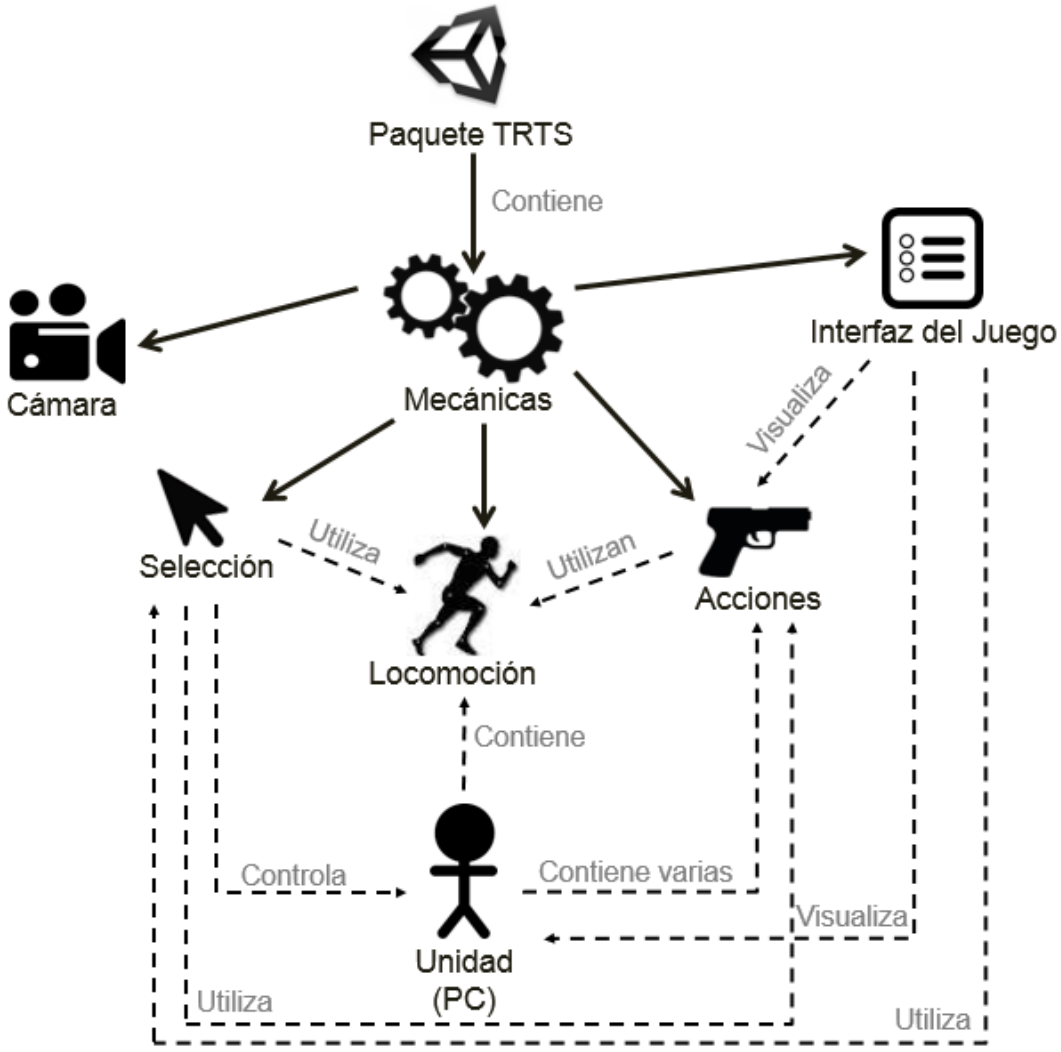


Figura 7. Estructura del paquete propuesto.

- **Cámara:** le brindará al usuario una cámara funcional propia de este subgénero, que visualizará cada elemento del escenario del juego y posee parámetros configurables como la velocidad de movimiento, la velocidad para enfocar y el máximo y mínimo factor de enfoque. El usuario también

podrá configurar si desea mantener la cámara siempre a una altura determinada respecto al terreno, cuál será esa altura máxima y la velocidad de la transición al ajustarse.

- **Selección:** como se puede observar en la figura anterior esta mecánica se encargará del control de PC en la escena, mediante la mecánica de locomoción que estos contienen para indicarles un destino y de las mecánicas de acciones para orientarles que ejecuten una acción determinada. Además, le brindará al usuario un conjunto de modos de selección que podrá configurar, el modo por defecto que debe tener es el de selección simple de unidades y los demás podrán usarse o no en el videojuego. Para la selección con cuadro el usuario podrá elegir un determinado cuadro de selección y para la selección puntual y la selección de todas las unidades podrá definir una tecla respectivamente.
- **Locomoción:** le garantizará al usuario la locomoción y traslación de unidades, le brindará diferentes formas de locomoción como: tenderse, arrastrarse, caminar y correr, las cuales podrá usar si desea, podrá definir la velocidad de traslación de cada una y sus respectivas animaciones.
- **Acciones:** las mecánicas de acciones utilizarán la locomoción para conducir a los PC hacia un objetivo. Esta le permitirá al usuario adicionar acciones a las unidades e implementarles el comportamiento de cada una de estas acciones haciendo uso de las formas de locomoción que brindará la mecánica Locomoción. Cada acción deberá contener parámetros de configuración tales como: el nombre, el ícono, el tipo de acción y los cursores que representarán la activación de esta.
- **Interfaz del juego:** deberá mostrar un área donde visualice los retratos de los PC seleccionados y otra con espacios o capacidades para colocar sus acciones. Estas áreas deberán permitir ser acomodadas en la pantalla según las preferencias del usuario y adicionar nuevas capacidades para retratos y acciones. Cada retrato tendrá asociado un PC, la imagen de este, la tecla con que se seleccionará y la barra de salud. Por otro lado, las capacidades de acciones tendrán asociadas una tecla para activarlas.

Tras el análisis de esta propuesta se pudo identificar qué parte de las mecánicas definidas se pueden reutilizar para la creación de NPC. Excluir las mecánicas de Interfaz del juego, Selección y sustituirlas por *scripts* adicionales con IA que empleen solamente de las mecánicas de Locomoción y Acciones, permitirían la asignación de sus comportamientos. Ver en la siguiente figura:

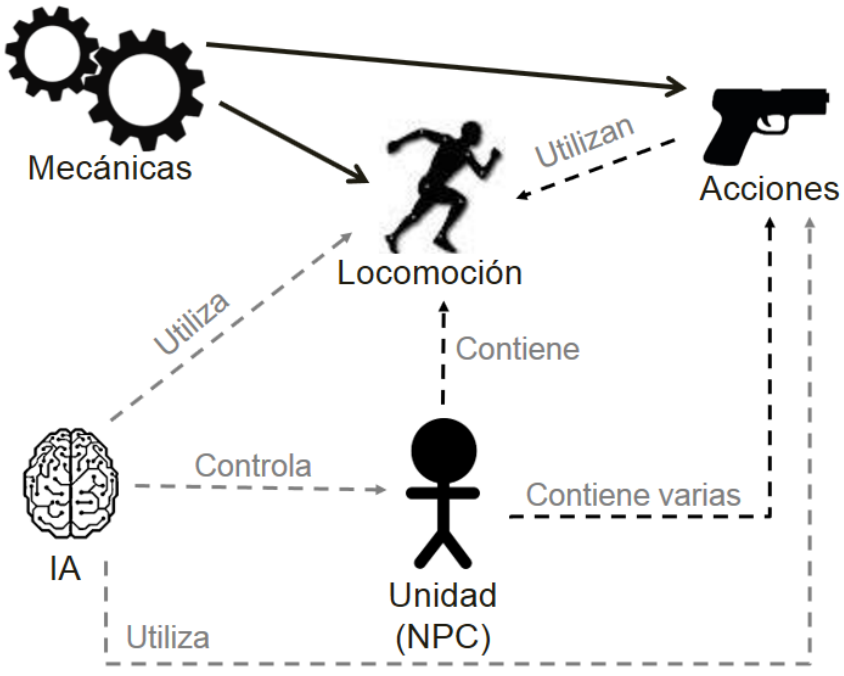


Figura 8. Reutilización de mecánicas para la creación de NPC.

2.3 Fase de exploración

La exploración es la etapa del proceso de desarrollo de software que propone XP para comenzar la construcción de un producto. Cuando sean entregadas las propuestas del cliente al equipo de trabajo, comienza el análisis, la tormenta de ideas y la conceptualización del software. Un aspecto de gran importancia es que el cliente debe estar inmerso en cada uno de las actividades antes expuestas (Pressman 2010). En esta etapa se realizará un estudio de las funcionalidades que se deben desarrollar para lograr primeramente el objetivo principal de la aplicación, así como las que se complementarían para lograr el éxito del proyecto.

2.3.1 Historias de usuario

Las Historias de usuario (HU) son escritas por el cliente, en un lenguaje no técnico, como descripciones cortas de lo que el sistema debe realizar. Deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo (Joskowicz 2008). Las siguientes tablas muestran algunas de las Historias de usuario que describen cómo debe funcionar el paquete de Unity 3D a realizar, el resto se pueden encontrar en el anexo 1.

Tabla 2. HU Insertar y configurar mecánica de cámara

Historia de usuario	
No 1: Insertar y configurar mecánica de cámara	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 7 días	**Iteración asignada:** 1
Descripción: El usuario tendrá la posibilidad de tomar el prefabricado de la cámara de Estrategia Táctica y arrastrarlo hacia la escena. Luego brindarle la opción de configurar los parámetros de movimiento y visualización en el inspector de Unity 3D. Como resultado se obtiene una cámara especializada para videojuegos de tipo Estrategia Táctica.	
Observaciones:	

Tabla 3. HU Seleccionar con cuadro de selección varios personajes

Historia de usuario	
No 3: Seleccionar con cuadro de selección varios personajes	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario podrá seleccionar varios personajes presentes en la escena utilizando un cuadro de selección.	
Observaciones: El cuadro de selección debe ser un prefabricado previamente creado y debe estar definido en la HU 2.	

Tabla 4. HU Insertar y configurar mecánica de locomoción

Historia de usuario	
No 10: Insertar y configurar mecánica de locomoción	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 5 días	**Iteración asignada:** 2
Descripción: El usuario debe tener la posibilidad de adicionarle a los modelos de los personajes un *script* que les proporcionará la locomoción y poderle configurar los parámetros de velocidad de cada forma de locomoción, así como definir cuál o cuáles de estas empleará.	

2.4 Fase de planificación

Durante esta fase se realiza una estimación del esfuerzo que costará implementar todas las historias de usuario, se parte del tiempo asignado a cada una en la fase de exploración. Una vez terminado, se procede a organizarlas en las iteraciones correspondientes, teniendo en cuenta la prioridad especificada por el cliente y el tiempo de desarrollo de cada una (Rondón 2012).

2.4.1 Plan de iteraciones

Una iteración es una parte esencial del proyecto que cuando culmina el cliente obtiene un resultado parcial. Al concluir la última iteración, el mismo quedará completamente satisfecho, pues culmina la producción del proyecto. En la organización de las iteraciones se debe evitar extender más de un mes laboral; por lo general se proponen entre 20 y 21 días. La demora de un producto puede atentar con los objetivos que tenga el cliente con la aplicación. Una manera de evitar que esto ocurra se lleva a cabo mediante las pruebas de los usuarios finales, las correcciones a las que son sometidas la aplicación en las revisiones y así se podrá definir si el software va por un buen camino (Rondón 2012). En la siguiente tabla se pueden observar las iteraciones que se determinaron necesarias para el desarrollo de la solución.

Tabla 5. Distribución de iteraciones por Historias de usuarios

Iteraciones	Historias de usuario a implementar	Tiempo de trabajo
Iteración 1	Insertar y configurar mecánica de cámara. Insertar y configurar mecánica de selección. Seleccionar con cuadro de selección varios personajes. Seleccionar un personaje con el clic izquierdo del *mouse.* Realizar selección puntual de personajes. Realizar selección total de personajes	25 días
Iteración 2	Insertar y configurar *script* de unidades. Visualizar indicadores de selección al seleccionar una unidad. Insertar y configurar mecánica de locomoción. Configurar máquina de estado de locomoción.	20 días
Iteración 3	Indicar al personaje que ejecute determinada forma de locomoción.	21 días

	Insertar y configurar mecánica de acción.	
Iteración 4	Brindar al usuario una base para la implementación de las acciones. Insertar y posicionar la interfaz del juego en la pantalla según el gusto del usuario. Insertar y eliminar capacidades para retratos de personajes y acciones.	18 días
Iteración 5	Configurar capacidades para retratos de personajes. Configurar capacidades de acciones.	15 días

2.4.2 Plan de entregas

El plan de entrega es el compromiso final del equipo de desarrollo con el cliente, pues en él se define cuando será entregado el producto. Representa un factor importante para el proyecto, pues la demora en la entrega del proyecto trae consigo insatisfacción con el cliente. En el plan de entrega se lleva a cabo la estimación del tiempo necesario para entregar al cliente las versiones del producto a medida que se cumplan los requisitos del software (Pressman 2010). La siguiente tabla muestra las versiones del producto que deben entregarse al final de cada iteración. Estas entregas se realizarán en la cuarta semana de los meses de enero, febrero, marzo, abril y mayo respectivamente.

Tabla 6. Plan de entregas del producto

Producto	1ra Iteración	2da Iteración	3ra Iteración	4ta Iteración	5ta Iteración
Paquete de mecánicas para videojuegos de Estrategia Táctica	Versión 0.1 del producto	Versión 0.2 del producto	Versión 0.3 del producto	Versión 0.4 del producto	Versión 1.0 del producto

2.5 Fase de diseño

En la fase de diseño la metodología XP sugiere que el diseño del software sea sencillo y claro, ya que esto reduce el tiempo de trabajo y el esfuerzo. El diseño de la aplicación establece conformar un diseño sencillo de la arquitectura y de las tarjetas de clase, responsabilidad y colaboración (CRC) (Rondón 2012). A continuación, se describen las etapas trabajadas en esta fase.

2.5.1 Descripción de la arquitectura

La solución se desarrollará sobre el motor de videojuego Unity 3D, por lo cual debe ajustarse la arquitectura en capas que este presenta. En la figura siguiente se describe cada una de sus capas.

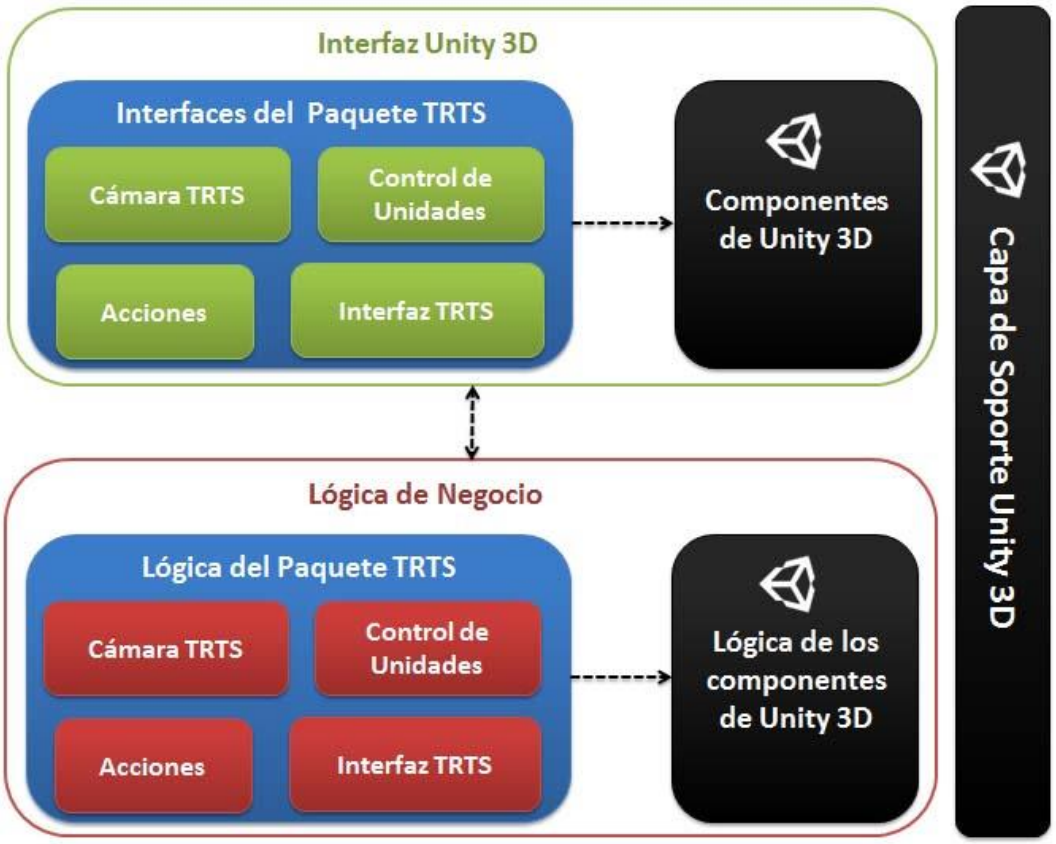


Figura 9. Diseño de la arquitectura.

Interfaz Unity 3D: es la capa encargada de visualizar en su interfaz los elementos del motor de Unity 3D, para que el usuario pueda interactuar con ellos. En esta se encontrará el paquete *"Interfaces del paquete TRTS"* el cual contendrá un conjunto de interfaces configurables propias de la solución, que serán visualizadas por esta capa. Este tendrá una dependencia de los componentes de Unity, representados en el paquete *"Componentes de Unity 3D".*

Lógica del Negocio: es la capa que contiene la lógica de cada componente de Unity 3D, en ella estará la lógica de cada mecanismo de la solución representados en el paquete *"Lógica del Paquete TRTS",* el cual posee una dependencia con el paquete *"Lógica de los componentes de Unity 3D".*

Capa de soporte Unity 3D: contiene las bibliotecas para la física, los gráficos, el sonido, y el almacenamiento de los datos, brindadas por el motor de videojuego Unity 3D. Esta capa se encuentra transversal a las otras, lo cual les posibilita el acceso para hacer uso de los recursos que brinda.

2.5.2 Patrones de diseño

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Los desarrolladores lo usan como una forma de reutilizar la experiencia, clasificando las soluciones con términos de común denominación y van formando un amplio repertorio de principios generales y de expresiones que los guían al crear un software (Moreira y Velázquez 2013). En el diseño de la solución se usarán algunos de los patrones generales de software para asignar responsabilidades (GRASP, *General Responsibility Asignment Software Patterns*, por sus siglas en inglés) y algunos de los GoF (*Gang of Four*).

A continuación, se describen los patrones GRASP que se usarán en el desarrollo de la solución.

Bajo acoplamiento: permitirá que el número de relaciones entre las clases de la solución sea el menor posible. De tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, lo cual potencia la reutilización y disminuye la dependencia entre estas (Palacios 2013).

Alta cohesión: la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo. En resumen, este patrón se observa cuando una clase tiene la responsabilidad de realizar una labor dentro del sistema, no desempeñada por el resto de los componentes del diseño. Este patrón en la solución se evidenciará en conjunto con el patrón bajo acoplamiento, de forma tal que cada clase realice sus acciones y se evite que otra clase realice acciones correspondientes a la clase con la que está relacionada (Palacios 2013).

Polimorfismo: este patrón se emplea cuando el comportamiento relacionado a una clase varía según el tipo. Para esto se utilizan operaciones polimórficas a los tipos para los que varía el comportamiento (Cartagena99 2011). En la solución se evidenciará en las acciones de los personajes, donde la lógica de cada una cambia respecto a su tipo, pero en esencia seguirá siendo una acción.

También se utilizarán para el diseño de la solución los patrones GoF. Estos se clasifican en tres grandes categorías (Vlissides et al. 1995):

Creacionales: los patrones creacionales tratan con las formas de crear instancias de objetos. Su objetivo es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. (Fábrica abstracta, Constructor, Método de fabricación, Prototipo, Instancia única). **Estructurales:** los patrones estructurales describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos. (Adaptador, Puente, Compuesto, Decorador, Fachada, Peso ligero).

Comportamiento: los patrones de comportamiento ayudan a definir la comunicación e iteración entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos. (Cadena de responsabilidad, Orden, Intérprete, Iterador, Mediador, Observador, Estado, Estrategia, Método plantilla, Visitante).

De los patrones GoF antes mencionados para la solución se empleará:

Observador: este patrón define una dependencia uno a muchos entre objetos de modo que cuando el estado de un objeto cambia, se les notifica el cambio a todos los que dependen de él y se actualizan de forma automática (Dodero y Llamas 2003). En la solución se usará para notificar a las clases que usen este patrón que deben actualizarse. Un ejemplo de su uso sería al seleccionar personajes en la escena se le debe informar a la interfaz del juego que debe actualizar el estado de los retratos de los personajes.

2.5.3 Diagrama de clases de la solución

Para una mayor comprensión de cómo estará estructurado el sistema de *scripts* que compone a la solución se define el siguiente diagrama de clases:

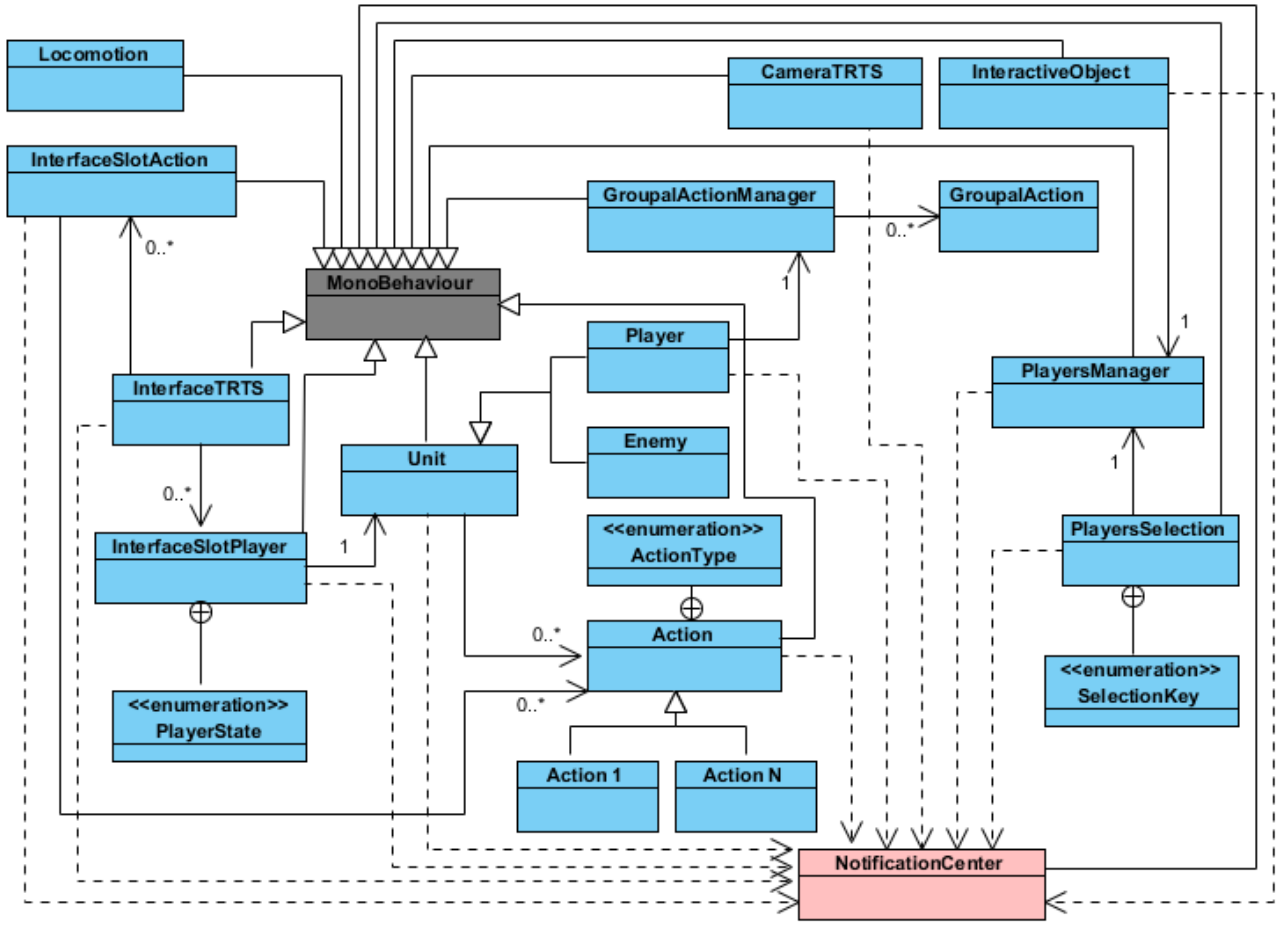


Figura 10. Diagrama de clases de la solución.

Las clases de color azul que se muestran en la figura son propias de la solución, la de color gris es la clase *MonoBehaviour* de Unity 3D de la cual hereda cada objeto o elemento del videojuego y la de color rosado es una clase externa ya implementada que se empleará para pasar mensajes de notificación de una clase a otra, es en esta donde se evidencia el patrón observador. En el próximo apartado se describen con más detalles las clases a implementar mediante las tarjetas CRC.

2.5.4 Descripción de las tarjetas CRC

El principal objetivo de las tarjetas CRC es dejar el pensamiento procedimental para enfocarse al orientado a objetos. Cada tarjeta representa una clase con su nombre en la parte superior, en la sección inferior izquierda están descritas las responsabilidades y a la derecha las clases que le sirven de soporte (Rondón 2012). A continuación, se presentan las tarjetas que fueron necesarias crear para el desarrollo de la solución.

Tabla 7. Tarjeta CRC PlayersManager

Tarjeta CRC	
Clase: PlayersManager	
Responsabilidades	**Clases relacionadas**
➢ Mostrar lista de jugadores seleccionados en el inspector de Unity 3D. ➢ Adicionar y eliminar PC a la lista de jugadores presentes en la escena. ➢ Activar indicadores de selección de los PC seleccionados. ➢ Desactivar todas las acciones de los PC presentes en la escena. ➢ Tender al suelo a los PC seleccionados. ➢ Ordenar a la cámara enfocar a un PC especificado.	➢ MonoBehaviour ➢ CameraTRTS ➢ PlayersSelection ➢ InteractiveObject ➢ NotificationCenter

Tabla 8. Tarjeta CRC CameraTRTS

Tarjeta CRC	
Clase: CameraTRTS	
Responsabilidades	**Clases relacionadas**
➢ Controlar el movimiento de la cámara. ➢ Controlar el *zoom* de la cámara. ➢ Suavizar la altura de la cámara según irregularidades del terreno. ➢ Activar, restablecer y cambiar el cursor por defecto del juego. ➢ Dirigir la cámara hacia la posición del personaje especificado.	➢ MonoBehaviour ➢ PlayersManager ➢ NotificationCenter

Tabla 9. Tarjeta CRC PlayersSelection

Tarjeta CRC	
Clase: PlayersSelection	
Responsabilidades	**Clases relacionadas**
➢ Realizar selección simple de PC. ➢ Realizar selección puntual de PC. ➢ Realizar selección con cuadro de PC.	➢ MonoBehaviour ➢ PlayersManager ➢ SelectionKey

➢ Pintar cuadro de selección en la pantalla del juego. ➢ Indicar el destino y la forma de locomoción a ejecutar el PC seleccionado.	➢ NotificationCenter

Tabla 10. Tarjeta CRC Unit

Tarjeta CRC	
Clase: Unit	
Responsabilidades	**Clases relacionadas**
➢ Manejar la salud de la unidad. ➢ Activar los indicadores de selección de la unidad.	➢ MonoBehaviour ➢ NotificationCenter ➢ Interface*PlayerSlot* ➢ Action

Tabla 11. Tarjeta CRC Player

Tarjeta CRC	
Clase: Player	
Responsabilidades	**Clases relacionadas**
➢ Completar las listas de acciones grupales e individuales. ➢ Verificar si está muerta la unidad. ➢ Activar una acción especificada. ➢ Verificar si está activada la acción especificada. ➢ Verificar si contiene la acción especificada. ➢ Adicionar nueva acción individual. ➢ Activar y desactivar el cursor de la unidad.	➢ Unit ➢ NotificationCenter

Tabla 12. Tarjeta CRC Enemy

Tarjeta CRC	
Clase: Enemy	
Responsabilidades	**Clases relacionadas**
➢ Verificar si está muerta la unidad.	➢ Unit

Tabla 13. Tarjeta CRC Locomotion

Tarjeta CRC	
Clase: Locomotion	
Responsabilidades	**Clases relacionadas**
➢ Dirigir a la unidad hasta un destino ejecutando la forma de locomoción: caminar, correr o arrastrarse. ➢ Tender al suelo a la unidad. ➢ Detener y restablecer el recorrido de la unidad. ➢ Verificar si es correcto el destino.	➢ MonoBehaviour

Tabla 14. Tarjeta CRC Action

Tarjeta CRC	
Clase: Action	
Responsabilidades	**Clases relacionadas**
➢ Activar y desactivar el cursor de la acción. ➢ Activar y desactivar la acción. ➢ Iniciar y finalizar la acción. ➢ Ejecutar la lógica de la acción.	➢ MonoBehaviour ➢ Unit ➢ Interface*ActionSlot* ➢ ActionType ➢ NotificationCenter

Tabla 15. Tarjeta CRC GroupalActionManager

Tarjeta CRC	
Clase: GroupalActionManager	
Responsabilidades	**Clases relacionadas**
➢ Buscar una acción grupal determinada por su nombre.	➢ MonoBehaviour ➢ GroupalAction

Tabla 16. Tarjeta CRC GroupalAction

Tarjeta CRC	
Clase: GroupalAction	
Responsabilidades	**Clases relacionadas**

➢ Almacenar los datos de la acción grupal.	➢ GroupalActionManager

Tabla 17. Tarjeta CRC InteractiveObject

Tarjeta CRC	
Clase: InteractiveObject	
Responsabilidades	**Clases relacionadas**
➢ Verificar la acción y los personajes que pueden interactuar con el objeto. ➢ Activar y desactivar el cursor de la acción que pueda interactuar con el objeto si está activa. ➢ Iniciar acción.	➢ MonoBehaviour ➢ PlayersManager ➢ NotificationCenter

Tabla 18. Tarjeta CRC InterfaceTRTS

Tarjeta CRC	
Clase: InterfaceTRTS	
Responsabilidades	**Clases relacionadas**
➢ Seleccionar PC. ➢ Actualizar interfaz de acciones.	➢ MonoBehaviour ➢ Interface*PlayerSlot* ➢ Interface*ActionSlot* ➢ NotificationCenter

Tabla 19. Tarjeta CRC InterfacePlayerSlot

Tarjeta CRC	
Clase: Interface*PlayerSlot*	
Responsabilidades	**Clases relacionadas**
➢ Mostrar la salud actual del PC asociado. ➢ Chequear muerte del PC asociado. ➢ Permitir seleccionar al PC asociado mediante una tecla y dando clic sobre el retrato de este en la interfaz. ➢ Cambiar el retrato del PC al seleccionarlo, deseleccionarlo o morir. ➢ Reproducir los audios que presenta el personaje.	➢ MonoBehaviour ➢ InterfaceTRTS ➢ Unit ➢ PlayerState ➢ NotificationCenter

Tabla 20. Tarjeta CRC InterfaceActionSlot

Tarjeta CRC	
Clase: Interface*ActionSlot*	
Responsabilidades	**Clases relacionadas**
➢ Activar y desactivar el *slot* o capacidad. ➢ Activar la acción asociada mediante una tecla y dando clic sobre esta en la interfaz. ➢ Activar y desactivar el cursor de la acción asociada.	➢ MonoBehaviour ➢ InterfaceTRTS ➢ Action ➢ NotificationCenter

2.5.5 Estándares de Codificación

Para el desarrollo de la solución se definen convenciones para la escritura del código fuente. Por ejemplo: el nombre de las variables, los métodos y clases, estilo de indentación y uso del idioma inglés. Todo esto respetando el estilo de codificación del lenguaje C#. El estándar de codificación propuesto se enfoca a la legibilidad del código ya que esto repercute directamente en la comprensión del software por parte de los programadores. El mantenimiento del sistema es más fácil y puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores o mejorar el rendimiento.

Definiciones generales

Las definiciones se realizan en inglés de manera de*script*iva, evitando las abreviaturas y los nombres cortos. Se utiliza este idioma debido a que las palabras son simples, no se acentúan y está muy difundido en el mundo informático.

Clases: las clases comienzan con mayúscula al inicio de la palabra y en caso de estar conformada por palabras compuestas, la definición debe ser continua y cada palabra debe iniciar con mayúscula siguiendo el estilo determinado.

Ejemplo:

public class PlayersManager

Declaración de variables: los nombres de las variables comienzan con minúscula y en caso de estar conformada por palabras compuestas, la definición debe ser continua y las demás palabras deben iniciar con mayúscula, como se muestra a continuación.

Ejemplo:

string name;
float walkSpeed;

De esta manera al visualizarse en la ventana de inspector de Unity 3D se separan por palabras y comienzan con mayúscula lo que ayuda a una mejor comprensión, véase en la siguiente figura.

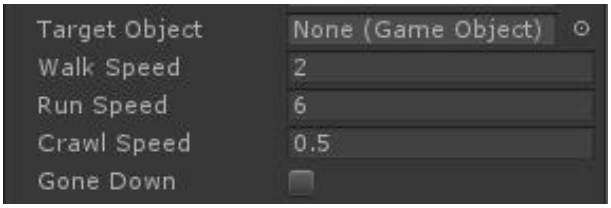


Figura 11. Variables visualizadas en la ventana de inspector de Unity 3D.

Métodos: en este caso, el nombre debe comenzar con mayúscula y en caso de estar conformada por palabras compuestas, la definición debe ser continua y cada palabra debe iniciar con mayúscula. Los constructores como lo exigen los compiladores, llevan el nombre de la clase.
Ejemplo:
Constructores:
Unit(string name, float live)

Métodos:
void Start()
bool SelectAllPlayers()

Estilo de indentación: el estilo utilizado es propio para lenguajes de programación que usan llaves para indentar o delimitar bloques lógicos de código y es también un punto clave para hacer el código más legible. Está presente en los ciclos y estructuras de control.
Ejemplo:

```
if(EnableAction()){
 // Instructions
}
```

Consideraciones parciales

Con la culminación del presente capítulo, se arribó a la conclusión de que las bases de la solución técnica del paquete para la creación de videojuegos de Estrategia Táctica, quedaron bien definidas gracias a los artefactos generados por la metodología empleada, la cual permitió la comprensión de las funcionalidades, elementos y guía de trabajo a seguir por los desarrolladores durante todo el ciclo de desarrollo de la solución. La planificación que se llevó a cabo permitió asignar el tiempo necesario para realizar cada historia de usuario y así evitar que se tengan que agregar más iteraciones de las previstas. El ajuste de la solución a la arquitectura de Unity 3D, permite que la lógica de sus mecanismos sea ajena al usuario y este pueda interactuar con ellos mediante interfaces visuales. Por último, la descripción de las clases de la solución y sus relaciones mediante las tarjetas CRC y el diagrama de clases, proporciona una mejor visión y entendimiento de su estructura, lo que contribuye a reducir el tiempo de desarrollo.

CAPÍTULO 3. IMPLEMENTACIÓN Y PRUEBA

Introducción

Luego de realizado el diseño de un proyecto, es necesario establecer tareas en las iteraciones de implementación para guiar el trabajo. En el desarrollo de este capítulo se aprecia cómo se lleva a cabo la implementación de la solución, definiéndose diferentes tareas de ingeniería por iteraciones. El capítulo culmina con las pruebas que se realizan en conjunto con el cliente para definir si la solución cumple con los objetivos definidos.

3.1 Fase de implementación

Según la metodología XP, la implementación o desarrollo del sistema es la parte más importante, porque permite obtener un resultado al culminar cada iteración y de esta manera una versión del producto funcional, el cual debe ser mostrado y probado al cliente, sirviendo de retroalimentación para el equipo de trabajo (Pressman 2010). A continuación, se exponen detalladamente las cinco iteraciones generadas por la planificación descrita en el capítulo anterior, así como las tareas de ingeniería definidas para la realización de cada HU.

Las tareas de ingeniería son descritas para el programador y creadas para ayudar a organizar la implementación exitosa de las HU. Ellas no ofrecen el nivel de detalle requerido para llevar a cabo esta acometida, por tal motivo son divididas generalmente en más de una tarea, las cuales pueden ser escritas en lenguaje técnico y no necesariamente entendibles por el cliente (Pressman 2010).

3.1.1 Primera iteración

En esta iteración se desarrollan las HU de mayor prioridad, con el objetivo de obtener una primera versión del producto con la cámara del juego y la selección de personajes completamente funcional. Para ello se trazaron ocho tareas, las siguientes tablas muestran dos de estas y el resto se pueden encontrar en el anexo 2:

Tabla 21. Tarea 1 Iteración 1

Tarea	
Número de tarea: 1	**Número de HU:** 1
Nombre de la tarea: Implementación de los atributos configurables de la clase *CameraTRTS*	

Tipo de tarea: Implementación	**Estimación:** 2 días
Descripción: Se declaran los atributos configurables para movimiento, altura, enfoque y cursores de la cámara del juego, que se visualizarán en la ventana de inspector de Unity 3D y se crea un prefabricado con este *script* que pueda ser adicionado en la escena.	

Tabla 22. Tarea 5 Iteración 1.

Tarea	
Número de tarea: 5	**Número de HU:** 3
Nombre de la tarea: Implementación de la selección con cuadro de personajes	
Tipo de tarea: Implementación	**Estimación:** 5 días
Descripción: Se implementa la lógica de la selección múltiple de PC con cuadro de selección en la clase *PlayerSelection*. Esta selección se debe hacer utilizando el clic derecho y manteniéndolo presionado para que se dibuje el cuadro en pantalla. Los PC que se encuentren dentro de este deben ser seleccionados.	

Al culminar el desarrollo de la iteración se realizaron las pruebas de aceptación, No. 1, 2, 3, 4, 5 y 6, que se explican en el anexo 4. Estas arrojaron un total de 7 no conformidades, 4 significativas y 3 no significativas que fueron solucionadas antes de dar inicio a la siguiente iteración.

3.1.2 Segunda iteración

En esta iteración, se incorpora a la versión anterior los mecanismos necesarios para seleccionar personajes e indicarles un destino al cual deben dirigirse mientras ejecutan determinada forma de locomoción. A continuación, se muestran dos de estas y el resto se encuentra en el anexo 2:

Tabla 23. Tarea 3 Iteración 2

Tarea	
Número de tarea: 3	**Número de HU:** 9
Nombre de la tarea: Implementación de indicadores de selección	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementa en la clase *PlayersManager* el mecanismo para activar y desactivar los indicadores de selección de los PC, declarados en la clase *Player*. Estos indicadores se activan para resaltar que la unidad está seleccionada y puede ser controlada.	

Tabla 24. Tarea 7 Iteración 2

Tarea	
Número de tarea: 7	**Número de HU:** 11
Nombre de la tarea: Implementación del control de la máquina de estado de locomoción	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se adiciona en las funciones implementadas anteriormente en la clase *Locomotion*, el código necesario para controlar las animaciones de la máquina de estado de locomoción.	

En el acápite 3.2.2 y en el anexo 4 se explican las pruebas de aceptación, No. 7, 8, 9 y 10, realizadas al culminar la iteración.

3.1.3 Tercera iteración

Esta iteración incorpora a la solución el control de los PC para indicarles que se trasladen hacia un destino mientras ejecuta determinada forma de locomoción e incorpora también la clase padre de la que heredan las acciones con los atributos configurables comunes. A continuación, se muestran dos de estas y el resto se encuentra en el anexo 2:

Tabla 25. Tarea 2 Iteración 3

Tarea	
Número de tarea: 2	**Número de HU:** 7
Nombre de la tarea: Implementación del control de la forma de locomoción Arrastrarse	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Se debe agregar en la clase *PlayerSelection* los mecanismos para controlar que los PC se tiendan en el suelo y se arrastren, luego de ser seleccionados, para esto se deben utilizar las funciones que brinda la clase *Locomotion*. Para indicarle a un PC que se arrastre, primero tiene que estar tendido y esto se realiza mediante una tecla determinada, de esta manera el PC puede arrastrarse haciendo uso de un simple clic.	

Tabla 26. Tarea 5 Iteración 3

Tarea	
Número de tarea: 5	**Número de HU:** 12

Nombre de la tarea: Implementación de la lógica de la clase *Action*	
Tipo de tarea: Implementación	**Estimación:** 5 días
Descripción: Se implementan los mecanismos principales de la clase *Action* que serán heredados por las acciones de cada PC.	

Al culminar el desarrollo de la iteración se realizaron las pruebas de aceptación, No. 11 y 12, explicadas en el acápite 3.2.2 y en el anexo 4.

3.1.4 Cuarta iteración

La cuarta iteración tiene como objetivo de incorporar a esta versión la base completa para que el usuario pueda implementar las acciones de los PC y un prefabricado con la interfaz del juego modificable pero aun sin su lógica implementada. A continuación, se puede observar una de esta y el resto se encuentra en el anexo 2:

Tabla 27. Tarea 2 Iteración 4

Tarea	
Número de tarea: 2	**Número de HU:** 13
Nombre de la tarea: Implementación de la clase *GroupalActionManager*	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Se implementa la lógica y los atributos configurables de la clase *GroupalActionManager*. Mediante estos atributos, el usuario definirá las acciones grupales que presentaran los PC en el juego. Cada PC podrá contener varias acciones grupales, pero al ejecutar el juego solo poseerá las definidas en esta clase.	

En el anexo 4 se explican las pruebas de aceptación, No. 13, 14, 15 y 16, realizadas al culminar la iteración.

3.1.5 Quinta iteración

Al culminar las tareas planificadas en esta iteración, se obtiene la versión final del paquete para videojuegos de Estrategia Táctica con todas sus mecánicas completamente funcional. A continuación, se puede observar una de esta y el resto se encuentra en el anexo 2:

Tabla 28. Tarea 1 Iteración 5

Tarea	
Número de tarea: 1	**Número de HU:** 16
Nombre de la tarea: Implementación de los atributos configurables de la clase *InterfacePlayerSlot*	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se declaran los atributos configurables de la clase *InterfacePlayerSlot* relacionados con el PC asociado, que se visualizarán en la ventana de inspector de Unity 3D. Este *script* debe ser asignado a cada uno de los cuadros de capacidades para retratos de personajes creados en la tarea No 4 de la iteración anterior.	

Al finalizar el desarrollo de la iteración se realizaron las dos últimas pruebas de aceptación, No. 17 y 18, que se explican en el acápite 3.2.2 y en el anexo 4.

3.2 Fase de pruebas

El proceso de pruebas es uno de los pilares fundamentales de la metodología XP, el cual ayuda al cliente a verificar y concretar las funcionalidades de las HU, por lo que favorece la comunicación entre el cliente y el equipo de desarrollo. Esta filosofía ayuda a identificar y corregir fallos u omisiones cometidas en las mismas, por lo que se reduce el número de errores no detectados así como el tiempo entre la introducción de éste en el sistema y su detección (Pressman 2010). La metodología XP propone la realización de dos tipos de pruebas, pruebas unitarias y pruebas de aceptación, descritas a continuación.

3.2.1 Pruebas unitarias

Son las pruebas diseñadas por los programadores y están enfocadas al código, consisten en verificar de manera manual o automatizada, si una parte específica del código, funciona de acuerdo con los requisitos del sistema. Deben ser definidas antes de realizar el código y repetirse hasta eliminar todos los errores para aumentar la calidad del desarrollo (Joskowicz 2008).

Debido a que en la solución se emplea un gran número de procedimientos encargados de mostrar algún contenido visual (o contribuir a esto) y debido también a que son pocas las funciones que se usan, además de presentar una baja complejidad, se decidió realizar las pruebas unitarias programadas de manera manual para adaptarlas a esta situación y verificar si este contenido visual se muestra correctamente. Se realizaron en cada una de las iteraciones definidas, para evitar el arrastre de errores lógicos a iteraciones posteriores.

En la siguiente figura se muestra un ejemplo de caso de prueba unitaria realizada al procedimiento *DrawSelectionBox* de la clase *PlayersSelection*, el cual en un primer momento arrojó un error. El cuadro se dibujaba en la pantalla, pero no actualizaba sus dimensiones al mover el cursor como debía ser. Dicho error fue solucionado en un segundo momento. En el anexo 3 se pueden encontrar otros ejemplos de estas pruebas, realizadas a algunos de los principales procedimientos de la solución.

Caso de prueba unitaria

```
public GameObject selectionBox; //Prefabricado de cuadro de seleccion

void Test_DrawSelectionBox () {
    PlayersSelection playersSelection = new PlayersSelection ();
    playersSelection.selectionWithBox = true;
    playersSelection.selectionBox = this.selectionBox;
    playersSelection.DrawSelectionBox ();

    Debug.Log ("Al presionar el clic izquierdo y arrastrarlo sin " +
        "soltarlo se debe pintar un cuadro de selección, desde que " +
        "comenzó el clic hasta la posición actual del cursor y " +
        "eliminarse al soltar el clic.");
}
```

Resultado

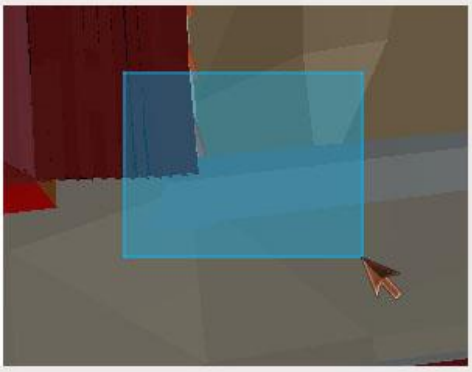


Figura 12. Caso de prueba unitaria Test_DrawSelectionBox.

Eliminar los errores que presenta el código no es suficiente para garantizar que la solución cumple con las especificaciones del cliente, por tal motivo fue necesario realizar otro tipo de prueba descrito a continuación.

3.2.2 Pruebas de aceptación

Las pruebas de aceptación son realizadas por el propio cliente en compañía de uno de los representantes del equipo de desarrollo y se orientan a las funcionalidades del sistema. Su objetivo es comprobar, desde la perspectiva del usuario final, el cumplimiento de las especificaciones de la lista de reservas del producto (Pressman 2010). A continuación, aparecen algunas de las pruebas de aceptación realizadas a la solución, las restantes se pueden encontrar en el anexo 4.

Tabla 29. Caso de Prueba de Aceptación P8HU9

Caso de Prueba de Aceptación	
Código: P8HU9	**Número de HU:** 9
Nombre: Visualizar indicadores de selección	
Descripción: Se verifica si al seleccionar un PC se activan sus indicadores de selección.	

Condiciones de ejecución: El PC debe tener asignado previamente un *script* de unidades.
Entrada/Pasos de ejecución: 1. El usuario selecciona el modelo del personaje en la ventana de escena o de jerarquía. 2. En la ventana de inspector, en el componente visual del *script "Player"* le adiciona en la lista *"Selection Indicators"* los objetos que se activarán al seleccionar el personaje. 3. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.
Resultado esperado: Una vez ejecutado el juego los indicadores de selección de cada PC se desactivan y al seleccionar uno o varios con algún modo de selección, se le activan.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 30. Caso de Prueba de Aceptación P12HU12

Caso de Prueba de Aceptación	
Código: P12HU12	**Número de HU:** 12
Nombre: Insertar y configurar mecánica de acción	
Descripción: Se debe probar que al insertar un *script* de acciones al modelo del personaje y configurarlo, funcione correctamente.	
Condiciones de ejecución: El *script* de acciones debe ser insertado en un personaje que contenga los script *Locomotion* y *Unit*.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el modelo del personaje en la ventana de escena o de jerarquía. 2. En la ventana de inspector adiciona un *script* de acción que herede de la clase *"Action"* y modifica los parámetros configurables para definir el icono, los cursores por defecto de la acción y otros propios de esta o simplemente deja los que posee por defecto.	
Resultado esperado: Se obtiene un personaje con sus acciones insertadas y configuradas.	

Tabla 31. Caso de Prueba de Aceptación P17HU16

Caso de Prueba de Aceptación	
Código: P17HU16	**Número de HU:** 16
Nombre: Configurar capacidades para retratos de personajes	

Descripción: Se prueba que al configurar las capacidades para retratos de personajes, funcionen correctamente.
Condiciones de ejecución: Debe existir en la escena un objeto *"Interface TRTS"* con una o más capacidades para retratos de personajes.
Entrada/Pasos de ejecución: 1. El usuario selecciona cada objeto *"PlayerSlot"* en la ventana de jerarquía o de escena. 2. En la ventana de inspector modifica los parámetros configurables o deja los que posee por defecto. 3. En la ventana de jerarquía, selecciona el objeto *"Interface TRTS*" y adiciona en la lista *"PlayersGridList"*, todos los objetos de las capacidades para retratos de personajes. 4. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.
Resultado esperado: En la interfaz del juego deben aparecer los retratos de todos los PC asignados en la interfaz y cambiar de imagen al seleccionar o al morir uno de ellos. Al dar clic sobre cualquiera de los retratos o al presionar la tecla asignada a cada uno, se debe seleccionar el PC correspondiente. También se podrá realizar una selección puntual sobre la interfaz.

Las pruebas a las funcionalidades se realizaron al finalizar cada una de las cinco iteraciones y se solucionaron las no conformidades detectadas antes de comenzar la siguiente. Algunas de las no conformidades significativas identificadas tienen relación con los siguientes errores: los personajes no cambiaban de la animación caminar a inactivo al llegar a su destino, la cámara no se detenía en los límites de terreno y los personajes no se seleccionaban con sus correspondientes teclas. Mientras las no significativas se centraron en errores ortográficos presentados al nombrar atributos y *scripts*. La siguiente figura muestra por iteración, el número de no conformidades significativas, no significativas y las recomendaciones.

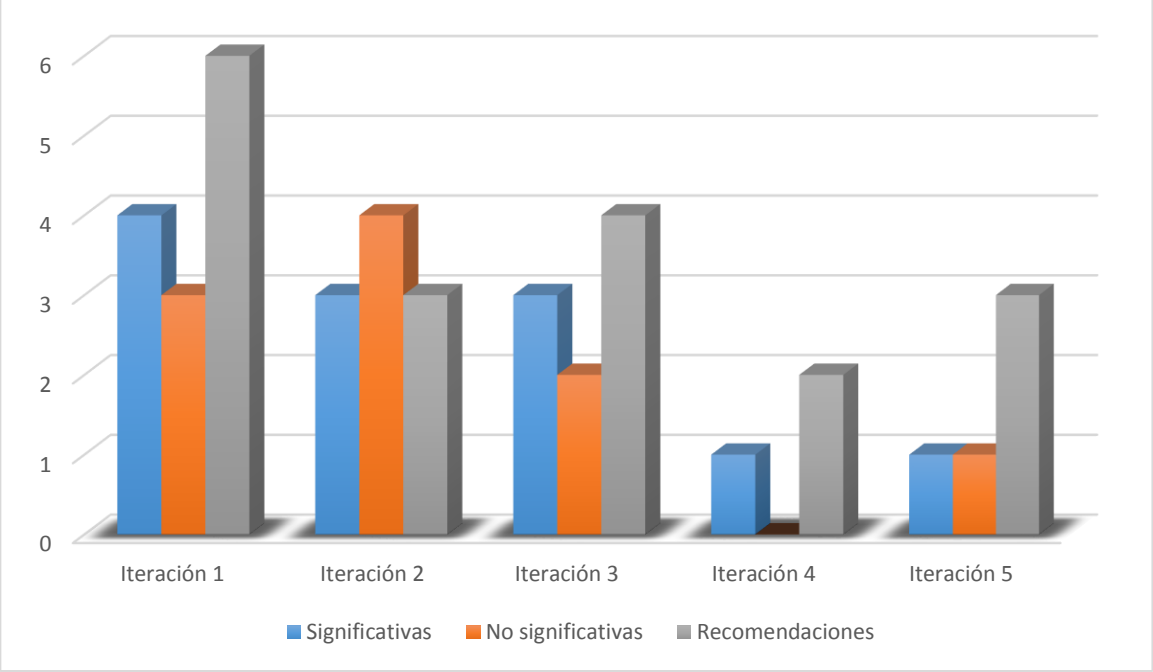


Figura 13. Resultados de las pruebas por iteraciones.

Una vez culminada la fase de pruebas, se realizó un demo en Unity 3D que emplea el paquete desarrollado, para demostrar que este último puede ser reutilizable, flexible y extensible.

3.3 Análisis de resultados

En el demo se representa un escenario de estilo barroco, con varias habitaciones, donde interactúan tres personajes principales en forma de esqueletos humanos con armaduras (el rey, guerrero y mago). Estos conforman un equipo y se complementan con sus acciones, que pueden ser propias o comunes. Todos los personajes pueden ejecutar la acción de atacar que sería común y en el caso del mago presenta también las acciones propias de curar a otro compañero y recoger frascos de poción curativa por todo el entorno. El objetivo del demo, es que esos personajes puedan infiltrarse hacia una puerta mágica que les permitirá regresar a su mundo. En el trayecto se podrán encontrar con varios NPC que fueron creados con una IA sencilla, para enriquecer el entorno y para que los personajes tuvieran contrincantes donde probar las acciones. Los modelos de los personajes y escenarios fueron descargados del Asset Store de Unity 3D (Technologies 2015). La estética fue mejorada con la incorporación de efectos de partículas, iluminación y objetos con comportamientos, que se acoplaron con las funcionalidades del paquete. En la siguiente figura se muestra una captura de pantalla del demo en ejecución.

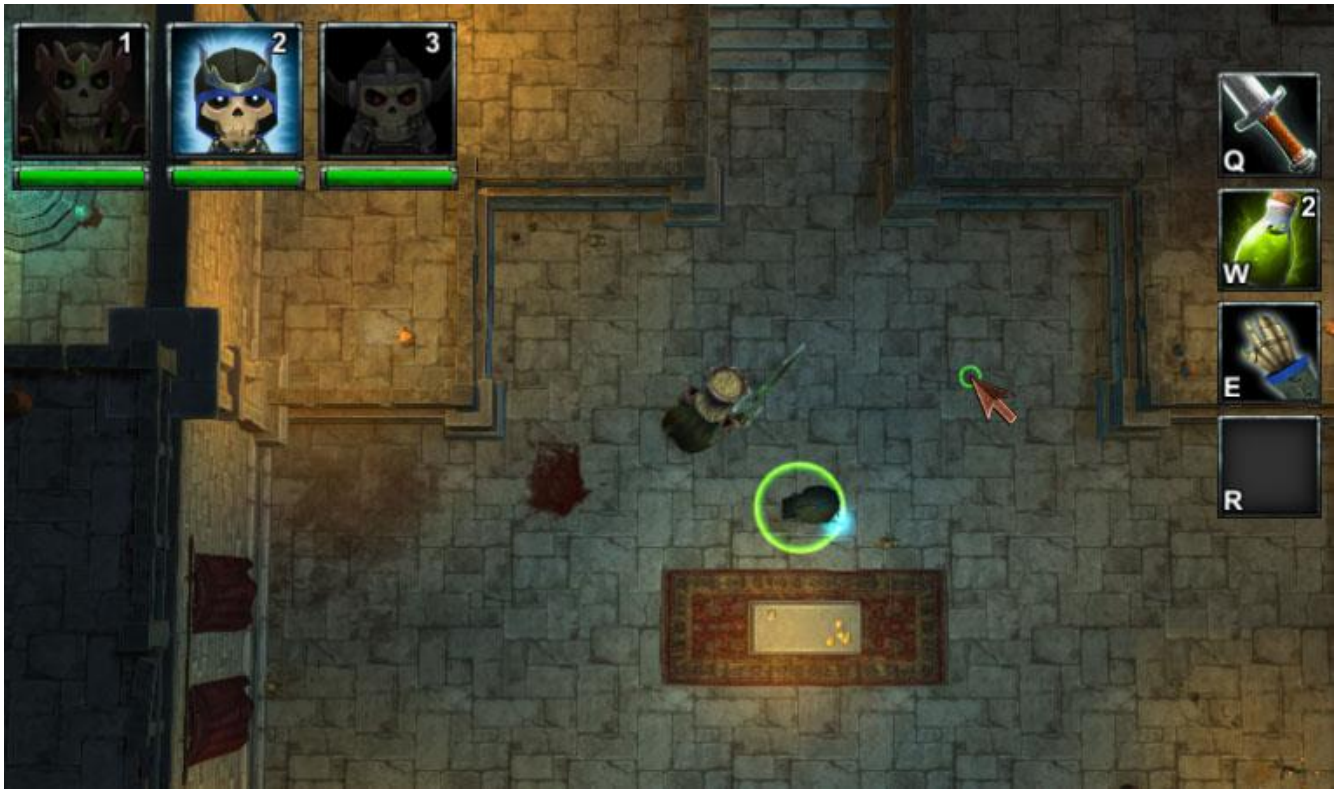


Figura 14. Captura de pantalla del demo realizado.

En esta figura se puede apreciar la selección de un personaje con su indicador de selección y su retrato en la interfaz activados, por otro lado, se encuentran visualizadas las acciones correspondientes a este y se observa como se le indica el punto hacia donde debe dirigirse.

Para demostrar cómo se refleja la reusabilidad, flexibilidad y escalabilidad en el demo, primero es necesario tener claro los siguientes términos. Para ello se usaron como referencia las cualidades del software enunciadas por Roger S. Pressman (Pressman 2010):

Reusabilidad: la posibilidad de utilizar un sistema construido anteriormente para resolver un problema nuevo.

Flexibilidad: el esfuerzo necesario para modificar un sistema que ya está en funcionamiento.

Extensibilidad: un sistema es extensible cuando pueden incorporarse nuevas características al mismo sin mayor impacto sobre las características actuales.

A continuación, se explicará de qué manera cada una de estas cualidades se trataron en el demo.

El primer paso para utilizar el paquete desarrollado fue importarlo en el nuevo proyecto e insertar las mecánicas que presenta a la escena. Esta inserción puede realizarse de varias maneras. Por ejemplo, las mecánicas Cámara, Selección e Interfaz del juego vienen ensambladas en un *prefab* con los componentes y propiedades que utiliza y son configuradas visualmente en la ventana inspector de Unity. En el demo estos tres *prefabs* con mecánicas se arrastraron hacia la escena y se mantuvieron los valores de las variables que tenían por defecto. Las mecánicas Locomoción y Acciones se insertan de otra manera. Estas en lugar de *prefabs,* son *scripts* que se asignan a las unidades y les incorporan comportamientos. También se mantuvieron con los valores que traían por defecto y se obtuvo entonces una primera versión del demo con todas las mecánicas funcionales. De este modo se evidencia la capacidad de reutilización del paquete.

Luego de estar funcionando el demo fueron ajustados los valores de algunas variables. Por ejemplo, se modificaron las velocidades de las distintas formas de locomoción, las animaciones correspondientes a cada personaje, las imágenes para las acciones y los retratos de los PC en la interfaz, las teclas para los controles y los valores para configuración de la cámara. Como las acciones de los personajes son diferentes se programó la lógica de cada una de modo distinto. Esta operación no se realizó visualmente, sino directo en el código de las acciones, que se crearon a partir de la modificación del *script ActionTemplate* brindado como plantilla. Todo este proceso demuestra que las mecánicas son flexibles a modificaciones sin que se afecte su funcionamiento.

La extensibilidad se evidencia con los NPC realizados. Para ello fue necesario crear nuevos *scripts* con comportamientos básicos de inteligencia artificial, que emplean la locomoción y las acciones para controlar a dichos NPC en lugar de ser controlados por el usuario mediante la selección y la interfaz. Estos *scripts* no modifican las funcionalidades de las mecánicas, sino que las utilizan para incorporar nuevas funcionalidades al demo.

Una vez terminado el demo se pasó a la realización de un conjunto de pruebas para analizar el rendimiento del paquete.

3.4 Pruebas de rendimiento

El rendimiento de un videojuego se mide por la cantidad de cuadros por segundo (comúnmente FPS) que se logren visualizar. Para que sea interpretado en tiempo real esta cantidad debe oscilar entre los 30 y 60

FPS (visualización en tiempo real) (Fernández y Angelina 2012). Algunos de los principales procesos presentes en el ciclo de actualización de un videojuego son:

Rendering: se encarga del dibujado de todos los objetos que se encuentren en el campo de visión de la cámara.

Scripts: se encarga de manipular toda la lógica presente en el videojuego.

Physics: se ocupa del control de toda la física que presenta el videojuego.

Garbage Collector: también es conocido como recolector de basura, tiene la tarea de recuperar los espacios en memoria ocupados por objetos que no están en uso.

Mediante la herramienta Profiler de Unity 3D (Technologies 2015), se realizó el análisis del proceso **Scripts** en el demo descrito en la sección anterior, con el objetivo de valorar el rendimiento del paquete realizado en cuanto a CPU[3]. Primero fue probado en modo de ejecución sin la interacción del usuario con los objetos del juego y luego con la interacción de este. En cada uno de estos casos se analizó de tres maneras diferentes: solo el paquete sin elementos gráficos, con todos los elementos gráficos y con todos los elementos gráficos y con *scripts* de IA, ver anexo 5. La computadora donde se llevó a cabo esta tarea tenía las siguientes propiedades: procesador Pentium Dual-Core a 2.30 GHz, 3 Gb de memoria RAM y sistema operativo Windows 8.1 de 64 bits. Las siguientes figuras muestran los tiempos medios en milisegundos (ms) que consume el proceso **Scripts** y todos los procesos del demo en cada prueba.

[3] Unidad central de procesamiento o unidad de procesamiento central (conocida por las siglas CPU, del inglés: *central processing unit*), es el hardware dentro de una computadora u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

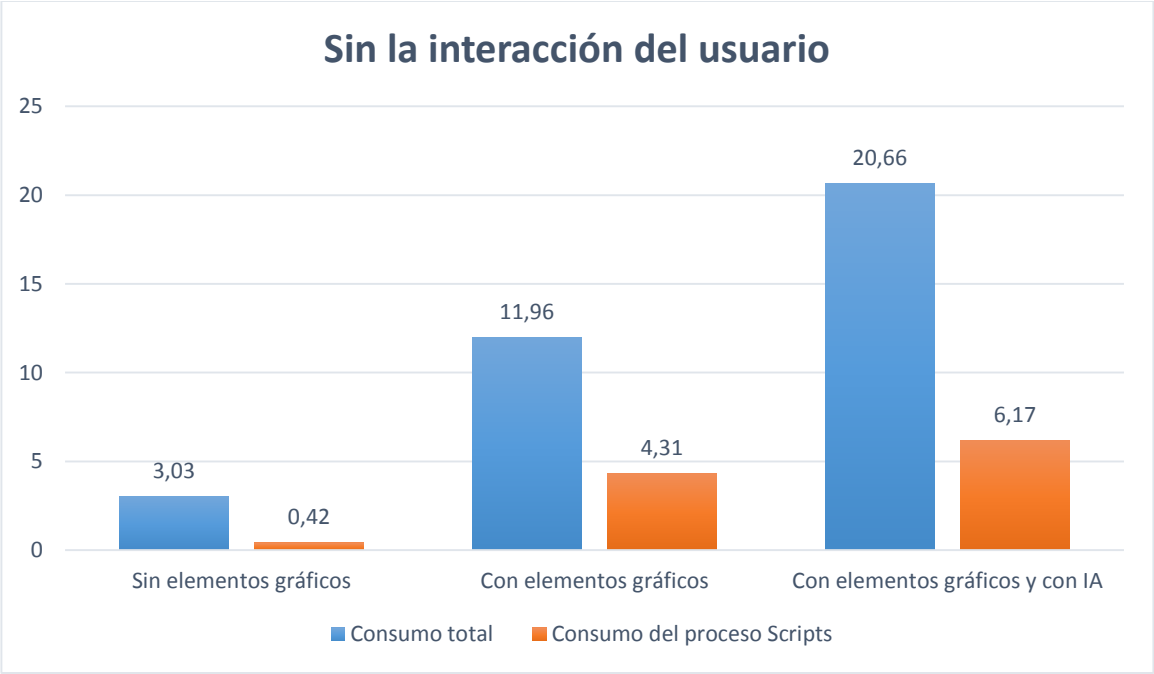


Figura 15. Resultados de las pruebas de rendimiento realizadas al demo sin la interacción del usuario.

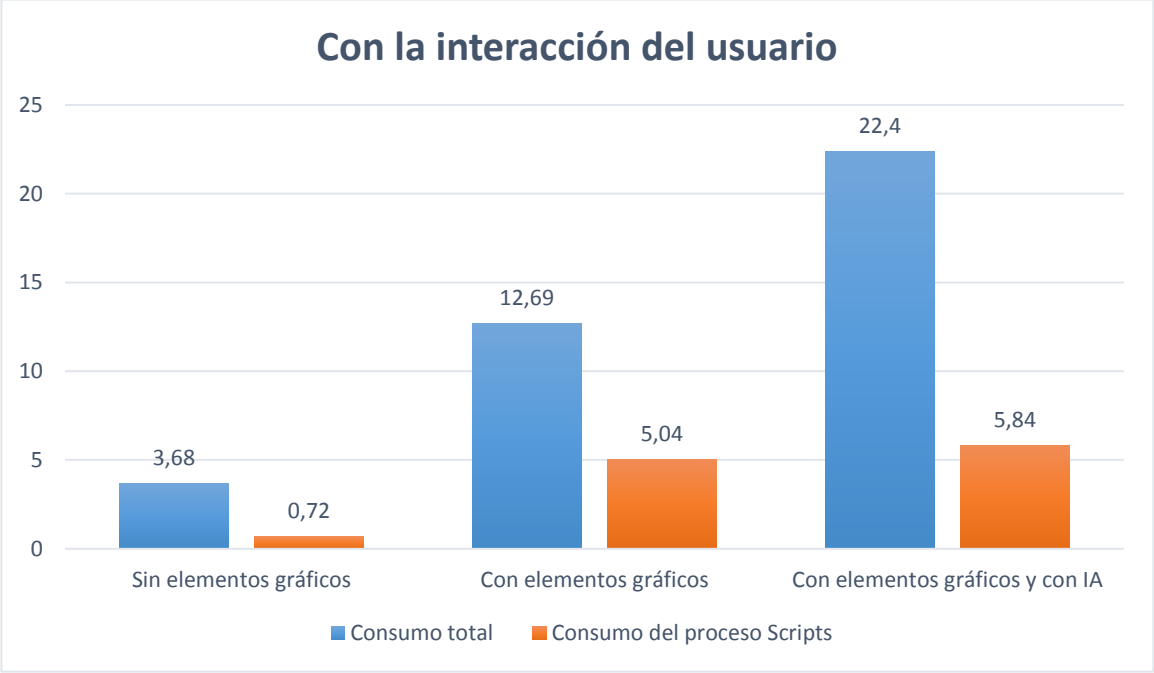


Figura 16. Resultados de las pruebas de rendimiento realizadas al demo con la interacción del usuario.

Según los resultados obtenidos en las pruebas, el tiempo que consume el proceso **Scripts** representa menos de un 50% del total de tiempo para procesar un cuadro. Como proceso completo, compuesto por los demás procesos implicados, el tiempo de respuesta está en el rango de tiempo real (33 ms (30 FPS) - 16 ms (60 FPS)) (Technologies 2015). No obstante se considera importante trabajar en la optimización del paquete para contibuir a un mejor desempeño de los videojuegos que se realicen con este.

Consideraciones parciales

Luego de realizadas las iteraciones, se reconoció la importancia que tuvo organizar el desarrollo por tareas, pues permitieron evitar duplicación de esfuerzos. A medida que se implementaron las tareas, se logró un avance y orden en el proceso de desarrollo de la solución; el tiempo que se le asignó a cada una de estas iteraciones fue suficiente para su desarrollo. Cada vez que se terminaba una iteración se realizaban pruebas unitarias y de aceptación, lo que garantizó que el código funcionara correctamente y posibilitó que el cliente obtuviera resultados desde etapas tempranas del desarrollo de la solución. Finalmente, con el demo realizado se pudo demostrar que el paquete está terminado y cumple con los requerimientos definidos.

CONCLUSIONES

Luego de culminado el trabajo se llegaron a las siguientes conclusiones:

- Con este trabajo se identificó que las mecánicas principales que componen los videojuegos de Estrategia Táctica y que no deben faltar en su desarrollo son: Cámara, Locomoción, Selección, Interfaz del juego, Acciones y las relacionadas con la IA de los NPC.
- El paquete de mecánicas para videojuegos de Estrategia Táctica obtenido como solución, brinda a los desarrolladores en Unity 3D, una base técnica con las principales mecánicas, que se pueden enriquecer con los elementos propios de la historia del videojuego a realizar.

RECOMENDACIONES

Para dar continuidad al desarrollo de la solución se recomienda:

- Trabajar en la optimización del rendimiento de las mecánicas del paquete.
- Agregar al paquete mecánicas que se encarguen de implementar los comportamientos inteligentes que presentan los NPC en este tipo de videojuego.
- Incorporar mecánicas que permitan extender el modo de juego de un solo jugador, a multijugador.

REFERENCIAS BIBLIOGRÁFICAS

ADAMS, E. y DORMANS, J. 2012. *Game Mechanics, Advance Game Desings*. United States of America: s.n. ISBN 978-0-321-82027-3.

AWAD, M.A. 2005. A Comparison between Agile y Traditional Software Development Methodologies. Western:

CARTAGENA99, A., 2011. *Patrones de Diseno*. 2011. Madrid: s.n.

COTERÓN, L.S. 2012. *Arte y videojuegos: mecánicas, estéticas y diseño de juegos en prácticas de creación contemporánea*. S.l.: Universidad Complutense de Madrid.

DÍAZ, R.V. 2015. *Videojuego para la rehabilitación cognitiva enfocado en la Atención*. S.l.: Universidad de las Ciencias Informáticas.

DODERO, J.M. y LLAMAS, C.F., 2003. *Patrones de comportamiento: Observer*. 2003. Madrid: s.n.

ENTERTAINMENT, S., 2001. *User manual, Desperados: Wanted dead or alive*. 2001. S.l.: s.n.

ENTERTAINMENT, S., 2002. *Robin Hood:The Legend of Sherwood, user manual*. 2002. S.l.: s.n.

ESPAÑOLA, A. de A. de la lengua 2015. Real Academia Española. [en línea]. Disponible en: http://dle.rae.es.

FERNÁNDEZ, D.V. y ANGELINA, C.M., 2012. *Desarrollo de Videojuegos, Arquitectura del Motor*. 2012. Ciudad Real: s.n. ISBN 978-84-686-1057-3.

GARCÍA, D.L. 2014. *Metodología ontológica para el desarrollo de videojuegos*. S.l.: Universidad Complutense de Madrid.

GAVALDÀ, J.D. i y NAVARRO, H.T., 2008. *Introducción a los videojuegos*. 2008. S.l.: s.n.

GÓMEZ, A.P. y OZETE, R.E.P. 2013. Videojuego serio para la enseñanza del esqueleto de la cabeza humana. ,

GOOGLE, 2013. *Definición.mx*. 2013. S.l.: s.n.

GROUP, O.M. 2016. Unified Modeling Language (UML). [en línea]. Disponible en: http://www.uml.org/.

JÁUREGUI, J.A.A. 2013. Términos de Enciclopedia Biomecánica. *Locomoción Humana* [en línea]. Disponible en: http://g-se.com/es/biomecanica/wiki/locomocion-humana.

JOSKOWICZ, J. 2008. *Reglas y Prácticas en eXtreme Programming*. Manitoba: Universidad de Vigo, España.

LARRATEGUY, L.I., PIVIDORI, M.D. y SANDRIGO, C. esar M., 2008. *Desarrollo de Software con Mono, una Implementacion Libre de .NET, Multiplataforma e Independiente del Lenguaje*. 2008. Santa Fe: s.n.

MOREIRA, E.M. y VELÁZQUEZ, Y.G. 2013. *Sistema para la verificación de personas en línea a través de huellas dactilares*. S.l.: Universidad de las Ciencias Informáticas.

PALACIOS, O.S. 2013. *Módulo para la creación de videojuegos para jugadores virtuales de tipo batalla utilizando Unity 3D*. La Habana: Universidad de las Ciencias Informáticas.

PEREIRA, A.M.M., 2014. *El proceso productivo del videojuego: fases de producción*. 2014. Madrid: s.n.

PRESSMAN, R.S. 2010. *Software Engineering a Practitioner's Approach*. 7. S.l.: s.n. ISBN 978-0-07-337 597 -7.

RONDÓN, E.B. cuba 2012. *Fusión de la información de los servicios de la comunidad universitaria de la UCI con el metaverso OpenSim*. La Habana: Universidad de las Ciencias Informáticas.

SECO, J.A.G., 2001. *El lenguaje de programación C#*. 2001. S.l.: s.n.

SOMMERVILLE, I. 2005. *Ingienería de Software*. 7. Madrid: s.n.

STUDIOS, P., 1998. *Commandos, user manual*. 1998. S.l.: s.n.

TECHNOLOGIES, U. 2011. Motores gráficos. [en línea]. Disponible en: http://mat.ub.edu/futurs_ub/activitats/Matefest/2011/triptics/motoresgraficos.pdf.

TECHNOLOGIES, U. 2015. Unity Manual. [en línea]. Disponible en: http://docs.unity3d.com/Manual/index.html.

TECHNOLOGIES, U. 2016a. Fury Framework. [en línea]. Disponible en: https://www.assetstore.unity3d.com/en/#!/content/3000.

TECHNOLOGIES, U. 2016b. Generic Strategy Framework. .

UNITY TECHNOLOGIES 2016. RTS Development Framework. [en línea]. Disponible en: https://www.assetstore.unity3d.com/en/#!/content/45789.

VLISSIDES, J., HELM, R., JOHNSON, R. y GAMMA, E. 1995. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, vol. 49, no. 120, pp. 11.

ZYDA, M., 2005. *From Visual Simulation to Virtual Reality to Games*. 2005. S.l.: s.n.

GLOSARIO DE TÉRMINOS

Jugabilidad: facilidad de uso que un juego, especialmente un videojuego, ofrece a sus usuarios.

Reglas: son los aspectos que conforman la conducta del jugador, definen el progreso del videojuego, las acciones que se pueden ejecutar, el modo de hacerlo y el orden.

Proyección isométrica: sistema de representación gráfico, llamado también axonométrica cilíndrica ortogonal. Es la representación visual de un objeto tridimensional en dos dimensiones, en la que los tres ejes ortogonales principales, al proyectarse, forman ángulos de 120º, y las dimensiones paralelas a dichos ejes se miden en una misma escala.

HUD: conjunto de iconos, números, mapas, que durante el juego brindan información sobre el estado de la partida y los personajes, como por ejemplo vida restante, ubicación, munición y objetos en uso.

Renderizar: es un término usado para referirse al proceso de generar una imagen desde un modelo 3D.

Streaming: es un término que hace referencia a la distribución digital de multimedia a través de una red de computadoras.

Plugin: es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

Script: generalmente es un documento de texto que contiene instrucciones, escritas en códigos de programación para ejecutar diversas funciones en el interior de un programa informático.

Prefabs: son un tipo de *asset* que permiten almacenar un objeto *GameObject* de Unity 3D con todos sus componentes y propiedades, para poder ser instanciados en el videojuego cada vez que se estime oportuno.

Algoritmo A*: es un algoritmo de búsqueda en grafos que encuentra la ruta de menor coste entre dos puntos.

Biblioteca: en ciencias de la computación, una biblioteca es un conjunto de implementaciones funcionales y codificadas en un lenguaje de programación que se emplean para desarrollar software.

ANEXO 1. HISTORIAS DE USUARIO

Tabla 32. HU Insertar y configurar mecánica de selección de personajes

Historia de usuario	
No 2: Insertar y configurar mecánica de selección	
Prioridad: alta	**Nivel de complejidad:** bajo
Estimación: 5 días	**Iteración asignada:** 1
Descripción: El usuario tomará el prefabricado de la mecánica de selección y lo arrastrará hacia la escena. Luego tendrá la opción de configurar los parámetros de selección en el inspector de Unity 3D.	
Observaciones:	

Tabla 33. HU Seleccionar un personaje con el clic izquierdo del mouse

Historia de usuario	
No 4: Seleccionar un personaje con el clic izquierdo del *mouse*	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 3 días	**Iteración asignada:** 1
Descripción: El usuario podrá seleccionar un personaje presente en la escena presionando el clic izquierdo del *mouse* sobre este.	
Observaciones:	

Tabla 34. HU Realizar selección puntual de personajes

Historia de usuario	
No 5: Realizar selección puntual de personajes	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 3 días	**Iteración asignada:** 1
Descripción: El usuario podrá seleccionar varios personajes presente en la escena, manteniendo presionada una determinada tecla y dando clic izquierdo sobre uno de estos.	
Observaciones: La tecla debe definirse en la HU 2.	

Tabla 35. HU Realizar selección total de personajes

Historia de usuario	
No 6: Realizar selección total de personajes	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 2 días	**Iteración asignada:** 1
Descripción: El usuario podrá seleccionar y deseleccionar a todos los personajes mediante una tecla definida.	
Observaciones: La tecla debe definirse en la HU 2.	

Tabla 36. HU Indicar al personaje que ejecute determinada forma de locomoción

Historia de usuario	
No 7: Indicar al personaje que ejecute determinada forma de locomoción	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 13 días	**Iteración asignada:** 3
Descripción: El sistema debe permitir que el usuario indique un destino a los personajes seleccionados y al dirigirse hacia ese sitio, estos ejecuten una forma de locomoción (caminar, correr o arrastrarse) especificada por el usuario.	
Observaciones: Para indicar que el personaje camine usar un clic izquierdo simple, al igual que para arrastrarse, solo que esto último depende de si el personaje está tendido y para correr usar doble clic izquierdo.	

Tabla 37. HU Insertar y configurar script de unidades

Historia de usuario	
No 8: Insertar y configurar *script* de unidades	
Prioridad: alta	**Nivel de complejidad:** medio
Estimación: 7 días	**Iteración asignada:** 2
Descripción: El usuario debe tener la posibilidad de adicionarle a los modelos de los personajes un *script* que los convierta en unidades y poderle configurar parámetros como la salud, el nombre, el cursor, los indicadores de selección y los audios que ejecutaran al ser seleccionados, al indicarles un destino y al morir.	

Observaciones: Los personajes en el videojuego son unidades, otro ejemplo de estas pudieran ser las unidades enemigas.

Tabla 38. HU Visualizar indicadores de selección al seleccionar una unidad

Historia de usuario	
No 9: Visualizar indicadores de selección al seleccionar una unidad	
Prioridad: media	**Nivel de complejidad:** medio
Estimación: 3 días	**Iteración asignada:** 2
Descripción: Al seleccionar uno o varios personajes en tiempo de ejecución se deben mostrar sus indicadores de selección y desactivar los de los personajes no seleccionados.	
Observaciones: Los indicadores de selección pueden ser objetos o prefabricados que se activan para indicar que el personaje está seleccionado y deben ser definidos en la HU 8.	

Tabla 39. HU Configurar máquina de estado de locomoción

Historia de usuario	
No 11: Configurar máquina de estado de locomoción	
Prioridad: alta	**Nivel de complejidad:** alto
Estimación: 5 días	**Iteración asignada:** 2
Descripción: El sistema debe brindar una máquina de estado base para que el usuario pueda modificarla en la ventana *Animator* de Unity 3D y agregarle nuevos estados si así lo desea y necesita, además de definir sus propias animaciones para cada forma de locomoción.	
Observaciones:	

Tabla 40. HU Insertar y configurar mecánica de acción

Historia de usuario	
No 12: Insertar y configurar mecánica de acción	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 8 días	**Iteración asignada:** 3
Descripción: El usuario debe tener la posibilidad de adicionarle a los modelos de los personajes uno o varios *scripts* de acciones y poderle configurar los parámetros que poseen por defecto.	

Observaciones:

Tabla 41. HU Brindar al usuario una base para la implementación de las acciones

Historia de usuario	
No 13: Brindar al usuario una base para la implementación de las acciones	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 10 días	**Iteración asignada:** 4
Descripción: El usuario debe contar con un *script* base que le sirva como plantilla para implementar la lógica de cada acción. Además podrá usar los parámetros configurables mencionados en la HU 12 y podrá adicionar los que le sean necesarios para la implementación.	
Observaciones:	

Tabla 42. HU Insertar y posicionar la interfaz del juego en la pantalla según el gusto del usuario

Historia de usuario	
No 14: Insertar y posicionar la interfaz del juego en la pantalla según el gusto del usuario	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 3 días	**Iteración asignada:** 4
Descripción: El usuario tendrá la posibilidad de tomar el prefabricado de la Interfaz del juego y arrastrarlo hacia la escena. Luego de esto debe poder ubicar cada componente en el lugar de la pantalla que desee.	
Observaciones:	

Tabla 43. HU Insertar y eliminar capacidades para retratos de personajes y acciones

Historia de usuario	
No 15: Insertar y eliminar capacidades para retratos de personajes y acciones	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 5 días	**Iteración asignada:** 4
Descripción: El usuario podrá eliminar e insertar nuevas capacidades para personajes y acciones en la interfaz según las necesidades del juego.	

Observaciones:

Tabla 44. HU Configurar capacidades para retratos de personajes

Historia de usuario	
No 16: Configurar capacidades para retratos de personajes	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 7 días	**Iteración asignada:** 5
Descripción: El usuario podrá configurar los parámetros de cada capacidad para retratos de personajes como, la tecla para seleccionarlo, el retrato del personaje, la barra de salud y podrá asociar a cada capacidad su respectivo personaje presente en la escena. Los personajes deben seleccionarse al presionar sobre los retratos.	
Observaciones:	

Tabla 45. HU Configurar capacidades de acciones

Historia de usuario	
No 17: Configurar capacidades de acciones	
Prioridad: media	**Nivel de complejidad:** alto
Estimación: 8 días	**Iteración asignada:** 5
Descripción: El usuario podrá configurar los parámetros de cada capacidad para acciones como, la tecla para seleccionarlo. Estas capacidades deben llenarse con el icono de cada acción de los personajes seleccionados y al presionarlas activar la acción correspondiente.	
Observaciones:	

ANEXO 2. TAREAS DE INGENIERÍA

Tabla 46. Tarea 2 Iteración 1

Tarea	
Número de tarea: 2	**Número de HU:** 1
Nombre de la tarea: Implementación de la lógica de la clase *CameraTRTS*	
Tipo de tarea: Implementación	**Estimación:** 5 días
Descripción: Son implementados todos los mecanismos encargados del funcionamiento de la cámara, como: el control y enfoque de la cámara, la activación de cursores por defecto y el ajuste de la altura de la cámara según las irregularidades del terreno.	

Tabla 47. Tarea 3 Iteración 1

Tarea	
Número de tarea: 3	**Número de HU:** 2
Nombre de la tarea: Implementación de la clase *PlayersManager*	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementa la lógica de la clase *PlayersManager* y se declara una lista con los PC seleccionados que se pueda visualizar en la ventana de inspector de Unity 3D.	

Tabla 48. Tarea 4 Iteración 1

Tarea	
Número de tarea: 4	**Número de HU:** 2
Nombre de la tarea: Implementación de los atributos configurables de la clase *PlayerSelection*	
Tipo de tarea: Implementación	**Estimación:** 2 días
Descripción: Se declaran los atributos configurables relacionados con los modos de selección de la clase *PlayerSelection* que se visualizarán en la ventana de inspector de Unity 3D y se crea un prefabricado con este *script* y el *script PlayersManager* que pueda ser adicionado en la escena.	

Tabla 49. Tarea 6 Iteración 1

Tarea	
Número de tarea: 6	**Número de HU:** 4

Nombre de la tarea: Implementación de la selección simple de personajes	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementa la lógica de la selección simple de PC en la clase *PlayerSelection,* que permita seleccionar a un PC dando clic directamente sobre él en la escena.	

Tabla 50. Tarea 7 Iteración 1

Tarea	
Número de tarea: 7	**Número de HU:** 5
Nombre de la tarea: Implementación de la selección puntual de personajes	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementa la lógica de la selección puntual de PC en la clase *PlayerSelection,* que permita seleccionar varios PC en el escenario del juego manteniendo presionada una determinada tecla y dando clic izquierdo sobre uno de estos.	

Tabla 51. Tarea 8 Iteración 1

Tarea	
Número de tarea: 8	**Número de HU:** 6
Nombre de la tarea: Implementación de la selección total de personajes	
Tipo de tarea: Implementación	**Estimación:** 2 días
Descripción: Se implementa la lógica de la selección total de PC en la clase *PlayerSelection*, que permita mediante una tecla definida seleccionar y deseleccionar a todos los PC presentes en la escena.	

Tabla 52. Tarea 1 Iteración 2

Tarea	
Número de tarea: 1	**Número de HU:** 8
Nombre de la tarea: Implementación de la clase *Unit*	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Se declaran los atributos configurables para el cursor, la salud, los indicadores de selección y el nombre de las unidades en la clase *Unit* que se visualizarán en la ventana de inspector de Unity 3D	

y se implementan las funcionalidades que son comunes para las clases *Player* y *Enemy* (que heredan de esta).

Tabla 53. Tarea 2 Iteración 2

Tarea	
Número de tarea: 2	**Número de HU:** 8
Nombre de la tarea: Implementación de la clase *Player*	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se declaran los atributos configurables para definir los audios de los PC en la clase *Player* que se sumarán a los heredados de *Unit* y se implementan las funcionalidades propias de estos.	

Tabla 54. Tarea 4 Iteración 2

Tarea	
Número de tarea: 4	**Número de HU:** 10
Nombre de la tarea: Implementación de los atributos configurables de la clase *Locomotion*	
Tipo de tarea: Implementación	**Estimación:** 1 días
Descripción: Se declaran los atributos configurables de velocidad y elección de las formas de locomoción a emplear para la locomoción de las unidades, que conformarán la interfaz a mostrar en la ventana de inspector de Unity 3D.	

Tabla 55. Tarea 5 Iteración 2

Tarea	
Número de tarea: 5	**Número de HU:** 10
Nombre de la tarea: Implementación de la locomoción de las unidades	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Debe implementarse el funcionamiento interno de la clase *Locomotion* para brindar al usuario un conjunto de funciones que permitan trasladar unidades hacia un destino específico.	

Tabla 56. Tarea 6 Iteración 2

Tarea	
Número de tarea: 6	**Número de HU:** 11

Nombre de la tarea: Diseño de la máquina de estado para la locomoción	
Tipo de tarea: Diseño	**Estimación:** 2 días
Descripción: Debe diseñarse una máquina de estado en la ventana *Animator* de Unity 3D para controlar las animaciones de las diferentes formas de locomoción de la unidad. Con esta máquina los usuarios podrán definir qué animación usar en cada estado.	

Tabla 57. Tarea 1 Iteración 3

Tarea	
Número de tarea: 1	**Número de HU:** 7
Nombre de la tarea: Implementación del control de la forma de locomoción Caminar	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Se debe agregar en la clase *PlayerSelection* los mecanismos para controlar el caminar de los PC luego de seleccionarlos, para esto se deben utilizar las funciones que brinda la clase *Locomotion*. Como bien se describe en la historia de usuario No 7, para indicarle a un PC que camine se debe emplear un simple clic.	

Tabla 58. Tarea 3 Iteración 3

Tarea	
Número de tarea: 3	**Número de HU:** 7
Nombre de la tarea: Implementación del control de la forma de locomoción Correr	
Tipo de tarea: Implementación	**Estimación:** 5 días
Descripción: Se debe agregar en la clase *PlayerSelection* los mecanismos para controlar el correr de los PC luego de seleccionarlos, para esto se deben utilizar las funciones que brinda la clase *Locomotion*. Para indicarle a un PC que corra se debe utilizar un doble clic izquierdo.	

Tabla 59. Tarea 4 Iteración 3

Tarea	
Número de tarea: 4	**Número de HU:** 12
Nombre de la tarea: Implementación de los atributos configurables de la clase *Action*	
Tipo de tarea: Implementación	**Estimación:** 3 días

Descripción: Se declaran los atributos configurables para definir el nombre, los cursores, el ícono y el tipo de acción que se visualizarán en la ventana de inspector de Unity 3D y serán comunes para todas las acciones.

Tabla 60. Tarea 1 Iteración 4

Tarea	
Número de tarea: 1	**Número de HU:** 13
Nombre de la tarea: Implementación de un *script* base para crear nuevas acciones	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementa un *script* que hereda de la clase *Action* el cual el usuario podrá duplicar y usar como base para implementar la lógica de cada nueva acción. Este debe contar con dos procedimientos polimórficos necesarios, uno donde se implementa el comportamiento que ejecutará la acción al iniciarse y otro para cuando se finalice.	

Tabla 61. Tarea 3 Iteración 4

Tarea	
Número de tarea: 3	**Número de HU:** 13
Nombre de la tarea: Implementación de la clase *InteractiveObject*	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Se implementan los mecanismos de la clase *InteractiveObject*, la cual se asignará a los objetos con los que el PC interactuará mediante el empleo de acciones. Con estos objetos solo interactuará el PC que posea una acción que lo permita.	

Tabla 62. Tarea 4 Iteración 4

Tarea	
Número de tarea: 4	**Número de HU:** 14
Nombre de la tarea: Diseño de la Interfaz de juego	
Tipo de tarea: Diseño	**Estimación:** 3 días
Descripción: Se debe diseñar y crear el prefabricado de la interfaz del juego que pueda ser adicionado a la escena y adaptarse. Este debe contar con las dos áreas principales (área para retratos de personajes	

y área para acciones), con todos sus elementos y que permitan ser distribuidas en la pantalla al gusto del usuario.

Tabla 63. Tarea 5 Iteración 4

Tarea	
Número de tarea: 5	**Número de HU:** 15
Nombre de la tarea: Diseño de capacidades para retratos de personajes y acciones	
Tipo de tarea: Diseño	**Estimación:** 5 días
Descripción: Se deben diseñar y crear los cuadros de capacidades para retratos de personajes y acciones que se integrarán en las dos áreas principales de la interfaz del juego. Se debe establecer también la jerarquía que tendrá cada elemento de la interfaz al visualizarse en la ventana de jerarquía de Unity 3D.	

Tabla 64. Tarea 2 Iteración 5

Tarea	
Número de tarea: 2	**Número de HU:** 16
Nombre de la tarea: Implementación de la lógica de la clase *InterfacePlayerSlot*	
Tipo de tarea: Implementación	**Estimación:** 4 días
Descripción: Son implementados todos los mecanismos encargados del funcionamiento de las capacidades para retratos de personajes. Se debe incorporar también en la clase *PlayersManager* en las funciones de selección de personajes, el código necesario para indicarle a la interfaz del juego que resalte a los personajes seleccionados.	

Tabla 65. Tarea 3 Iteración 5

Tarea	
Número de tarea: 3	**Número de HU:** 17
Nombre de la tarea: Implementación de los atributos configurables de la clase *InterfaceActionSlot*	
Tipo de tarea: Implementación	**Estimación:** 2 días
Descripción: Se declaran los atributos configurables de la clase *InterfaceActionSlot* que se visualizarán en la ventana de inspector de Unity 3D y tienen relación con la activación de la acción asociada. Este	

script debe ser asignado a cada uno de los cuadros de capacidades para acciones creados en la tarea No 4 de la iteración anterior.

Tabla 66. Tarea 4 Iteración 5

Tarea	
Número de tarea: 4	**Número de HU:** 17
Nombre de la tarea: Implementación de la lógica de la clase *InterfaceActionSlot*	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Son implementados todos los mecanismos encargados del funcionamiento de las capacidades acciones. Se debe incorporar también en la clase *PlayersManager* en las funciones de selección de personajes, el código necesario para indicarle a la interfaz del juego que visualice las acciones de los personajes seleccionados.	

Tabla 67. Tarea 5 Iteración 5

Tarea	
Número de tarea: 5	**Número de HU:** 17
Nombre de la tarea: Implementación de la clase *InterfaceTRTS*	
Tipo de tarea: Implementación	**Estimación:** 3 días
Descripción: Se implementan las funciones de la clase *InterfaceTRTS* para controlar los elementos de sus dos áreas principales. Este *script* es asignado al objeto padre de la interfaz del juego.	

ANEXO 3. CASOS DE PRUEBA UNITARIA

Caso de prueba unitaria

```
public GameObject player;
void Start(){
    Test_SelectPlayers ();
}
void Test_SelectPlayers () {
    PlayersManager playersManager
        = GameObject.FindObjectOfType<PlayersManager>();
    playersManager.SelectPlayers (player);
    Debug.Log ("Al ejecutar el juego debe seleccionarse "+
               "el player definido y se debe mostrar en la interfaz "+
               "su correspondiente retrato activado.");
}
```

Resultado

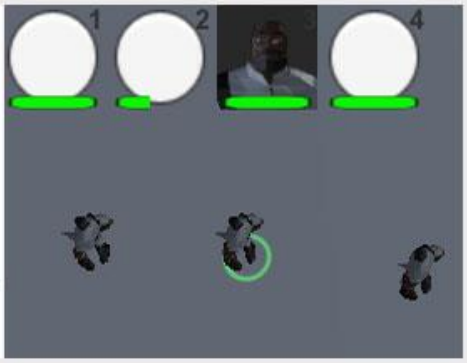


Figura 17. Caso de prueba unitaria Test_SelectPlayers.

Caso de prueba unitaria

```
public GameObject player;
void Start(){
    Test_I_ActualizeActions ();
}
void Test_I_ActualizeActions () {
    InterfaceTRTS interfaceTRTS
        = GameObject.FindObjectOfType(InterfaceTRTS);
    Notification notification
        = new Notification (this, null, player);
    interfaceTRTS.I_ActualizeActions (notification);
    Debug.Log ("Al ejecutar el juego deben visualizarse "+
               "en la interfaz las acciones correspondientes "+
               "al player definido. ");
}
```

Resultado

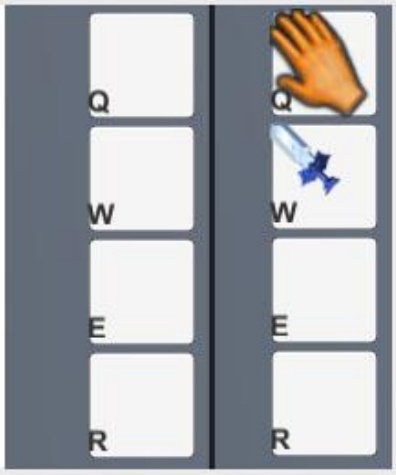


Figura 18. Caso de prueba unitaria Test_I_ActualizeActions.

Caso de prueba unitaria

```
public GameObject player2;
public GameObject player3;
private PlayersManager playersManager;
void Start(){
    playersManager
        = GameObject.FindObjectOfType<PlayersManager>();
    playersManager.SelectPlayer (player2);
    Debug.Log ("Al presionar la tecla 'A' el player3 "+
               "debe mostrarse también seleccionado en la
               "interfaz y con los indicadores de "+
               "selección activados.");
}
void Update(){
    if(Input.GetKeyDown(KeyCode.A)){
        Test_AddPlayer ()
    }
}
void Test_AddPlayer () {
    playersManager.AddPlayer (player3);
}
```

Resultado

Figura 19. Caso de prueba unitaria Test_AddPlayer.

ANEXO 4. CASOS DE PRUEBA DE ACEPTACIÓN

Tabla 68. Caso de Prueba de Aceptación P1HU1

Caso de Prueba de Aceptación	
Código: P1HU1	**Número de HU:** 1
Nombre: Insertar y configurar mecánica de cámara en la escena	
Descripción: Se debe probar que la mecánica de cámara al insertarse en la escena y configurarse, funcione correctamente.	
Condiciones de ejecución: Al insertar el prefabricado de la cámara, no debe existir en la escena otra cámara principal.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el prefabricado *"Camera TRTS"* en la ventana de proyecto, dentro de *Package TRTS -> Prefabs* y lo arrastra hacia la ventana de escena o de jerarquía. 2. En la ventana de inspector modifica los parámetros configurables de la cámara o deja los que posee por defecto. 3. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.	
Resultado esperado: La cámara del juego se controla con las teclas de dirección y acercando el mouse a los bordes de la pantalla, en este caso el cursor por defecto se cambia por otro definido. El enfoque se maneja con el *scroll* del *mouse* y la altura de la cámara se ajusta según las irregularidades del terreno.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 69. Caso de Prueba de Aceptación P2HU2

Caso de Prueba de Aceptación	
Código: P2HU2	**Número de HU:** 2
Nombre: Insertar y configurar mecánica de selección de PC	
Descripción: Se debe probar que la mecánica de selección al insertarse en la escena y configurarse, funcione correctamente.	
Condiciones de ejecución: Al insertar el prefabricado de selección, no debe existir uno igual en la escena.	
Entrada/Pasos de ejecución:	

1. El usuario selecciona el prefabricado *"PlayersSelection TRTS"* en la ventana de proyecto, dentro de *Package TRTS -> Prefabs* y lo arrastra hacia la ventana de escena o de jerarquía. 2. En la ventana de inspector especifica los modos de selección que usará y modifica los restantes parámetros configurables o deja los que posee por defecto.
Resultado esperado: Correcta creación y configuración de los elementos necesarios para la selección de personajes. En el caso de no estar correctamente configurado algún parámetro se notifica en la ventana de inspector.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 70. Caso de Prueba de Aceptación P3HU3

Caso de Prueba de Aceptación	
Código: P3HU3	**Número de HU:** 3
Nombre: Seleccionar múltiples PC con cuadro de selección	
Descripción: Se debe probar que funcione correctamente la selección de varios PC haciendo uso del cuadro de selección.	
Condiciones de ejecución: Debe existir en la escena el objeto *"PlayersSelection TRTS"*.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el objeto *"PlayersSelection TRTS"* en la ventana de jerarquía. 2. Luego en la ventana de inspector activa el modo de selección *"Selection Box"* y especifica un cuadro de selección. 3. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.	
Resultado esperado: Al presionar el clic derecho y arrastrarlo sin soltarlo se dibuja un cuadro en la pantalla y al soltar se seleccionan los PC que quedaron dentro de este y se deseleccionan los restantes. En caso de que no se defina un cuadro de selección, antes de ejecutar el juego, se notificará en la ventana de inspector que debe elegirse.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 71. Caso de Prueba de Aceptación P4HU4

Caso de Prueba de Aceptación	
Código: P4HU4	**Número de HU:** 4

Nombre: Seleccionar un PC
Descripción: Se debe probar que funcione correctamente la selección de un PC dando clic sobre él.
Condiciones de ejecución: Debe existir en la escena el objeto *"PlayersSelection TRTS*".
Entrada/Pasos de ejecución: 1. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.
Resultado esperado: Al presionar el clic izquierdo sobre un PC en la escena, este debe marcarse como seleccionado y el resto desmarcarse. Con esta acción solo se puede seleccionar un PC a la vez.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 72. Caso de Prueba de Aceptación P5HU5

Caso de Prueba de Aceptación	
Código: P5HU5	**Número de HU:** 5
Nombre: Seleccionar múltiples PC con selección puntual	
Descripción: Se debe probar que funcione correctamente la selección puntual de PC.	
Condiciones de ejecución: Debe existir en la escena el objeto *"PlayersSelection TRTS*".	
Entrada/Pasos de ejecución: 1. El usuario selecciona el objeto *"PlayersSelection TRTS*" en la ventana de jerarquía. 2. Luego en la ventana de inspector activa el modo de selección *"Selection Punctual"* y especifica la tecla deseada. 3. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.	
Resultado esperado: Al presionar el clic izquierdo sobre un PC deseleccionado, acompañado de la tecla especificada, este se selecciona y se mantienen también los que estaban seleccionados. Si esta acción se realiza a un PC seleccionado ocurre lo contrario.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 73. Caso de Prueba de Aceptación P6HU6

Caso de Prueba de Aceptación	
Código: P6HU6	**Número de HU:** 6
Nombre: Seleccionar todos los PC	
Descripción: Se debe probar que funcione correctamente la selección de todos los PC a la vez.	

Condiciones de ejecución: Debe existir en la escena el objeto *"PlayersSelection TRTS*".
Entrada/Pasos de ejecución: 1. El usuario selecciona el objeto *"PlayersSelection TRTS*" en la ventana de jerarquía. 2. Luego en la ventana de inspector activa el modo de selección *"Selection All"* y especifica la tecla deseada. 3. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.
Resultado esperado: Al presionar la tecla especificada se deben seleccionar todos los personajes al mismo tiempo y si se presiona nuevamente se deseleccionan. En caso de que no se defina una tecla, antes de ejecutar el juego, se notificará en la ventana de inspector que debe elegirse.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 74. Caso de Prueba de Aceptación P7HU8

Caso de Prueba de Aceptación	
Código: P7HU8	**Número de HU:** 8
Nombre: Insertar y configurar *script* de unidades	
Descripción: Se debe probar que el *script* de unidades al insertarse al modelo del personaje y configurarse, funcione correctamente.	
Condiciones de ejecución: Al insertar el *script* de unidades al modelo, no debe contener un componente de colisión.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el modelo del personaje en la ventana de escena o de jerarquía. 2. En la ventana de inspector le adiciona el *script "Player" o "Enemy*" en el caso de que sea un NPC y modifica los parámetros configurables o deja los que posee por defecto. 3. En esta misma ventana se define si es un PC, el *"Layer"* correspondiente a este.	
Resultado esperado: Se obtiene un modelo convertido en jugador o enemigo listo para seleccionar.	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 75. Caso de Prueba de Aceptación P9HU10

Caso de Prueba de Aceptación	
Código: P9HU10	**Número de HU:** 10

Nombre: Insertar y configurar mecánica de locomoción
Descripción: Se debe probar que el *script* de locomoción al insertarse al modelo del personaje y configurarse, funcione correctamente.
Condiciones de ejecución: Al insertar el *script* de locomoción, el personaje no debe contener otro igual.
Entrada/Pasos de ejecución: 1. El usuario selecciona el modelo del personaje en la ventana de escena o de jerarquía. 2. En la ventana de inspector adiciona el *script "Locomotion"* y modifica los parámetros configurables o deja los que posee por defecto.
Resultado esperado: Correcta creación y configuración de los elementos necesarios para la locomoción de personajes. En el caso de no estar correctamente configurado algún parámetro se notifica en la ventana de inspector.

Tabla 76. Caso de Prueba de Aceptación P10HU11

Caso de Prueba de Aceptación	
Código: P10HU11	**Número de HU:** 11
Nombre: Configurar máquina de estado de locomoción	
Descripción: Se debe probar que al configurar la máquina de estado de locomoción y al insertarse al modelo del personaje, funcione correctamente.	
Condiciones de ejecución: El personaje debe contener el componente *"Animator" y* los *scripts "Locomotion" y "Player"* configurados.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el controlador o máquina de estado *"CharactersLocomotion TRTS"* en la ventana de proyecto y abre la ventana *Animator* de Unity para cambiar las animaciones de los estados si así lo desea. 2. En la ventana de jerarquía o de escena selecciona al personaje y le adiciona el *asset* de la máquina de estado en el componente *"Animator"*.	
Resultado esperado: Se obtiene un personaje con su máquina de locomoción asignada y configurada.	

Tabla 77. Caso de Prueba de Aceptación P11HU7

Caso de Prueba de Aceptación	
Código: P11HU7	**Número de HU:** 7
Nombre: Ejecutar formas de locomoción	
Descripción: Se debe probar que los PC luego de seleccionados ejecuten una determinada forma de locomoción.	
Condiciones de ejecución: Los PC deben contener los *scripts*, *"Locomotion"* y *"Player"* y deben tener asignado en el componente *"Animator"*, el controlador *"CharactersLocomotion TRTS".*	
Entrada/Pasos de ejecución: 1. Cumplidas las condiciones anteriores, el usuario presiona el botón *"play"* en la ventana de juego para ejecutarlo.	
Resultado esperado: Al seleccionar uno o varios PC e indicarles que caminen con un simple clic se deben dirigir hacia su destino ejecutando la animación correspondiente a caminar. Si se les indica que corran con doble clic ejecutan la de correr, si se indica que se tiendan y luego se arrastren con un simple clic deben ejecutar las animaciones correspondientes.	

Tabla 78. Caso de Prueba de Aceptación P13HU13

Caso de Prueba de Aceptación	
Código: P13HU13	**Número de HU:** 13
Nombre: Usar base para la implementación de las acciones	
Descripción: Se verifica si se pueden crear nuevas acciones a partir de un script de plantilla.	
Condiciones de ejecución: No puede existir una acción con el mismo nombre que la creada.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el *script "ActionTemplate"* en la ventana de proyecto, dentro de *Package TRTS -> Scripts -> TRTS -> Actions y lo duplica.* 2. Luego le cambia el nombre que trae por defecto el *script* y la clase y define uno nuevo que no exista.	
Resultado esperado: Se obtiene un *script* hijo de la clase *"Action"* con los métodos, *"ActionLogic"*, donde se implementa la lógica de la acción y *"FinalizeAction"*, donde se le puede implementar un comportamiento al personaje para que lo ejecute cuando finalice la acción.	

Tabla 79. Caso de Prueba de Aceptación P14HU14

Caso de Prueba de Aceptación	
Código: P14HU14	**Número de HU:** 14
Nombre: Insertar y posicionar la interfaz del juego en la pantalla	
Descripción: Se prueba si puede insertarse la interfaz del juego en la pantalla y distribuirse en ella libremente.	
Condiciones de ejecución: No puede existir en la escena otro objeto “Interface TRTS”.	
Entrada/Pasos de ejecución: 1. El usuario selecciona el prefabricado *“Interface TRTS*” en la ventana de proyecto, dentro de *Package TRTS -> Prefabs* y lo arrastra hacia la ventana de escena o de jerarquía. 2. En la ventana de escena, posiciona el área de los retratos de personajes y el área de acciones en el lugar de la pantalla que desee, auxiliándose si es necesario del componente *“RectTranform”* de cada área en la ventana de inspector.	
Resultado esperado: Se obtiene la interfaz del juego con sus dos áreas principales distribuidas en la pantalla.	

Tabla 80. Caso de Prueba de Aceptación P15HU15

Caso de Prueba de Aceptación	
Código: P15HU15	**Número de HU:** 15
Nombre: Insertar y eliminar capacidades para retratos de personajes	
Descripción: Se prueba si se puede insertar y eliminar capacidades para retratos de personajes en la interfaz del juego.	
Condiciones de ejecución: Debe existir en la escena un objeto *“Interface TRTS”*.	
Entrada/Pasos de ejecución: 1. El usuario despliega los hijos del objeto *“Interface TRTS”* en la ventana de jerarquía o de escena. 2. Selecciona el objeto hijo *“InterfaceTRTS_Players”* y despliega también sus hijos. 3. Si desea eliminar uno de estos hijos nombrados *“PlayerSlot”*, lo selecciona y lo elimina con el botón “Suprimir” del teclado. Si lo que desea es insertar uno nuevo, selecciona uno existente y lo duplica.	

Resultado esperado: Al eliminar una de estas capacidades se deben reajustar en la interfaz las restantes. Al insertar una nueva debe incorporarse en la interfaz y reajustarse.

Tabla 81. Caso de Prueba de Aceptación P16HU15

Caso de Prueba de Aceptación	
Código: P16HU15	**Número de HU:** 15
Nombre: Insertar y eliminar capacidades para acciones	
Descripción: Se prueba si se puede insertar y eliminar capacidades para acciones en la interfaz del juego.	
Condiciones de ejecución: Debe existir en la escena un objeto *"Interface TRTS"*.	
Entrada/Pasos de ejecución: 1. El usuario despliega los hijos del objeto *"Interface TRTS"* en la ventana de jerarquía o de escena. 2. Selecciona el objeto hijo *"InterfaceTRTS_Actions"* y despliega también sus hijos. 3. Si desea eliminar uno de estos hijos nombrados *"ActionSlot"*, lo selecciona y lo elimina con el botón "Suprimir" del teclado. Si lo que desea es insertar uno nuevo, selecciona uno existente y lo duplica.	
Resultado esperado: Al eliminar una de estas capacidades se deben reajustar en la interfaz las restantes. Al insertar una nueva debe incorporarse en la interfaz y reajustarse.	

Tabla 82. Caso de Prueba de Aceptación P18HU17

Caso de Prueba de Aceptación	
Código: P18HU17	**Número de HU:** 17
Nombre: Configurar capacidades para acciones	
Descripción: Se prueba que al configurar las capacidades para acciones, funcionen correctamente.	
Condiciones de ejecución: Debe existir en la escena un objeto *"Interface TRTS"* con una o más capacidades para acciones.	
Entrada/Pasos de ejecución: 1. El usuario selecciona cada objeto *"ActionSlot"* en la ventana de jerarquía o de escena. 2. En la ventana de inspector modifica los parámetros configurables o deja los que posee por defecto. 3. En la ventana de jerarquía, selecciona el objeto *"Interface TRTS*" y adiciona en la lista *"ActionsGridList"*, todos los objetos de las capacidades para acciones.	

4. Presiona el botón *"play"* en la ventana de juego para ejecutarlo.
Resultado esperado: En la interfaz del juego deben aparecer las acciones de los personajes seleccionados, primero las acciones grupales y luego las individuales. Al dar clic sobre una de estas acciones o al presionar la tecla asignada a cada capacidad, se debe activar la acción.

ANEXO 5. PRUEBAS DE RENDIMIENTO CON LA HERRAMIENTA PROFILER DE UNITY 3D

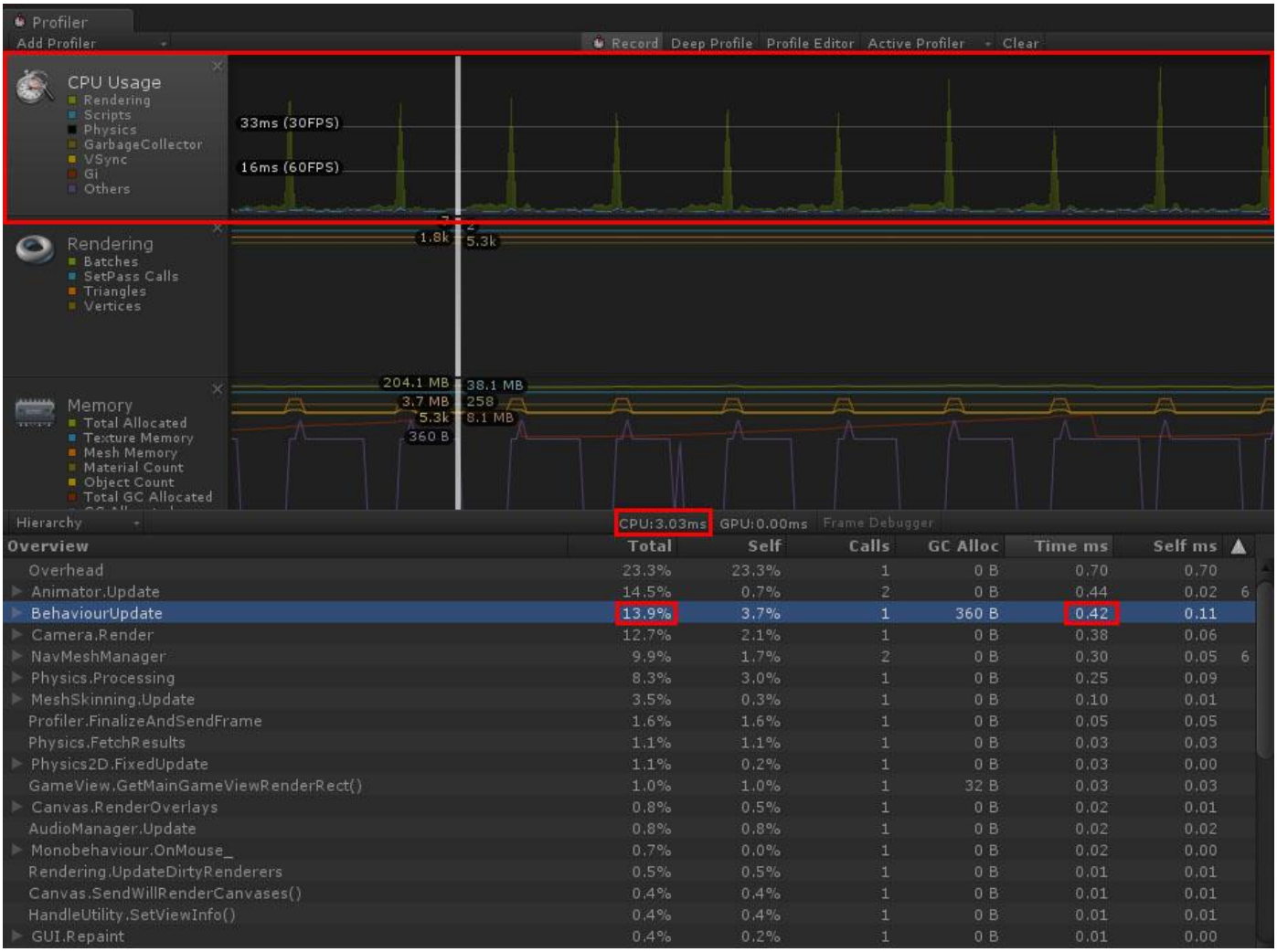


Figura 20. Prueba de rendimiento sin interacción del usuario y sin elementos gráficos en el Profiler de Unity 3D.

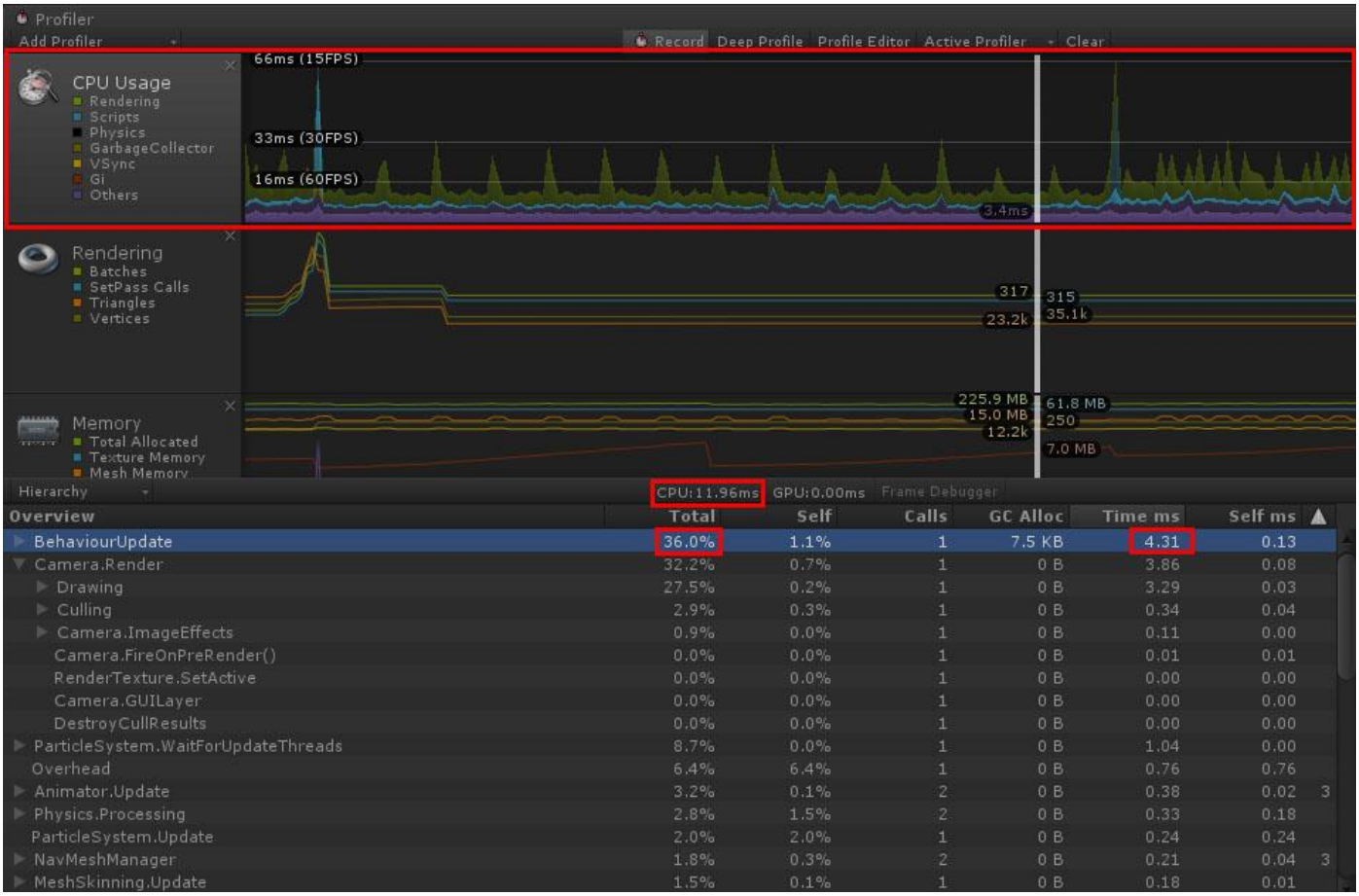


Figura 21. Prueba de rendimiento sin interacción del usuario, con elementos gráficos y sin IA en el Profiler de Unity 3D.

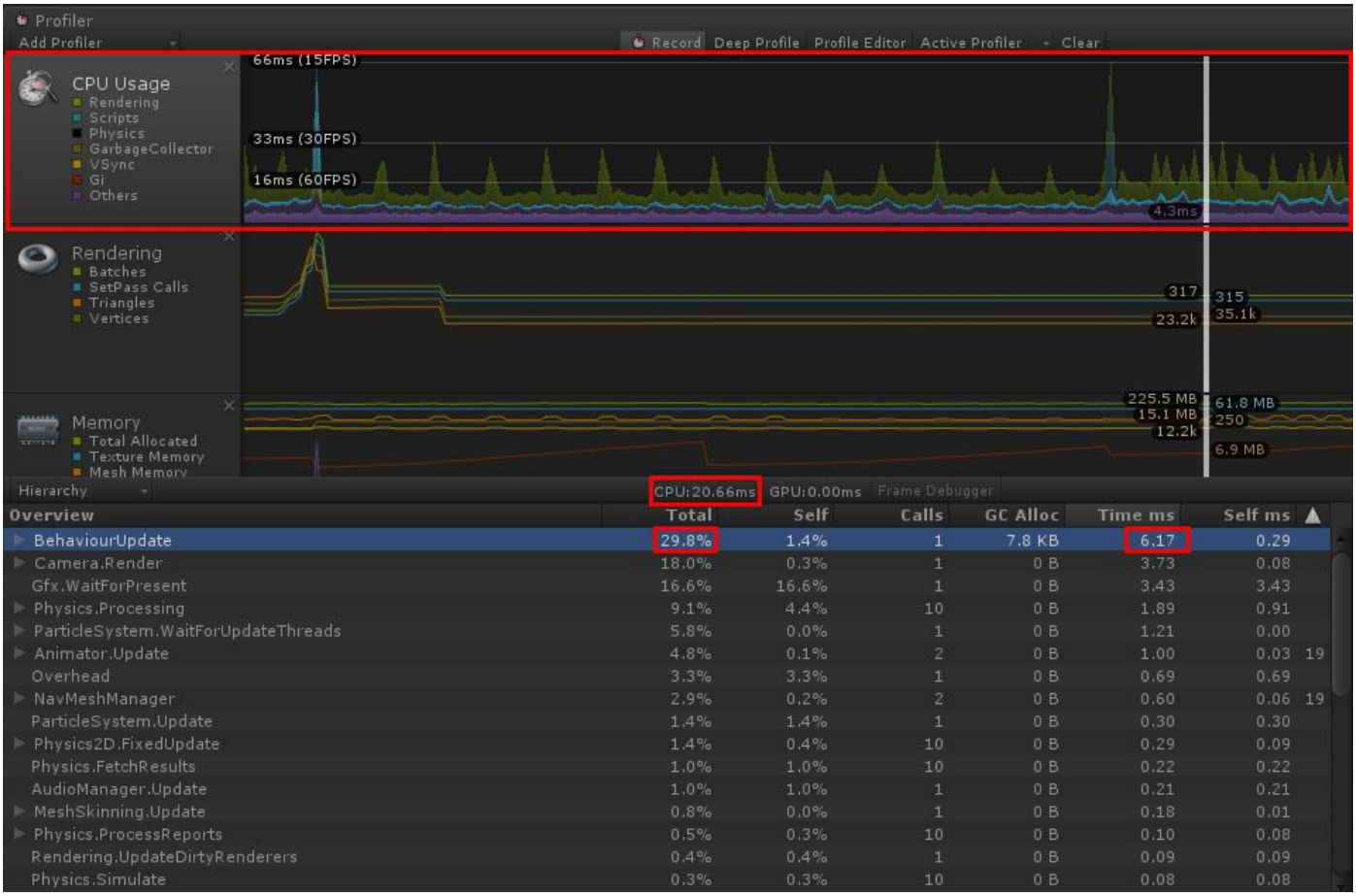


Figura 22. Prueba de rendimiento sin interacción del usuario, con elementos gráficos y con IA en el Profiler de Unity 3D.

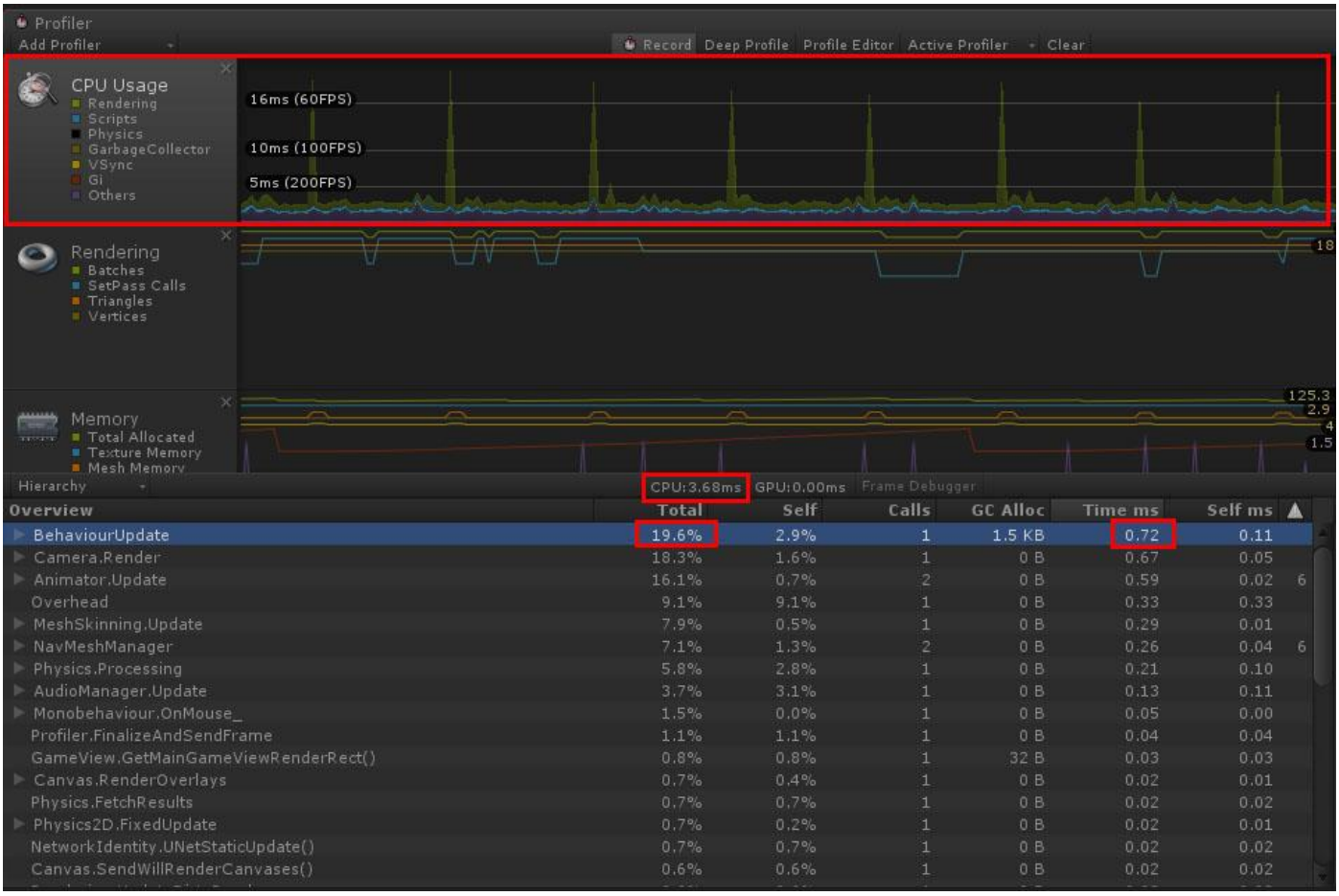


Figura 23. Prueba de rendimiento con interacción del usuario y sin elementos gráficos en el Profiler de Unity 3D.

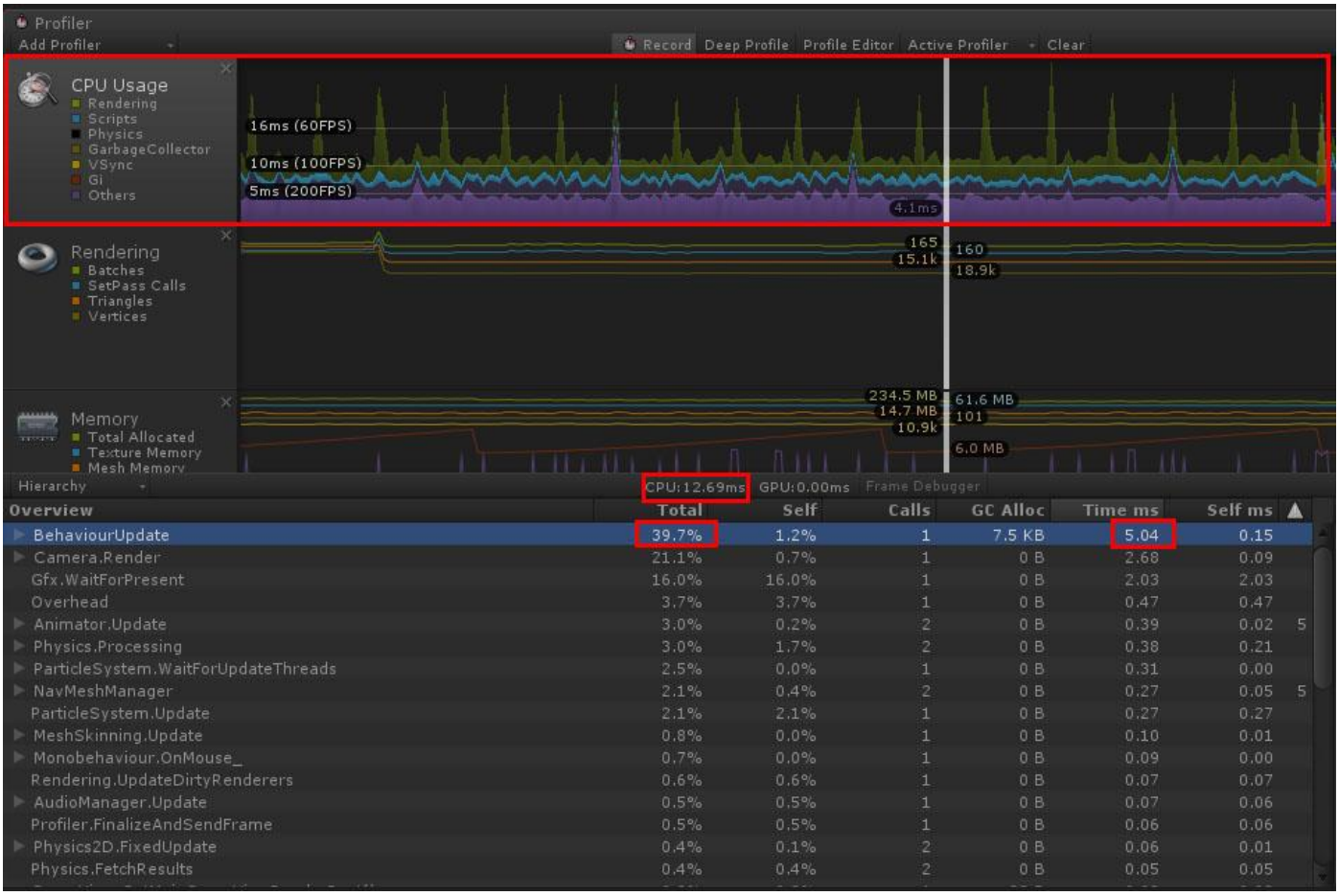


Figura 24. Prueba de rendimiento con interacción del usuario, con elementos gráficos y sin IA en el Profiler de Unity 3D.

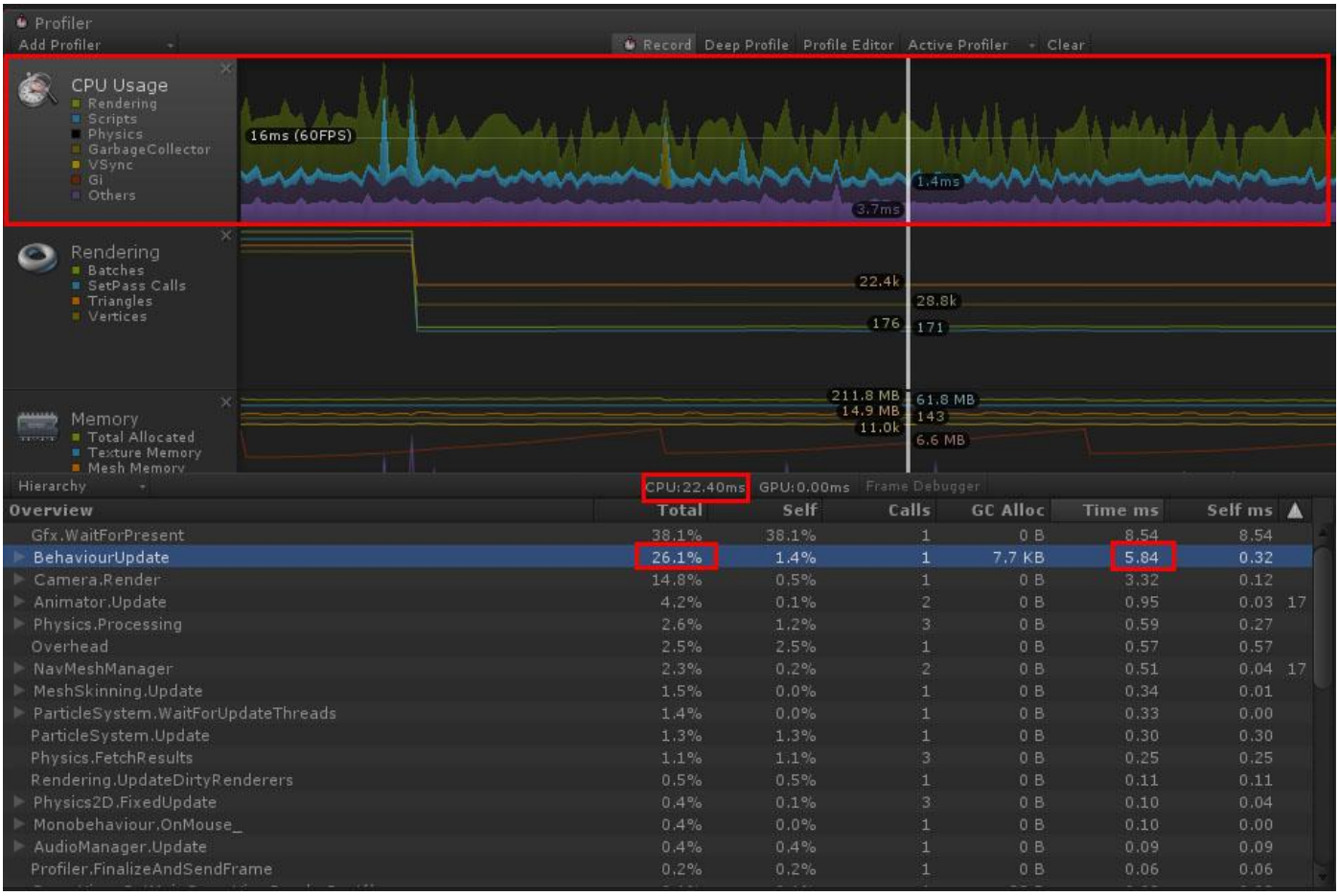


Figura 25. Prueba de rendimiento con interacción del usuario, con elementos gráficos y con IA en el Profiler de Unity 3D.

ANEXO 6. VIDEOJUEGO STARTCRAFT

Figura 26. Captura de pantalla del videojuego StartCraft 1.

ANEXO 7. MALLA DE NAVEGACIÓN DE UNITY 3D

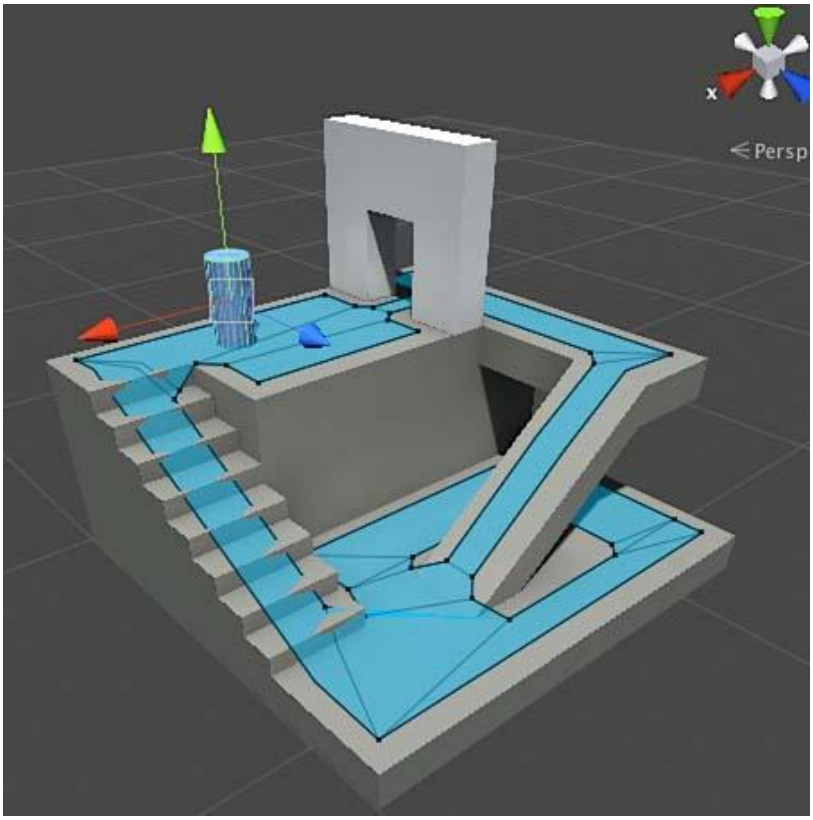


Figura 27. Malla de navegación de Unity 3D. Tomado de (Technologies 2015).