Facultad 5

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: Ingeniería Inversa del Motor GT2D.

Autor: Mariana Pérez Valdés.

Tutor: Ing. Adiel Durán Rodríguez.

Co-Tutor: Ing. Enerys Mesa Morales.

Ciudad de La Habana, 2011.

Año 53 del Triunfo de la Revolución.

DECLARACIÓN DE AUTORÍA.

Declaro ser la autora de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2010.

Mariana Pérez Valdés

_____

Firma del autor

Ing. Adiel Durán Rodríguez**.**

_____

Firma del tutor

Ing. Enerys Mesa Morales.

_____

Firma del Co-Tutor

DATOS DE CONTACTO.

Nombre y apellidos: Adiel Durán Rodríguez**.**

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: aduranr@uci.cu

Ingeniero en Ciencias Informáticas, en la Universidad de Ciencias Informáticas (UCI) en el 2008, Instructor, tres años de experiencia, tres años de graduado.

Nombre y apellidos: Enerys Mesa Morales

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: emesa@uci.cu

Ingeniero en Ciencias Informáticas, en la Universidad de Ciencias Informáticas (UCI) en el 2009, Adiestrada, un año de experiencia, dos años de graduado.

DEDICATORIA.

Dedicarles esta tesis a mis padres que desde que nací se han sacrificado por mí y por mi hermano, dándonos lo que tiene y a veces hasta lo que no tienes para hacernos la vida mucho más asequible.

A mi hermano Susito por ser como dice él para mi, mi ángel de la guarda y apoyarme en mis decisiones siempre por encima de las cosas respetándolas.

A mi Novio Yoxsnel por enseñarme al cómo hacer de mi una mejor profesional y enseñarme todo lo que él sabe.

AGRADECIMIENTOS.

- ✓ **Agradecer antes a la razón de mi existir, a los responsables de que yo sea hoy, quien soy, a los que le debo hoy mi vida, a mis padres Aleida y Jesús.**
- ✓ **A mi hermano que aunque esté lejos, sé que me está dando su apoyo y que está muy feliz por mí.**
- ✓ **A mi novio Yoxsnel por tener tanta paciencia conmigo y por estar siempre que lo necesité.**
- ✓ **A mi Co-tutora Enerys por apoyarme desde un principio.**
- ✓ **A mi Tutor por tener tanta paciencia conmigo.**
- ✓ **A mi profesor de todas las luchas el Licenciado en educación en la especialidad de Física y profesor general de la enseñanza media Hugo Eduardo Estrada Aguilera.**
- ✓ **A mi abuela por ser tan especial para mí.**
- ✓ **A mis tíos por apoyar en todo los que han podido a mis padres y hermano.**
- ✓ **A Toni, Espinosa y Elibetsy por responderme, apoyarme y darme consejos siempre de la mejor forma siempre que los molesté con mis problemas.**
- ✓ **Al profesor Millet por apoyarme en todas las asignaturas que lo necesité.**
- ✓ **A la decana por insistir tanto en que lo más importante en la escuela es formarnos como profesionales.**
- ✓ **A las muchachitas que compartieron estos 5 años conmigo soportando mis pesadeces.**

RESUMEN.

Con la culminación de la presente documentación se generaron algunos de los artefactos ingenieriles que propone la Metodología Open Up para las etapas de Análisis y Diseño, de los mismos se identificaron 12 requisitos funcionales y 5 categorías de los requisitos no funcionales.

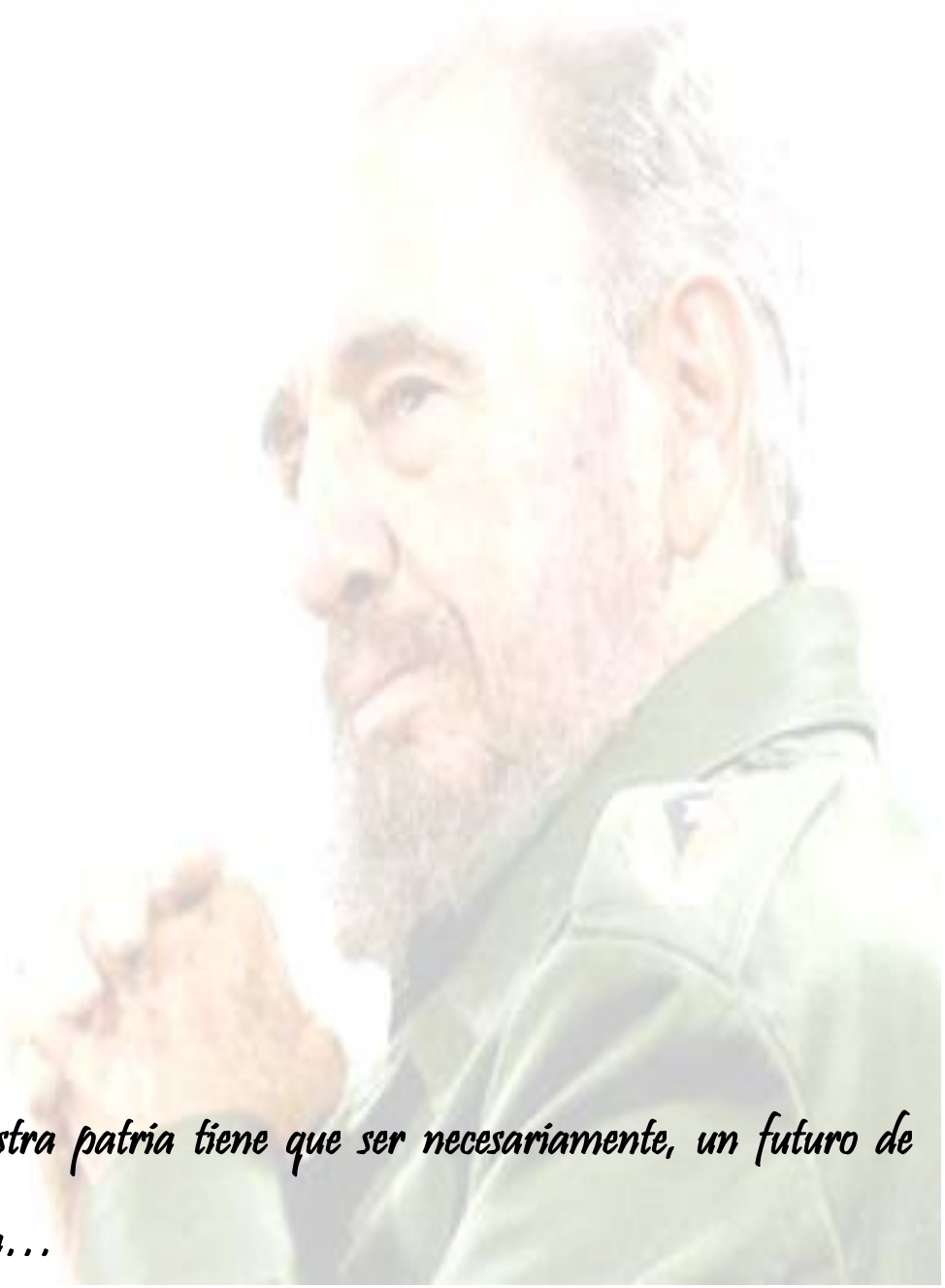
Se identificaron además 12 casos de uso de los cuales 10 son críticos realizándole sus respectivos diagramas de secuencias, se realizó además el diagrama de paquetes explicando en el mismo la relación entre las clases y sus interacciones mediante los patrones de diseño GRASP y GoF.

La relación de los resultados obtenidos se mostró a través de las Matrices de Trazabilidad la cuales se representaron siguiendo un orden lógico de Requisitos Funcionales vs Casos de Uso, Casos de Uso vs Diagramas de Secuencias, Diagramas de Secuencias vs Clases del Diseño y por ultimo entre las Clases del Diseño y los Paquetes de Implementación, para luego hacer una representación de las mismas en el Árbol de Trazabilidad.

Se expone además una recopilación de informaciones, como resultado de las diversas bibliografías que se consultaron, donde se muestra el comportamiento de la aplicación de una Ingeniería Inversa a un software.

PALABRAS CLAVE.

Ingeniería Inversa, Motor de juego, Videojuegos



El futuro de nuestra patria tiene que ser necesariamente, un futuro de hombres de ciencia…

Fidel Castro.

INTRODUCCIÓN

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Según ha ido avanzando la ciencia y la tecnología, el hombre ha tenido entre sus principales objetivos, lograr que las máquinas piensen igual que el ser humano, o al menos tratar tal situación.

Hoy en día el uso de los Videojuegos, ha posibilitado de una forma recreativa enseñarle al mundo lo que una máquina con un código adecuado es capaz de hacer. Desde hace tiempo los Videojuegos han influido en las vidas de las personas, haciendo así que se sientan identificados con los diversos personajes que fueron creados, con el propósito de darle vida a un juego.

Muchas empresas de software han sacado a los mercados nuevos Videojuegos tanto en consola de última generación, como para computadoras, los que a nivel gráfico son increíbles, pero el modo del juego, o sea la tecnología que rige el comportamiento de sus jugadores no ha estado sujeto a muchos cambios.

Actualmente muchos de los Videojuegos son creados para enseñar diversas temáticas en diversas escuelas de una forma más divertida. Desde el 2008 en la Universidad de las Ciencias Informáticas se ha fortalecido la realización de juegos electrónicos, y algunos proyectos que se intentaban realizar ya están agrupados en áreas temáticas, según la arquitectura base de los mismos, lo que ha permitido enfocarlos en un modelo de producción orientado a líneas de productos de software.

En la Facultad 5 de dicha Universidad, han surgido varios proyectos de Investigación y Desarrollo que se dedican al tema de los Videojuegos. Después del surgimiento del Área Temática de Videojuegos y su reconfiguración surge la idea de crear juegos con la colaboración directa del ICAIC, para llevar al ámbito de los juegos, algunas de las series de animados más populares.

El primer juego basado en dibujos animados concluido, fue Fernanda, premio en juegos de habilidad en el primer taller nacional del Videojuegos celebrado en el 2009 por la dirección nacional de los joven club de computación y electrónica del país; se inicia otro sobre El Capitán Plín, y tienen en proyecto otro Video Juego acerca de Elpidio Valdés, con el respaldo de Juan Padrón.

Próximamente se realizará un estudio sobre realizar un convenio con Venezuela en el cual se realizará un juego 3D donde su objetivo principal será representar los hechos históricos de dicho país.

Con el tiempo se ha logrado adquirir experiencia en la creación Videojuegos y se desarrolló una herramienta para la realización de nuevos Videojuegos, con el objetivo de ahorrarle tiempo en implementación y estructura del juego, que significaría centrarse solamente en la programación específica del juego. Game Tools 2D (GT2D) es un framework desarrollado en C++ para la rápida implementación de videojuegos 2D que funciona como un motor de juegos estándar. Posee además un gestor de estados (máquina de estado) para la implementación de los niveles o pantallas del juego y un prototipo de estado genérico mediante el cual se acelera notablemente la integración final del proyecto.

El sonido y los efectos básicos como el paneo y cambio de frecuencia también son fáciles de manipular utilizando el gestor de sonidos que viene integrado con el núcleo de la herramienta. Posee un paquete de componentes de interfaz de usuario que es básico pero que irá incrementándose a medida que iteren las nuevas versiones; por el momento es muy sencillo crear botones, diálogos personalizados, imprimir textos completamente ajustables y personalizados, cuadros de textos, cuadros de chequeo, listas de elementos, barras de progreso, marcos, punteros animados y algunos otros que son comunes en el trabajo cotidiano.

El framework cuenta con un editor gráfico que permite pintar las colisiones directamente sobre los objetos y el núcleo de la herramienta interpreta esta información y la genera en tiempo de ejecución para obtener colisiones precisas y fáciles de editar. El gestor de animación se ha implementado para brindar el máximo rendimiento y la mayor facilidad, se puede incluso sustituir los fotogramas en tiempo real de manera parcial o total, todo de forma totalmente transparente al programador. Posee además, algunos editores auxiliares para hacer el trabajo más fácil y eficiente como son el editor de mapa y editor de animaciones.

La herramienta fue desarrollada por programadores con métodos de prueba y error, sin aplicar en ningún momento alguna metodología para desarrollar los artefactos ingenieriles correspondientes hasta las etapas de análisis y diseño del software. Al no contarse con esta información no se conoce cómo es que funciona internamente esta herramienta, desconociendo así sus funcionalidades provocada por la falta de documentación del sistema.

Posteriormente el equipo de desarrollo del Motor se dispersó, unido a esto la ausencia de alguna documentación del sistema, trajo como consecuencia la falta de conocimiento sobre el diseño y las funcionalidades de la herramienta, lo que produjo que los usuarios actuales del mismo no tuviesen ninguna guía que les indicara cuales eran las características de dicho sistema, por ende no ha podido cumplir su objetivo de ahorrarle tiempo al desarrollador. Los proyectos que hacen uso de esta herramienta están retrasados esperando la documentación de la misma.

De lo expuesto anteriormente se define como **problema científico**: La inexistencia de los artefactos ingenieriles del Motor GT2D.

La investigación tiene como **objeto de estudio**: Proceso de Ingeniería Inversa, teniendo como **campo de acción:** Proceso de Ingeniería Inversa para las herramientas de desarrollo de Videojuegos.

Siendo el **objetivo general** de esta investigación: Generar los principales artefactos ingenieriles correspondientes a la etapa Análisis y Diseño del Motor GT2D, teniendo como **idea a defender** de este trabajo sea: Si se realiza la Ingeniería Inversa al Motor GT2D, se logrará obtener los artefactos ingenieriles de la herramienta, que tributará al correcto uso de la herramienta.

Para dar cumplimiento al objetivo general y se plantean las siguientes **tareas investigativas:**

1. Descripción del proceso de la Ingeniería Inversa para aplicarlo al Motor.

2. Identificación de herramientas existentes para el desarrollo de los Videojuegos 2D.

3. Caracterización del Motor GT2D para aplicar el proceso de la Ingeniería Inversa.

4. Selección de Metodología para realizar la Ingeniería Inversa del Motor de Juego.

5. Restructuración del código fuente del Motor como punto de partida para el proceso de Ingeniería Inversa.

6. Definición de los artefactos ingenieriles correspondientes a las etapas de Análisis y Diseño de la metodología seleccionada.

7. Validación de la propuesta mediante las Matrices de Trazabilidad.

Para la realización de la investigación se utilizarán varios **métodos y técnicas en la búsqueda y procesamiento de la información** como son:

A nivel teórico:

- **Análisis histórico–lógico:** Para conocer, con mayor profundidad los antecedentes y las tendencias actuales referidas al desarrollo de los Videojuegos y el proceso de Ingeniería Inversa en aplicaciones de este tipo, conociendo así la trayectoria histórica que tiene a través de su origen y las herramientas para el desarrollo del tema.

A nivel empírico:

- **Entrevista:** Para obtener información a partir de conversaciones planificadas con los encargados de la realización del Motor y con los que tienen conocimientos acerca de cómo aplicar una Ingeniería Inversa.

Este trabajo está estructurado de la siguiente forma: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas, bibliografía consultada y anexos.

A continuación se describen los elementos más importantes identificados durante la confección del cuerpo del documento de la presente investigación:

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

En este capítulo se pretende plasmar los conceptos fundamentales sobre Ingeniería Inversa, se realizará un estudio sobre las herramientas que existen similares al Motor para así poder compararla en cuanto a funcionalidad. Se realizará además un estudio sobre las metodologías para realizar software para así seleccionar la más conveniente para el presente trabajo.

CAPÍTULO 2: ANÁLISIS DEL SISTEMA.

En este capítulo según lo que expone la etapa de análisis en la metodología de Open Up se extraerán los requisitos funcionales y no funcionales, así como los Casos de Uso con las respectivas descripciones a los Casos de Usos que son arquitectónicamente significativos.

CAPÍTULO 3: DISEÑO DEL SISTEMA.

En esta etapa de diseño según lo que propone la metodología de Open Up se extraerán los Diagramas de Secuencias y el Diagrama de Clase organizados por paquetes según su funcionalidad, además de que se realizarán las matrices de Trazabilidad con su correspondiente árbol de trazabilidad como una forma de validación a la información generada.

Capítulo 1

Fundamentación Teórica

Introducción

En este capítulo se estudiará el proceso de la Ingeniería Inversa para buscar la mejor forma de aplicarla al código del Motor, se abundarán definiciones importantes que se deben de tener en cuenta para entender de lo que se está hablando en el documento y se realizará un estudio de algunas de las Metodologías ágiles para así seleccionar la más óptima para la presente tesis, además de elegir lenguajes y herramientas para el desarrollo de la tesis.

1.1- Ingeniería Inversa.

1.1.1- Definición.

La Ingeniería Inversa (IV) del software es el proceso consistente en analizar un programa, en un esfuerzo por crear una representación del mismo con un nivel de abstracción más elevado que el código fuente, además de que es el proceso de recuperación de diseño del sistema. Las herramientas de IV extraen información acerca de los datos, arquitectura y diseño de procedimientos de un programa ya existente. Ver anexos figura 1.

Los objetivos que persigue la IV son de menor a mayor nivel de abstracción, como son la representación de diseños de procedimientos, la obtención de la información de las estructuras de datos que contienen los sistemas, la extracción de los modelos de flujos de datos y de control y por último la extracción de los modelos de entidades y de relaciones.

Según va aumentando la abstracción va aumentando la complejidad del trabajo, así como la necesidad de comprensión de la aplicación. [1]

1.1.2- Relación entre reingeniería vs Ingeniería Inversa.

Reingeniería:

Es la evaluación y la modificación de un sistema objeto que se va a reconstruir de una forma diferente; este proceso se acompaña de una combinación de subprocesos tales como la Ingeniería Inversa, la reestructuración, la redocumentación y la ingeniería avanzada. Ver anexos figura 18. [2]

Ingeniería Inversa:

Es utilizada en el campo de reingeniería para generar nueva información obtenida del mismo software, así como abstracciones sintetizadas o generadas por diferentes formas de datos.

También es utilizada para pasar de una codificación de entrada o de una traslación del código fuente, a otro lenguaje u otra herramienta que preserve las funcionalidades del sistema.

1.1.3- Aplicación de la Ingeniería Inversa en la Informática.

En el campo de la informática, la Ingeniería Inversa está empezando a cobrar una gran importancia por la necesidad de tener que mantener o improvisar software y sistemas de información.

A la vez la Ingeniería Inversa proporciona un significado sistemático acerca del diseño de procesos de un sistema, tanto lógico como físico. Fomenta en los estudiantes el interés hacia el diseño de aplicaciones más complejas, explora nuevas rutas para la producción y evolución de nuevos productos y apoya el rediseño del producto. [1]

1.1.4- Pasos propuestos para realizar una Ingeniería Inversa.

La IV hasta el momento, no propone ninguna metodología, pero si se han utilizado unas series de pasos para la realización de la misma [2]:

Paso 1:

Establecer una descripción clara de los procesos, desarrollar una hipótesis acerca de las características y principios de los sistemas y generar una hipótesis refinada de una descomposición funcional del sistema.

Paso 2:

Adquirir experiencia del sistema por medio de la realización de la disección del mismo, para discutir y comparar, estableciendo de esta forma predicciones, crear estructuras de funciones refinadas del sistema actual y transformar a ingeniería las especificaciones y sus métricas de calidad escogiendo todas aquellas aplicaciones, funciones y subsistemas que estarán involucrados dentro del mismo.

Paso 3:

Modelar y analizar las funciones del sistema a nivel de componentes y subsistemas, identificar los principios básicos actuales del sistema y crear un balance entre las relaciones existentes dentro del sistema.

1.2. Herramientas existentes similares al Motor GT2D.

Antes de profundizar en las funcionalidades del Motor GT2D se estudiaron algunas de las herramientas que hasta el momento se encontraron en la web, para realizar una comparación con el Motor y así proponer nuevas funcionalidades, en el anexo 7 hay una tabla donde se hace esta comparación donde se evidencia las nuevas funcionalidades que se le pueden agregar al Motor.

Entre los Motores están **M.U.G.E.N [5], Construct [6], Game Maker [7] y RPG Maker XP [8]**, las cuales se describen a continuación:

RPG Maker XP

Basa su aspecto en dos apartados: edición de personajes, escenarios y desarrollo de la historia. Definiendo los personajes y objetos, así como su comportamiento en diferentes situaciones
Aspecto negativo: el sistema de combate por defecto es muy básico, aunque a cambio se pueden diseñar los ataques y magias que desee el usuario. Ver Anexo 8.

Algunos juegos creados: Yume Nikki - Secret of Gaya.

Game Maker

Presenta un editor de Sprites, que representan los objetos visibles del juego, incluye una interfaz de programación basada en su propio lenguaje (GML). Este lenguaje, al estar simplificado y orientado al mundo de los videojuegos, es mucho más fácil de aprender que otros usados a nivel profesional, como C++ o Java, y hace posible la creación de videojuegos de casi cualquier género. Dispone de dos modos de edición: uno más básico, que consiste en arrastrar y soltar acciones referidas a objetos, y otro más avanzado para quienes sepan programar y quieran hacer juegos más originales y trabajados. Ver Anexo 9.

Algunos juegos creados: Spelunky – Karoshi.

Construct

En este caso se colocan los objetos sobre el escenario y se le adjudican un comportamiento determinado, creador de videojuegos más visual, posee compatibilidad con librerías DirectX, los juegos corren más fluidos, así como la propia interfaz. Requiere algo más de esfuerzo aprender el lenguaje propio de la aplicación, aunque los resultados a la larga sean más vistosos. Ver Anexo 10.

Algunos juegos creados: Bit Fortress - Embryo

M.U.G.E.N

Este es un Videojuego de lucha con la posibilidad de añadir cuantos personajes se quieran. Ver Anexo 11.

Algunos juegos creados: Dragon Ball Z - Naruto

1.3. Caracterización de las Metodologías ágiles.

1.3.1- Definición de las metodologías ágiles.

Las metodologías ágiles son **iterativas**. No intentan minimizar los cambios, sino estar preparados para aceptarlos. Son **adaptativas** en lugar de repetibles. Priorizan a los **individuos** y a las **interacciones** por sobre los procesos. Incorporan **feedback** sobre el proceso. Priorizan la **colaboración** con el cliente. [9]

¿Cuándo elegir las metodologías ágiles?

Para elegir la metodología ágil se tienen que tener en cuenta varios aspectos fundamentales como son:

Que los requerimientos deben ser poco claros o altamente volátiles, el cliente entiende el proceso y está involucrado en el proyecto, se cuenta con profesionales capacitados y competentes, se tiene canales ricos de comunicación, el grupo de trabajo no es demasiado grande (~ 50) y se desea fomentar la mejora continua del proceso. [9]

1.3.2- Metodologías ágiles consultadas.

Para poder realizar el presente trabajo se realizaron estudios de las metodologías que se consideraron adecuadas para la elaboración de la Ingeniería Inversa.

Extreme Programming (XP)

Es la primera metodología ágil y la que dio conciencia al movimiento actual de metodologías ágiles. Se centra en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los programadores, y propiciando un buen clima de trabajo. Se basa en la realimentación continua entre el cliente y el equipo de desarrollo. Se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes y donde existe un alto riesgo técnico. El ciclo ideal de XP consiste de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. Se diferencia de las demás metodologías en que tiene un alto nivel de disciplina de las personas que participan en el proyecto. Constituye la práctica de programación, usualmente orientada a objetos y con un fuerte uso de patrones de diseño. [10]

Scrum

Define un proceso empírico, iterativo e incremental de desarrollo que intenta obtener ventajas respecto a los procesos definidos (cascada, espiral, prototipos, etc.) mediante la aceptación de las naturales caótica del desarrollo de software, y el riesgo a niveles aceptables. Método iterativo e incremental que enfatiza prácticas y valores de gestión de proyecto por sobre las demás disciplinas del desarrollo. Se definen algunos roles y artefactos que contribuyen a tener un proceso que maximiza el feedback para mitigar cualquier riesgo que pueda presentarse. Se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad. Su intención es la de maximizar la realimentación sobre el desarrollo pudiendo corregir problemas y mitigar riesgos de forma temprana. No propone el uso de ninguna práctica de desarrollo en particular; sin embargo, es habitual emplearlo como un framework ágil de administración de proyectos que puede ser combinado con cualquiera de las metodologías mencionadas. Suministra un marco de gestión basado en patrones organizacionales. [9]

Crystal

Esta metodología presenta un enfoque ágil, con gran énfasis en la comunicación, y con cierta tolerancia que la hace ideal en los casos en que sea inaplicable la disciplina requerida por XP. Se define como mucho énfasis en la comunicación y de forma muy liviana en relación a los entregables. Maneja iteraciones cortas con feedback frecuente por parte de los usuarios/clientes, minimizando de esta forma la necesidad de productos intermedios. La familia

Crystal dispone un código de color para marcar la complejidad de una metodología: cuanto más oscuro un color, más “pesado” es el método. El proceso se basa en una exploración refinada de los inconvenientes de los modelos clásicos. [9]

Open Up

Preserva la esencia del Proceso Unificado. Desarrollo iterativo e incremental, dirigido por Casos de Uso. Centrado en la Arquitectura. Sólo lo fundamental está incluido, sin dejar de ser completo y extensible (menos de 20 artefactos). Está pensado para proyectos pequeños. Ciclo de vida basado en Valor y Riesgo. Planificación a dos niveles. Integración Continua. Arquitectura y Diseño Evolutivo. Visión Compartida. Ver anexos figura 17. [13]

1.4- Selección de la Metodología.

De todas las metodologías estudiadas se escoge Open UP ya que conserva las características esenciales de RUP, que incluyen el desarrollo iterativo, casos de uso y escenarios de desarrollo de prácticas, la gestión de riesgos y enfoque centrado en la arquitectura. La mayoría de piezas opcionales de RUP han sido excluidas, y muchos elementos han sido fusionados. El resultado final es un proceso mucho más sencillo que sigue siendo fiel a los principios RUP [13].

Constituye un Proceso Unificado de desarrollo de corta duración, aplicado de manera iterativa e incremental dentro de un ciclo de vida estructurado en tres capas. Open UP adopta una pragmática y ágil filosofía centrado en el proceso colaborativo de desarrollo de software que puede ser aplicado a una gran variedad de proyectos en diferentes direcciones.

El Open UP es un proceso mínimo y suficiente, lo que significa que solo el contenido fundamental y necesario es incluido. Por lo tanto, no provee lineamientos para todos los elementos que se manejan en un proyecto, pero tiene los componentes básicos que pueden servir de base a procesos específicos. La mayoría de los elementos de Open UP están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Open UP en cada iteración del ciclo de vida, incrementa progresivamente los objetivos de las iteraciones anteriores añadiendo nuevas funcionalidades a las versiones estables del software

que se tiene hasta el momento. Dentro del ciclo de vida tiene 4 fases: Intercepción, Elaboración, Construcción y Transición [13].

Se propone Open UP por ser un proceso de desarrollo de software completo, ser extensible, además que fue diseñada para un equipo de desarrollo reducido, es ágil, ligera y proporciona una comprensión detallada del proyecto, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad. Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.

1.5- Características Fundamentales de Open UP.

Open Up como toda metodología propone varios artefactos los cuales se mencionarán a continuación [13]:

Dominio de Arquitectura:

En este dominio está presente el **Artefacto Cuaderno de Arquitectura**, este artefacto describe la razón de ser, las hipótesis, la explicación, y las implicaciones de las decisiones que se hicieron en la formación de la arquitectura.

Dominio de Desarrollo:

Contiene varios artefactos que a continuación se le hace mención:

Artefacto Aplicación: Este artefacto genera secuencias de comandos y otros archivos que se transforman en los sistemas ejecutables.

Artefacto: Construir es una versión operativa de un sistema o parte de un sistema que pueda servir para un subconjunto de las capacidades que se incluirá en el producto final.

Artefacto: Prueba para desarrolladores en este caso se valida un aspecto específico de un elemento de ejecución. Este artefacto cubre todos los pasos para validar un aspecto específico de un elemento de ejecución.

Artefacto: Diseño, este artefacto describe la realización de la funcionalidad del sistema necesario y sirve como una abstracción del código fuente. En él se describen los elementos que componen el sistema implementado.

Dominio Gestión de Proyectos:

Artefacto: Lista de Riesgos, este artefacto es una lista ordenada de los riesgos conocidos ordenados en orden de importancia y asociadas con la mitigación o acciones específicas de contingencia.

Artefacto: Trabajo Lista de artículos, este artefacto contiene una lista de todo el trabajo programado para ser realizado dentro del proyecto, así como el trabajo propuesto, que pudiera afectar el producto en los proyectos de este o futuros.

Artefacto: Plan de Iteración, los principales objetivos del plan de iteración son proporcionar el equipo con un lugar central para obtener información sobre los objetivos de la iteración, un plan detallado con las asignaciones de tareas y resultados de la evaluación.

Artefacto: Plan de proyecto, el propósito de este artefacto es proporcionar un documento central, donde cualquier miembro del equipo del proyecto puede encontrar la información sobre cómo el proyecto se llevará a cabo.

Dominio de Requisitos:

Artefacto: Glosario, este artefacto define los términos importantes utilizados por el proyecto.

Artefacto: Visión, este artefacto define la vista de los actores de la solución técnica a desarrollar.

Artefacto: Requisitos de todo el sistema, este artefacto captura los atributos de calidad y las limitaciones que tienen un ámbito de todo el sistema. También recoge los requisitos funcionales de todo el sistema.

Artefacto: Modelo de Casos de Uso, este artefacto captura un modelo de las funciones previstas y el medio ambiente del sistema y sirve como un contrato entre el cliente y el equipo.

Artefacto: Caso de Uso, este artefacto captura el comportamiento del sistema para producir un resultado observable de valor para aquellos que interactúan con el sistema.

Dominio de Prueba:

Artefacto: Caso de Prueba, este artefacto es la especificación de un conjunto de entradas de prueba, las condiciones de ejecución, y los resultados esperados que usted identifique a evaluar un aspecto particular de un escenario.

Artefacto: Secuencia de comandos de prueba, este artefacto contiene las instrucciones paso a paso que componen la prueba, lo que permite su ejecución.

Artefacto: registro de la prueba, el propósito de este artefacto es proporcionar la comprobación de que un conjunto de pruebas se ha ejecutado y proporcionar información que se relaciona con el éxito de las pruebas**.**

1.5.1- Conceptos fundamentales asociados a esta Metodología.

Requisitos.

Un requisito es una descripción de una condición o capacidad que debe cumplir un sistema, ya sea derivada de una necesidad de usuario identificada, o bien, estipulada en un contrato, estándar, especificación u otro documento formalmente impuesto al inicio del proceso [14].

Existen varias definiciones de requisito a nivel mundial, entre las cuales se pueden citar las siguientes:

- Condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo.
- Condición o capacidad que debe exhibir o poseer un sistema para satisfacer un contrato, estándar, especificación, u otra documentación formalmente impuesta.
- Un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema o una restricción de éste.

Clasificación de Requisitos.

Los requisitos del sistema de software se clasifican en funcionales, no funcionales o del dominio [14].

Requisitos Funcionales.

Los requisitos funcionales son los que definen las funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Es importante que se describa el ¿Qué? y no el ¿Cómo? se deben hacer esas transformaciones. Estos requisitos al tiempo que avanza el proyecto de software se convierten en los algoritmos, la lógica y gran parte del código del sistema.

La especificación de requisitos funcionales de un sistema debe ser completa para que todos los servicios solicitados por el usuario estén definidos, y ser consistente para no tener definiciones contradictorias [14].

Requisitos no Funcionales.

Los requisitos no funcionales son restricciones de los servicios o funciones ofrecidas por el sistema. Incluyen restricciones de tiempo, sobre el proceso de desarrollo y estándares. Se refieren a las propiedades emergentes del sistema como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. Definen además las restricciones del sistema como la capacidad de los dispositivos de entrada y salida, y las representaciones de datos que se utilizan en las interfaces del sistema. Pueden especificar el rendimiento del sistema, la protección, la disponibilidad y otras propiedades emergente [14].

Surgen de las necesidades del usuario, debido a las restricciones en el presupuesto, a las políticas de organización, a la necesidad de interoperabilidad con otros sistemas software o hardware, o a factores externos como regulaciones de seguridad o legislaciones sobre privacidad.

Este tipo de requisitos tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), Hardware, interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, ambiente. Además dependen del volumen de operaciones, cantidad de usuarios y de las aplicaciones que deben de estar en uso cuando se implemente el sistema.

Requisitos del Dominio.

Son requisitos que provienen del dominio de aplicación del sistema y que reflejan las características de ese dominio. Se derivan del dominio de aplicación más que de las necesidades específicas de los usuarios. Incluyen terminología especializada del dominio o referencias a conceptos del dominio. Pueden ser requisitos funcionales nuevos, restringir los

existentes o establecer cómo se deben ejecutar cálculos particulares y además pueden ser requisitos no funcionales.

Los requisitos del dominio son importantes debido a que a menudo reflejan los fundamentos del dominio de aplicación. Si estos requisitos no se satisfacen, es imposible hacer que el sistema trabaje de forma satisfactoria [14].

Casos de Uso.

Consiste en una técnica para especificar el comportamiento de un sistema y permiten describir la posible secuencia de interacciones entre este y uno o más actores, es una descripción de un conjunto de escenarios, cada uno de ellos comenzado con un evento inicial desde un actor hacia el sistema. Casi la totalidad de los requisitos se pueden expresar con casos de uso [15].

Se basan en escenarios para la obtención de requisitos. Actualmente, se han convertido en una característica fundamental de la notación UML (Lenguaje unificado de modelado), que se utiliza para describir modelos de sistemas orientados a objetos.

1.6- Lenguaje y Herramientas utilizadas para el modelado.

Lenguaje Unificado de Modelado (UML).

Es un Lenguaje Unificado de Modelado para la especificación, visualización, construcción y documentación de los artefactos de un proceso de desarrollo de software. Fue originalmente concebido por la industria de la tecnología y sistemas de información. Constituyen un lenguaje para especificar y no para describir métodos o procesos. Existen 13 tipos diferentes de diagramas que se pueden categorizar en: Diagramas de Estructura, Diagramas de Comportamiento y Diagramas de Iteración [16].

Se escogió este lenguaje porque el mismo integra todos los elementos que propone la metodología RUP para cubrir el ciclo de vida de un proyecto y como Open Up es una metodología que contiene los artefactos ingenieriles más significativos de RUP este lenguaje es el más ideal para el modelado del sistema.

Herramienta de Modelado: Visual Paradigm.

Visual Paradigm es una Herramienta Case que apoya actividades que tienen lugar a lo largo de todo el ciclo de vida, incluyendo actividades como la Gestión de Proyectos y la Estimación [17].

Es una herramienta UML profesional que soporta el ciclo de vida completo de desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Proporciona abundantes tutoriales, demostraciones interactivas y proyectos UML, lo que para la elaboración de esta tesis se hace muy adecuada debido a que el trabajo se hace mucho más simple.

OSRMT.

Esta herramienta permite crear las dependencias entre los elementos a tracear donde se especifican las dependencias desde y hacia el elemento. Esta herramienta también permite visualizar gráficamente el impacto de la trazabilidad y se puede generar la Matriz de Trazabilidad que la misma es mantenida durante todo el ciclo de vida del proyecto manteniendo información sobre los requisitos, sus atributos y dependencias. Por todo lo antes expuesto se considera esta herramienta como la más optima para el desarrollo del presente trabajo.

Conclusiones parciales

Luego de haber culminado el presente capítulo se estudiaron algunas de las metodologías ágiles como son Scrum, XP, Crystal y Open UP, escogiendo esta última como la metodología la más adecuada para aplicarla en el proceso de la Ingeniería Inversa, por su característica de ser aplicable a proyectos pequeños y que genera menos de 20 artefactos, luego de hacer la selección de metodología se plasmaron las definiciones que se creyeron importantes para desarrollar la presente investigación, exponiéndose el porqué del uso del lenguaje UML y de las herramientas Visual Parading y OSRMT.

Capítulo 2:

Análisis del Sistema.

Introducción

El siguiente capítulo se encuentra enfocado al tratamiento de Requisitos y Casos Uso seleccionando de estos los críticos, para así realizar la descripción de ellos y dar paso al modelado de los diagramas.

Se enfocará además, en la representación del modelo conceptual explicando el funcionamiento de los conceptos tratados en el mismo, así como la relación entre ellos

2.1- Modelo Conceptual

Un modelo conceptual no es más que "toda construcción hecha con información de una situación, idea o fenómeno real, en la cual hayamos resaltado ciertos aspectos de interés al tiempo que hemos ocultado todo lo demás". [14]:

Juego: El juego es una actividad que se utiliza para la diversión y el disfrute de los participantes, en muchas ocasiones, incluso como herramienta educativa.

Imagen: Una imagen óptica es una figura formada por el conjunto de puntos donde convergen los rayos que provienen de fuentes puntuales del objeto tras su interacción con el sistema óptico.

Superficie: Aspecto externo de algo. Magnitud que expresa la extensión de un cuerpo en dos dimensiones, largo y ancho.

Animación: Procedimiento de diseñar los movimientos de los personajes o de los objetos y elementos.

Colisión: Choque de dos cuerpos.

Entidad: Lo que constituye la esencia o la forma de una cosa.

Cámara: Aparato portátil que registra imágenes y sonidos y los reproduce.

Luz: Agente físico que hace visibles los objetos.

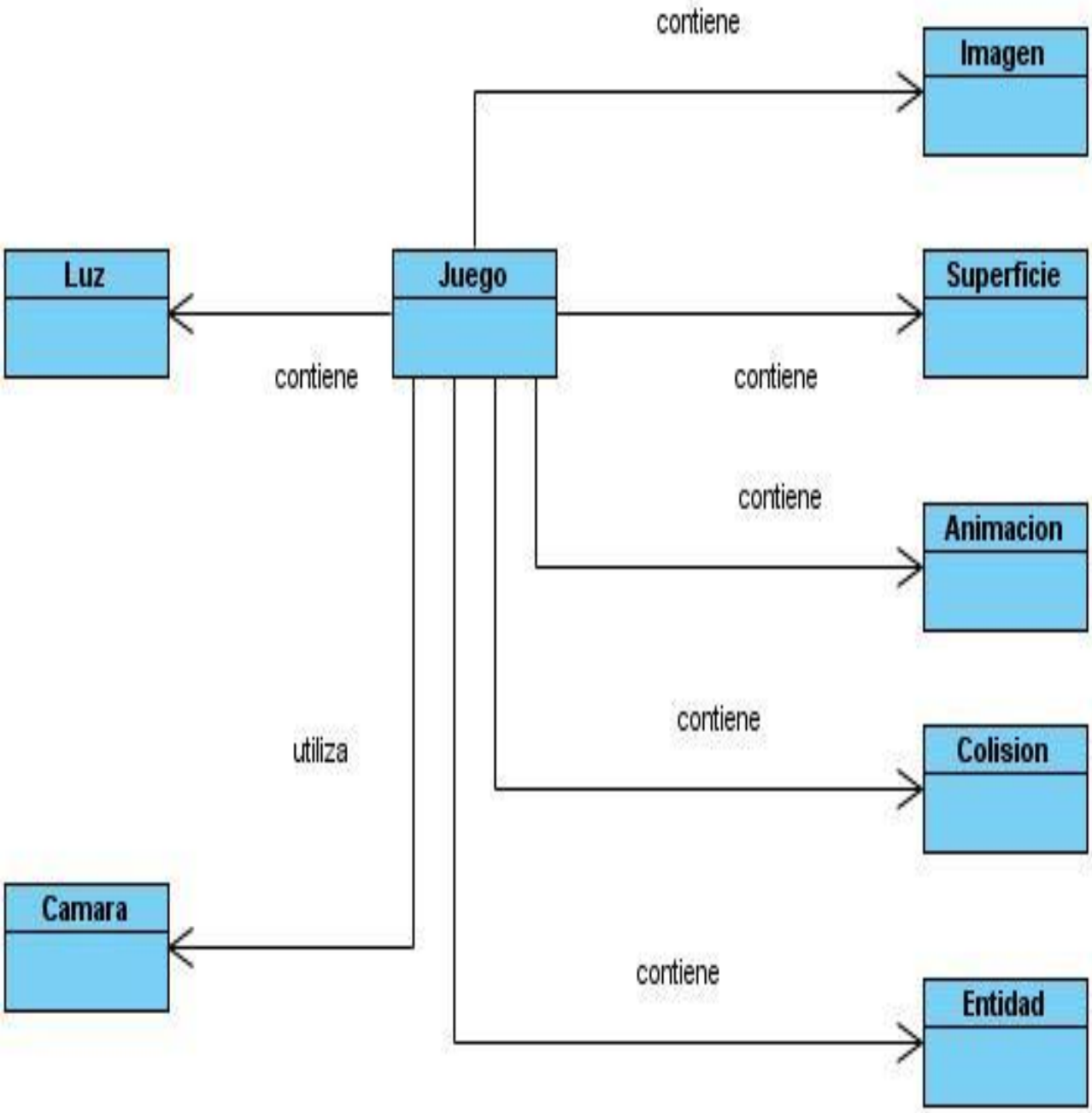


Figura 2.1 Modelo Conceptual.

2.2- Proceso de Captura de Requisitos.

En el estudio del sistema se identificaron 12 requisitos funcionales y 3 requisitos no funcionales:

Requisitos Funcionales	Requisitos No Funcionales
RF1: Cargar juego.	**RnF:** Restricciones de diseño.
RF2: Cargar Superficie.	**RnF:** Interfaz.
RF3: Obtener Colisiones.	**RnF:** Usabilidad.
RF4: Gestionar Animación.	**RnF:** Soporte.
RF5: Gestionar Entidades.	**RnF:** Requisitos Legales.
RF6: Cargar Luces.	
RF7: Cargar Imagen.	
RF8 Posicionar Cámara.	
RF9: Cargar Animación.	
RF10: Cargar Entidad.	
RF11: Gestionar Superficie.	
RF12: Gestionar Imagen.	

2.2.1- Requisitos Funcionales del Motor GT2D.

RF1 Cargar juego.

RF1.1 El sistema debe ser capaz de retornar la cantidad de frames por segundo que tiene actualmente.

RF1.2 El sistema debe ser capaz de obtener las características del punto de visión.

RF1.3 El sistema debe ser capaz de obtener el punto del objeto render donde ha sido creado.

RF1.4 El sistema debe ser capar se obtener todos los eventos realizado del mouse y el teclado.

RF1.5 El sistema debe ser capaz de manejar los eventos del joystick.

RF1.6 El sistema debe ser capaz de manejar los eventos del teclado y del mouse.

RF2 Cargar Superficie.

RF2.1 El sistema debe ser capaz de obtener las características fundamentales de la superficie.

RF2.5 El sistema debe ser capaz de identificar si la superficie ha sido cargada

RF2.5 El sistema debe ser capaz de identificar si la superficie tiene una cuadricula asignada.

RF3 Obtener Colisiones.

RF3.1 El sistema debe posibilitar detectar la ubicación de los personajes en el mapa.

RF3.2 El sistema debe permitir detectar las posibles colisiones del juego.

RF3.3 El sistema debe posibilitar detectar el uso del mouse.

RF3.3 El sistema debe posibilitar detectar el uso del teclado.

RF4 Gestionar Animación.

RF4.1 El sistema debe ser capaz de adicionar una animación.

RF4.2 El sistema debe ser capaz de eliminar una animación.

RF5 Gestionar Entidades.

RF5.1 El sistema debe ser capaz de adicionar una entidad

RF5.2 El sistema debe ser capaz de eliminar una entidad.

RF6 Cargar Luces.

RF6.1 El sistema debe ser capaz de identificar si la luz esta activa.

RF6.2 El sistema debe ser capaz de retornar la posición de la luz.

RF6.3 El sistema debe ser capaz de retornar los componentes los cuales definen la dirección de la luz.

RF7 Cargar Imagen.

RF7.1 El sistema debe ser capaz de obtener las características fundamentales de la imagen en pixeles.

RF7.2 El sistema debe ser capaz de retornar la cantidad de byte por pixeles de la imagen.

RF7.3 El sistema debe ser capaz de retornar el formato de la imagen.

RF7.4 El sistema debe ser capaz de retornar la extensión que tiene una imagen cuando está almacenada en un archivo.

RF7.5 El sistema debe ser capaz de retornar el punto de la memoria el cual forma la imagen.

RF8 Posicionar Cámara.

RF8.1 El sistema debe ser capaz de retornar la posición de la cámara.

RF8.2 El sistema debe ser capaz de retornar el zoom de la cámara.

RF8.3 El sistema debe ser capaz de retornar el ángulo de la cámara.

RF8.4 El sistema debe ser capaz de retornar los componentes que está mirando la cámara.

RF8.5 El sistema debe ser capaz de retornar los componentes que están por encima de la cámara.

RF8.6 El sistema debe ser capaz de retornar los componentes que están a la derecha de la cámara.

FR9 Cargar Animación.

RF9.1 El sistema debe ser capaz de retornar el total de frames de la animación.

RF9.2 El sistema debe ser capaz de retornar el total de secuencias de la animación.

RF9.3 El sistema debe ser capaz de retornar la ubicación de cualquier objeto en esta animación.

RF9.4 El sistema debe ser capaz de retornar el nombre de la secuencia que ha sido entrada por parámetro.

RF9.5 El sistema debe ser capaz de obtener las características fundamentas de esta secuencia que ha sido entrada por parámetro.

FR10 Cargar Entidad.

RF10.1 El sistema debe ser capaz de retornar el punto de una superficie si este tiene una asignada.

RF10.2 El sistema debe ser capaz de retornar el punto de una animación si este tiene una asignada.

RF10.3 El sistema debe ser capaz de identificar si la entidad está siendo mostrada.

RF10.4 El sistema debe ser capaz de obtener las características fundamentales de la entidad.

RF10.6 El sistema debe ser capaz de retornar el número de repeticiones que tiene que hacer una animación.

RF10.7 El sistema debe ser capaz de retornar el número de secuencia que ha sido asignada a la animación.

RF11 Gestionar Superficie.

RF 11.1 El sistema debe ser capaz de adicionar una superficie.

RF 11.2 El sistema debe ser capaz de eliminar una superficie.

RF 12 Gestionar Imagen.

RF 12.1 El sistema debe ser capaz de adicionar una imagen.

RF 12.2 El sistema debe ser capaz de eliminar una imagen.

2.2.2- Requisitos No Funcionales del Motor GT2D.

Restricciones de diseño:

RnF1: Sistema Operativo Windows 2003 o superior.

RnF2: Se debe tener instalado el Visual Studio 2005 o una versión más avanzada.

RnF3: Lenguaje de programación C++.

Interfaz:

RnF1: Microprocesador Intel Pentium 2, 5 GB libre en disco duro y Memoria RAM 512 MB, o superior.

Usabilidad:

RnF1: El usuario debe tener conocimientos sobre programación y de los Videojuegos.

Soporte:

RnF1: Utiliza las bibliotecas, DevIL (para el tratamiento de imágenes), IndieLib (como gestor gráfico y tratamiento de entidades), SDL (para los eventos), DirectX (solo para la aceleración por hardware y render sobre Windows), OpenGL (para render multiplataforma), Box2D (para la dinámica de los objetos).

Requisitos Legales:

RnF1: Se regirá por la norma ISO 9000.

2.3. Descripción y Modelado del sistema.

2.3.1- Actores del Sistema.

Actor	Descripción
Sistema Externo.	El actor es el que interactúa con las funcionalidades de la herramienta.

Tabla 1: Descripción de los Actores del Sistema.

2.3.2- Casos de Uso del Sistema.

Los CU proporcionan uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico utilizando un lenguaje más cercano al usuario final [15].

Para la representación de los diagramas de secuencia y la descripción de los CU, se realizaron solamente a los CU arquitectónicamente significativos o sea a los críticos, estos son el CU1, CU2, CU3, CU4, CU5, CU7, CU9, CU10, CU11, CU12, estos CU son críticos porque sin ellos la herramientas no pudiera funcionar, que no es el caso de los CU6 y CU8.

Código	Nombre	Actor	Referencia
CU1	Cargar Juego.	Sistema Externo	RF1, RF1.1, RF1.2, RF1.3, RF1.4, RF1.5
CU2	Cargar Superficie.	Sistema Externo	RF2, RF2.1, RF2.2, RF2.3, RF2.4, RF2.5
CU3	Obtener Colisión.	Sistema Externo	RF3, RF3.1, RF3.2, RF3.3
CU4	Gestionar Animación	Sistema Externo	RF4, RF4.1, RF4.2
CU5	Gestionar Entidad	Sistema Externo	RF5, RF5.1, RF5.2
CU6	Cargar Luces.	Sistema Externo	RF6, RF6.1, RF6.2, RF6.3
CU7	Cargar Imagen.	Sistema Externo	RF7, RF7.1, RF7.2, RF7.3, RF7.4, RF7.5, RF7.6, RF7.7
CU8	Posicionar Cámara.	Sistema Externo	RF8, RF8.1, RF8.2, RF8.3, RF8.4, RF8.5, RF8.6
CU9	Cargar Animación	Sistema Externo	RF9, RF9.1, RF9.2, RF9.3, RF9.4, RF9.5, RF9.6
CU10	Cargar Entidad	Sistema Externo	RF10, RF10.1, RF10.2, RF10.3, RF10.4, RF10.5, RF10.6, RF10.7

CU11	Gestionar Superficie	Sistema Externo	RF11, RF11.1, RF11.2
CU12	Gestionar Imagen	Sistema Externo	RF12, RF12.1, RF12.2

Tabla 2: Descripción de los CU del Sistema.

2.3.3- Diagrama de Casos de Uso.

La representación de los CU en diagramas se realiza con el objetivo de modelar de una forma simple y efectiva los requisitos del sistema desde el punto de vista del usuario. El diagrama de CU constituye una visión general que se ha identificado para satisfacer los requerimientos funcionales del sistema [15].

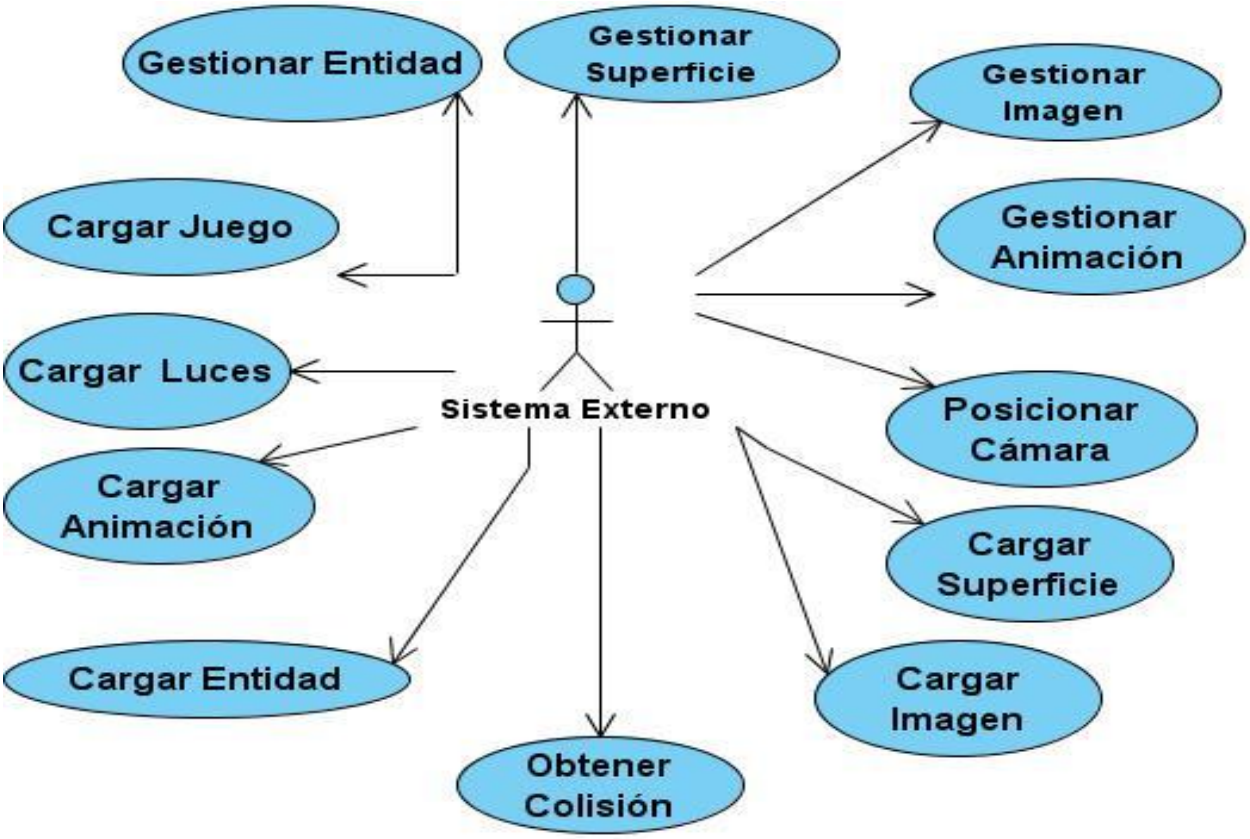


Figura 2.2 Diagrama del Caso de Uso del sistema.

2.3.4- Descripción de Casos de Uso.

Los Casos de Uso (CU) pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Un CU contiene una descripción textual de todas las maneras que los actores previstos podrían trabajar con el software o el sistema. Los CU no describen

ninguna funcionalidad interna del sistema, ni explican cómo se implementará. Simplemente muestran los pasos que el actor sigue para realizar una tarea [15].

CU1 Cargar Juego.

Caso de Uso:	Cargar Juego.
Actor:	Sistema Externo.
Descripción:	Permite al sistema tener las configuraciones básicas a la hora que se carga el juego.
Prioridad	Crítico
Referencia:	RF1, RF1.1, RF1.2, RF1.3, RF1.4, RF1.5
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se ajustaron las condiciones básicas para poder iniciar el juego.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1. El CU se inicia cuando el Sistema Externo solicita comenzar el juego.	1.1 Almacena la cantidad de frames por segundo que tiene actualmente. 1.2 Almacena las posiciones de los puntos de visión. 1.3 Almacena el tamaño del punto de visión. 1.4 Se ajusta la pantalla del juego. 1.5 Verifica la posición del mouse, para así saber si se quiere salir del juego o si no. 1.6 Almacena los eventos que recibe del teclado. 1.7 Maneja los eventos del joystick. 1.8 Maneja los eventos del teclado y mouse. 1.9 Termina el CU.

Tabla 3: Descripción del CU Cargar Juego.

CU2: Cargar Superficie.

Caso de Uso:	Cargar Superficie.
Actor:	Sistema Externo.
Descripción:	Permite almacenar toda la información necesaria sobre la superficie que está controlando en ese momento el juego.
Prioridad	Crítico
Referencia:	RF2, RF2.1, RF2.2, RF2.3, RF2.4, RF2.5
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se almacenó la información sobre la superficie en cuestión.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1. El CU se inicia cuando el Sistema Externo solicita configurar la superficie entrando por parámetro la secuencia que corresponde a esta superficie.	1.1 Almacena el total de frames de la animación. 1.2 Almacena el total de secuencias de la animación. 1.3 Obtiene la ubicación de cualquier objeto en la superficie. 1.4 Obtiene el nombre de la secuencia que ha sido entrada por parámetro. 1.5 Almacena el ancho máximo de la secuencia. 1.6 Obtiene el largo máximo de la secuencia. 1.7 Termina el CU.

Tabla 4: Descripción del CU Cargar Superficie.

CU3: Obtener Colisión.

Caso de Uso:	Obtener Colisión.
Actor:	Sistema Externo.

Descripción:	Permite calcular las posibles colisiones en las que pueda estar sometida una entidad.
Prioridad	Crítico
Referencia:	RF3, RF3.1, RF3.2, RF3.3
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se almacenó la información de las colisiones que ocurrieron en el juego.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1. El CU se inicia cuando el Sistema Externo solicita manejar las.	1.1 Detecta la ubicación de los personajes en el mapa. 1.2 Detecta las posibles colisiones del juego. 1.3 Detecta el uso del mouse. 1.4 Detectar el uso del teclado. 1.5 Termina el CU.

Tabla 5: Descripción del CU Obtener Colisiones.

CU4: Gestionar Animación.

Caso de Uso:	Gestionar Animación.
Actor:	Sistema Externo.
Descripción:	El caso de uso se inicia cuando el Usuario en algún momento del juego necesita adicionar o eliminar animaciones.
Prioridad	Crítico
Referencia:	RF4, RF4.1, RF4.2
Precondiciones:	La aplicación debe estar ejecutándose.

Poscondiciones:	Se controla todas las animaciones del Motor.
Flujo normal de eventos	
Sección "Adicionar Animación"	
Acción del Actor	**Respuesta del Sistema**
1. El Sistema Externo solicita adicionar una Animación pasándole por parámetro la imagen y la superficie.	1.1 Se envían los datos que se quieren adicionar. 1.2 Actualiza la imagen de la animación anterior por la nueva. 1.3 Actualiza la cantidad de frames que tiene esta animación. 1.4 Vuelve a enviar los datos actualizados de la imagen. 1.5 Crea la superficie actualizada pasada por parámetro. 1.6 Adiciona la Superficie creada. 1.7 Cambia la superficie que tenía antes por la que actualizó anteriormente. 1.8 Envía las actualizaciones. 1.9 Elimina la imagen anterior por la nueva. 1.10 Envía los datos de las imágenes que utilizan las animaciones actualizadas. 1.11 Actualiza la lista de animaciones con la nueva animación. 1.12 Termina el CU.
Sección "Eliminar Animación"	

Acción del Actor	Respuesta del Sistema
2. El Sistema Externo solicita eliminar una animación pasándole la animación que quiere eliminar por parámetro.	2.1 Envía un mensaje de si se quiere eliminar la animación y espera respuesta. 2.2 Envía los datos de la animación que quiere eliminar. 2.3 Busca en la lista de animaciones a la que se refiere el usuario. 2.4 Envía información de la animación. 2.5 Obtiene las imágenes a la que corresponde esa animación. 2.6 Elimina todas las imágenes de esa animación. 2.7 Obtiene la superficie que corresponde con la animación. 2.8 Elimina todas las superficies de la animación. 2.9 Libera todas las listas de secuencias que posee de esa animación. 2.10 Termina el CU.
Flujo alterno de los eventos de la sección "Eliminar Animación":	
	2.1 Si la respuesta es falsa se termina el CU.

Tabla 6: Descripción del CU Gestionar Animación.

CU5: Gestionar Entidad.

Caso de Uso:	Gestionar Entidad.
Actor:	Sistema Externo.
Descripción:	El caso de uso se inicia cuando el Usuario en algún momento del juego necesita adicionar o eliminar una entidad.

Prioridad	Crítico
Referencia:	RF5, RF5.1, RF5.2
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las entidades del Motor.

Flujo normal de eventos	
Sección "Adicionar Entidad"	
Acción del Actor	**Respuesta del Sistema**
3. El Sistema Externo solicita adicionar una entidad pasándole por parámetro la entidad que quiere adicionar.	2.1 Se envía mensaje para adicionar la entidad y espera respuesta. 2.2 Envía los datos de la entidad. 2.3 Adiciona la entidad. 2.4 Actualiza la lista de entidades. 2.5 Envía mensaje de confirmación. 2.6 Termina el CU.
Sección "Eliminar Entidad"	
Acción del Actor	**Respuesta del Sistema**
4. El Sistema Externo solicita eliminar una entidad pasándole por parámetro la entidad que quiere eliminar.	3.1 Se envía mensaje para eliminar la entidad. 3.2 Se envía datos de la entidad que se quiere eliminar. 3.3 Busca en la lista de entidades. 3.4 Elimina la entidad si fue encontrada. 3.5 Termina el CU.
Flujo alterno de los eventos de la sección "Eliminar Entidad":	
	2.1 Si la respuesta es falsa se termina el CU. 3.4 Envía mensaje de error.

Tabla 7: Descripción del CU Gestionar Entidad.

CU7: Cargar Imagen.

Caso de Uso:	Cargar Imagen.
Actor:	Sistema Externo.
Descripción:	Clase que gestiona las Imágenes del Motor.
Prioridad	Crítico
Referencia:	RF7, RF7.1, RF7.2, RF7.3, RF7.4, RF7.5, RF7.6, RF7.7
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las imágenes del Motor.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1. El CU se inicia cuando el Sistema Externo solicita trabajar con las imágenes del Motor.	1.1 Almacena el ancho de la imagen en pixeles. 1.2 Almacena el largo de la imagen en pixeles. 1.3 Almacena la cantidad de byte por pixeles de la imagen. 1.4 Obtiene el formato de la imagen, los formatos permitidos son “jpg” y “png”. 1.5 Obtiene la extensión que tiene una imagen cuando está almacenada en un archivo. 1.6 Obtiene el nombre de la imagen. 1.7 Termina el CU.

Tabla 8: Descripción del CU Cargar Imagen.

CU9: Cargar Animación.

Caso de Uso:	Cargar Animación.
Actor:	Sistema Externo.

Descripción:	Permite cagar una animación con los requisitos que ella necesita.
Prioridad	Crítico
Referencia:	RF9, RF9.1, RF9.2, RF9.3, RF9.4, RF9.5, RF9.6
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las animaciones del Motor.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
3. El CU se inicia cuando el Sistema Externo solicita manejar las animaciones del Motor pasándole por parámetro la secuencia que corresponde a la animación.	1.1 Almacena el total de frames de la animación. 1.2 Almacena el total de secuencias de la animación. 1.3 Obtiene ubicación de cualquier objeto. 1.4 Obtiene el nombre de la secuencia que ha sido entrada por parámetro. 1.5 Almacena ancho máximo de la secuencia. 1.6 Almacena largo máximo de la secuencia. 1.7 Termina el CU.

Tabla 9: Descripción del CU Cargar Animación.

CU10: Cargar Entidad.

Caso de Uso:	Cargar Entidad.
Actor:	Sistema Externo.
Descripción:	CU que gestiona las entidades 2D y 3D.
Prioridad	Crítico
Referencia:	RF10, RF10.1, RF10.2, RF10.3, RF10.4, RF10.5, RF10.6, RF10.7.

Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las entidades del Motor.
Flujo normal de eventos	
Acción del Actor	**Respuesta del Sistema**
1. El CU se inicia cuando el Sistema Externo solicita cargar una entidad 2D o 3D del Motor.	1.1 Almacena el punto de una superficie si este tiene una asignada. 1.2 Almacena el punto de una animación si este tiene una asignada. 1.3 Muestra Verdadero o Falso si la entidad está siendo mostrada. 1.4 Almacena la posición de la entidad. 1.5 Almacena el ángulo de la entidad. 1.6 Almacena la escala de la entidad. 1.7 Obtiene el número de repeticiones que tiene que hacer una animación. 1.8 Termina el CU.

Tabla 10: Descripción del CU Cargar Entidad.

CU11: Gestionar Superficie.

Caso de Uso:	Gestionar Superficie.
Actor:	Sistema Externo.
Descripción:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita adicionar o eliminar superficie.
Prioridad	Crítico
Referencia:	RF11, RF11.1, RF11.2
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las superficies del Motor.

Flujo normal de eventos	
Sección "Adicionar Superficie"	
Acción del Actor	**Respuesta del Sistema**
1. El Sistema Externo solicita adicionar una superficie pasándole por parámetro: la superficie que se quiere adicionar, el nombre de la superficie, el tipo.	1.1 Carga la imagen que le corresponde a esa superficie. 1.2 Crea la superficie con las imágenes que le corresponden. 1.3 Actualiza la lista de superficies. 1.4 Elimina las imágenes anteriores. 1.5 Termina el CU.
Sección "Eliminar Superficie"	
Acción del Actor	**Respuesta del Sistema**
2. El Sistema Externo solicita eliminar una superficie pasándole la superficie que quiere eliminar por parámetro.	2.1 Recorre la lista de superficie buscando la entrada por parámetro. 2.2 Libera todas las texturas que pertenecen a esa superficie. 2.3 Libera los vértices de la memoria. 2.4 Termina el CU.
Flujo alterno de los eventos de la sección "Eliminar Superficie":	
	2.1 Si la respuesta es falsa se termina el CU.

Tabla 11: Descripción del CU Gestionar Superficie.

CU12: Gestionar Imagen.

Caso de Uso:	Gestionar Imagen.
Actor:	Sistema Externo.

Descripción:	El caso de uso se inicia cuando el Sistema en algún momento del juego necesita adicionar o eliminar imágenes.
Prioridad	Crítico
Referencia:	RF12, RF12.1, RF12.2
Precondiciones:	La aplicación debe estar ejecutándose.
Poscondiciones:	Se controla todas las imágenes del Motor.

Flujo normal de eventos	
Sección "Adicionar Imagen"	
Acción del Actor	**Respuesta del Sistema**
1. El Sistema Externo solicita adicionar una imagen pasándole por parámetro: la imagen, el ancho, el largo, el formato y el nombre.	1.1 Obtiene y chequea la extensión de la imagen que se quiere adicionar. 1.2 Carga la imagen que se quiere adicionar. 1.3 Cambia el id de la imagen. 1.4 Obtiene todos los objetos que pertenecen a la imagen. 1.5 Crea la imagen con los atributos que le pertenecen. 1.6 Adiciona la imagen a la lista de imágenes 1.7 Termina el CU.
Sección "Eliminar Imagen"	
Acción del Actor	**Respuesta del Sistema**
2. El Sistema Externo solicita eliminar una imagen pasándola la que quiere eliminar por parámetro.	2.1 Busca en la lista de imágenes la que pasaron por parámetro. 2.2 Libera los objetos k pertenece a esta

	imagen. 2.3 Elimina de la lista la imagen. 2.4 Libera la imagen. 2.5 Libera la extensión de la memoria 2.6 Termina el CU.
Flujo alterno de los eventos de la sección "Eliminar Imagen":	
	2.1 Si la respuesta es falsa se termina el CU.

Tabla 12: Descripción del CU Gestionar Imagen.

Conclusiones Parciales

Con el desarrollo del Capítulo 2 , realizó el modelo conceptual definiendo como universo las imágenes, superficies, colisiones, juego, animaciones, cámara, luces y entidades, se generaron los artefactos ingenieriles de requisitos, encontrandose en el mismo 12 requisitos funcionales y 5 categorías de requisitos no funcianales como son los de Restricciones de diseño, Interfaz, Usabilidad, Soporte y los Requisitos Legales.

En la confección de los CU, se detectaron 12, de ellos se definieron como críticos a 10 de los cuales se le realizó la descripcion de Casos de Usos y sus respectivos descripciones.

Con el estudio de las herramientas similares al Motor se detectaron dos funcionalidades que se le pueden añadir al mismo como son: Que disponga de dos modos de edición: uno que consista en arrastrar y soltar acciones referidas a objetos, y otro para quienes sepan programar y quieran hacer juegos más originales y trabajados y el otro es que se pueda crear juegos que combinen personajes modelados en 3D con escenarios renderizados.

Capítulo 3:

Diseño.

Introducción

El objetivo que se propone este capítulo es la extracción de los diagramas de Secuencias y del Diagrama de Clases por paquete, acoplando estos paquetes según la funcionalidad de las clases, aplicando los patrones de diseño o arquitectura según se muestren en las relaciones entre las clases.

Se realizará además la Matriz de Trazabilidad como una forma de mostrar las relaciones entre requisito, casos de usos, diagramas de secuencia y clases del sistema.

3.1 Diseño del diagrama de clases por paquetes del Motor GT2D.

Un diagrama de Clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas, generalmente por definición son estáticos [14].

Los desarrolladores orientados a objetos con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, y se les da un nombre, podrían llamarse “Patrones”.

Los patrones de Diseño son “…el esqueleto de las soluciones a problemas comunes en el desarrollo de software.” [19].

Patrón de asignación de responsabilidad dentro están los patrones de GRASP “….los patrones de GRASP, nos guían para ayudarnos a encontrar los patrones de diseño (que son más concretos)…..” [19].

Los patrones de Creación son parte de los patrones GoF (Gang of Four). Este patrón estructura y encapsular una clase u objeto delegando responsabilidades a otros objetos [19].

Entre los utilizados están:

Patrones de GRASP:

Bajo Acoplamiento: Debe haber pocas dependencias entre las clases. Esto se evidencia en todas las relaciones entre las clase GT_ImagenManager y GT_Imagen que pertenecen al paquete de Cargar Imagen, en el paquete Cargar superficie las clases GT_SurfaceManager y GT_Surface, en el paquete Cargar Luces del Juego GT_LightManager y GT_Light, y en el paquete Gestionar Entidad GT_Entity3dManager y GT_Entity3d, GT_Entity2dManager y GT_Entity2d. .

Alta Cohesión: Cada elemento de nuestro diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. Esto se evidencia en todos los paquetes que tienen una función única en el sistema como por ejemplo los paquetes Posicionar Cámara, Gestionar Animación, Cargar Superficie y en Gestionar Entidad.

Controlador: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto se demuestra a través del Paquete Cargar Juego en la clase Render que es la encargada de hacer funcionar todas las funcionalidades del sistema, otro tipo de controladora es en el Paquete Gestionar Animación donde se encuentra la clase controladora GT_AnimationManager donde controla el flujo de eventos de la clase GT_Animation que controla el flujo de las clases GT_Secuence y GT_Frame.

Patrones GoF

Singleton: Factoría que asegura que solo hay un miembro de una clase y proporciona un punto global de acceso a él, demostrándose en todas las clases Manager que son las que controlan la función de cada paquete, como son las clases GT_LightManager, GT_ImagenManager, GT_Entity3dManager, GT_Entity2dManager, GT_AnimationManager, GT_SurfaceManager .

El diagrama de clases está constituido por un:

Paquete Posicionar Cámara: Contiene las clases GT_Cámara2D y GT_Cámara3D y estas clases velan por el buen funcionamiento de las cámaras en el juego.

Paquete Cargar Luces del Juego: Contiene las clases GT_LightManager y GT_Light y estas se encargar de velar por el buen funcionamiento de las luces en el juego.

Paquete Cargar Imagen: Contiene las clases GT_ImagenManager y GT_Imagen y estas se encargar de velar por el buen funcionamiento de las imágenes correspondientes a cada animación y superficie en el juego.

Paquete Gestionar Entidad: Contiene las clases GT_Entity3dManager, GT_Entity2dManager, GT_Entity2d y GT_Entity3d y estas se encargar de velar por el buen funcionamiento de las entidades 2d y 3d correspondientes a cada animación y superficie en el juego.

Paquete Gestionar Animación: Contiene las clases GT_AnimationManager, GT_Animation, GT_Secuence, GT_Frame y estas se encargar de velar por el buen funcionamiento de las animaciones en el juego.

Paquete Cargar Superficie: Contiene las clases GT_SurfaceManager y GT_Surface y estas se encargar de velar por el buen funcionamiento de las superficies en el juego.

Paquete Cargar Joystick: Contiene las clases CJoystickButton, CJoystickAxis, CJoystickHat y CJoystick son las encargadas de guardar toda la información sobre las acciones realizadas con el teclado y el mouse en el juego.

CKey: Esta clase junto con **GT_Timer, GT_Input, GT_Render** son las encargadas de dar la integración final al juego.

Collision Parser: Esta es la clase que se encarga generar las colisiones del juego.

Para más información de la descripción de las clases se anexó un artefacto de descripción de las clases al documento de tesis, donde el mismo está estructurado por los campo nombre de la clase, la descripción de la clase para poner que función es la que realiza dentro del sistema, métodos de la clases, clases amigas, estructuras que utiliza, donde de cada uno se registras el tipo el nombre y su descripción. Para más información ver anexos las figuras 15, 16, 17, 18, 19, 20, 21.

3.2 Diagrama de Paquetes del Motor GT2D.

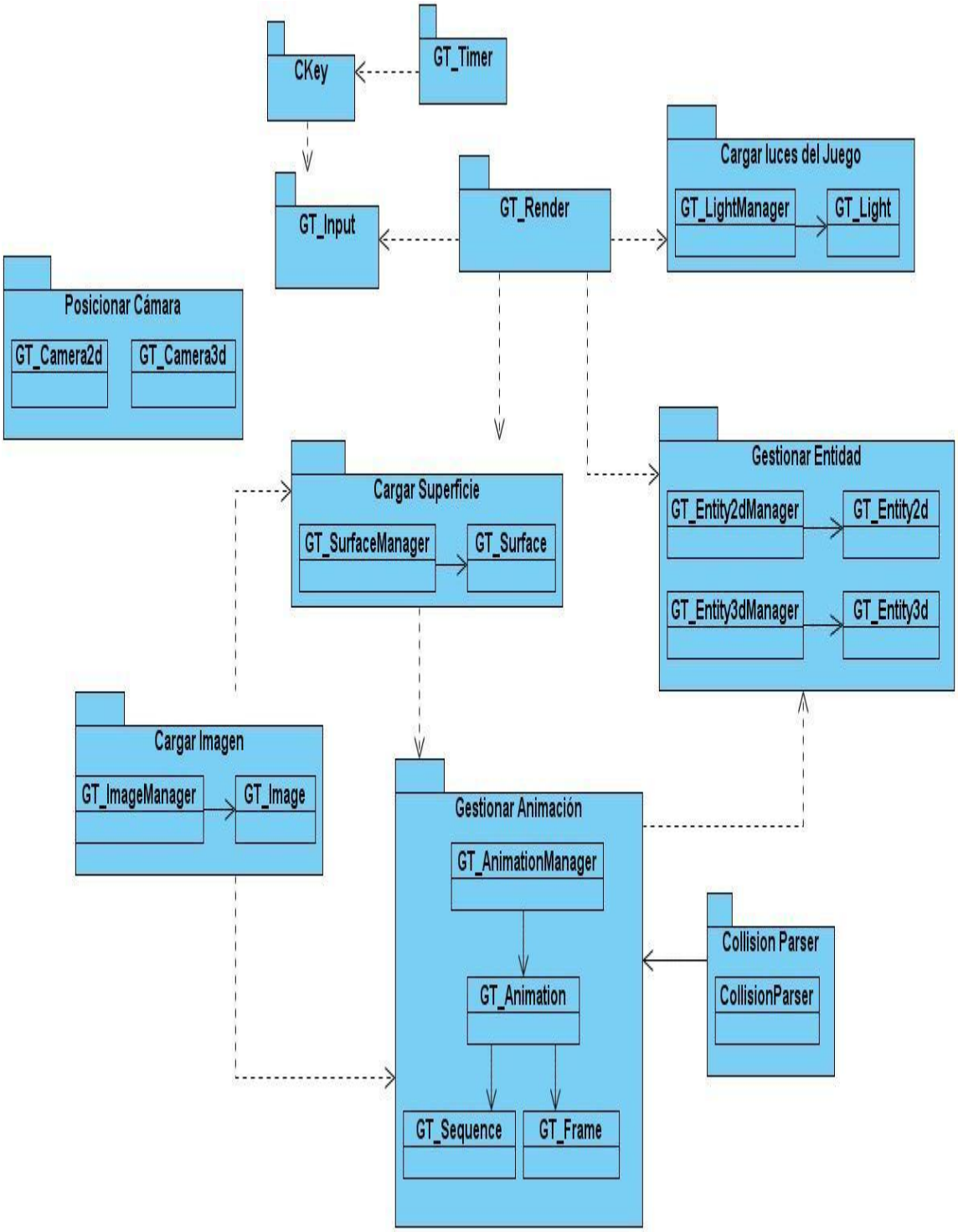
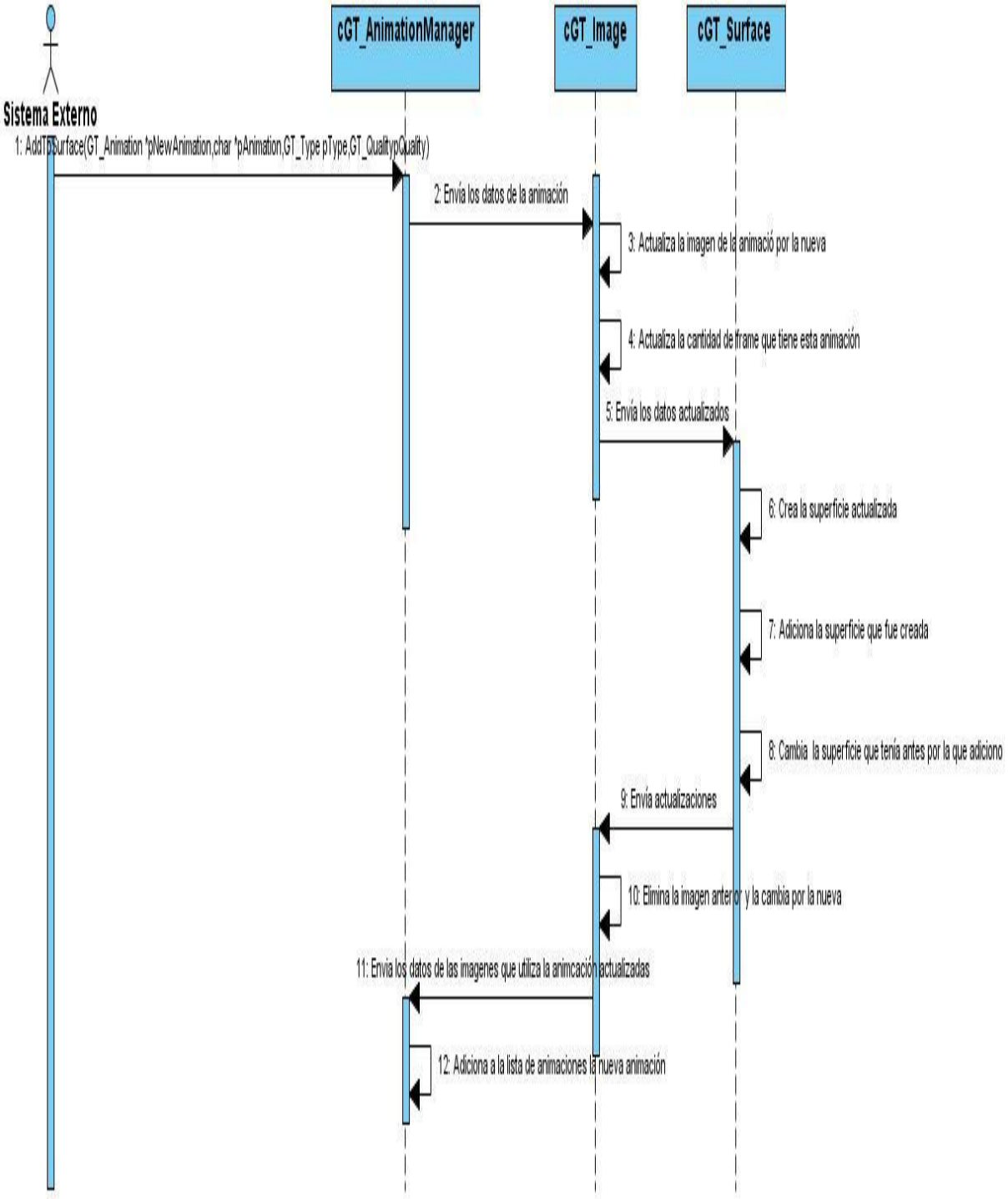


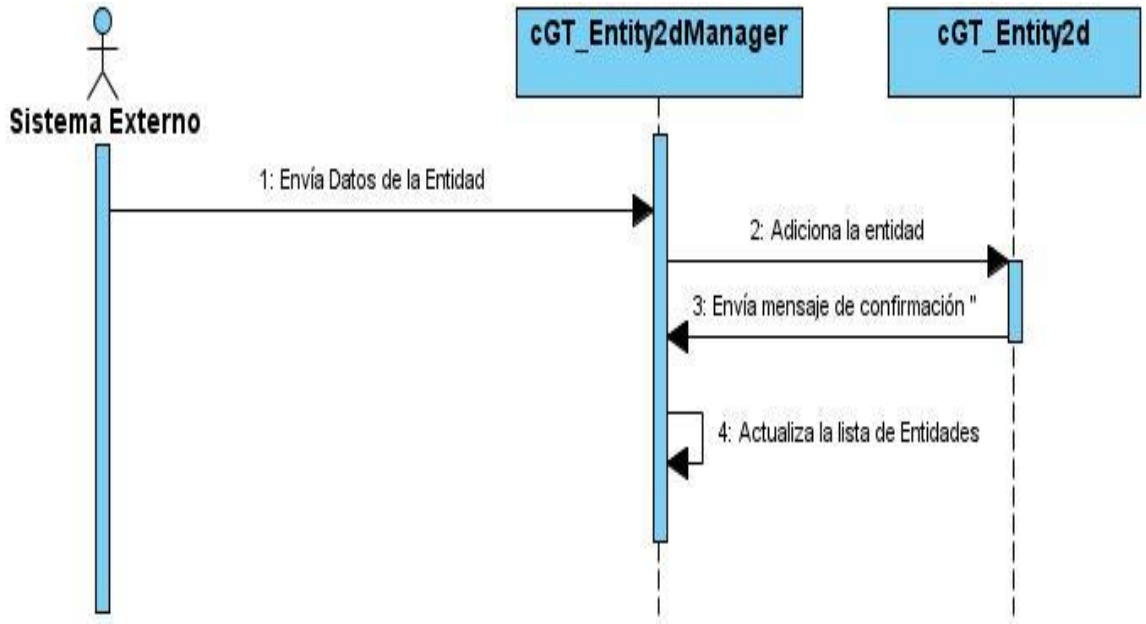
Figura 3.2 Diagrama de Paquetes.

3.3 Extracción de los Diagramas de Secuencia.

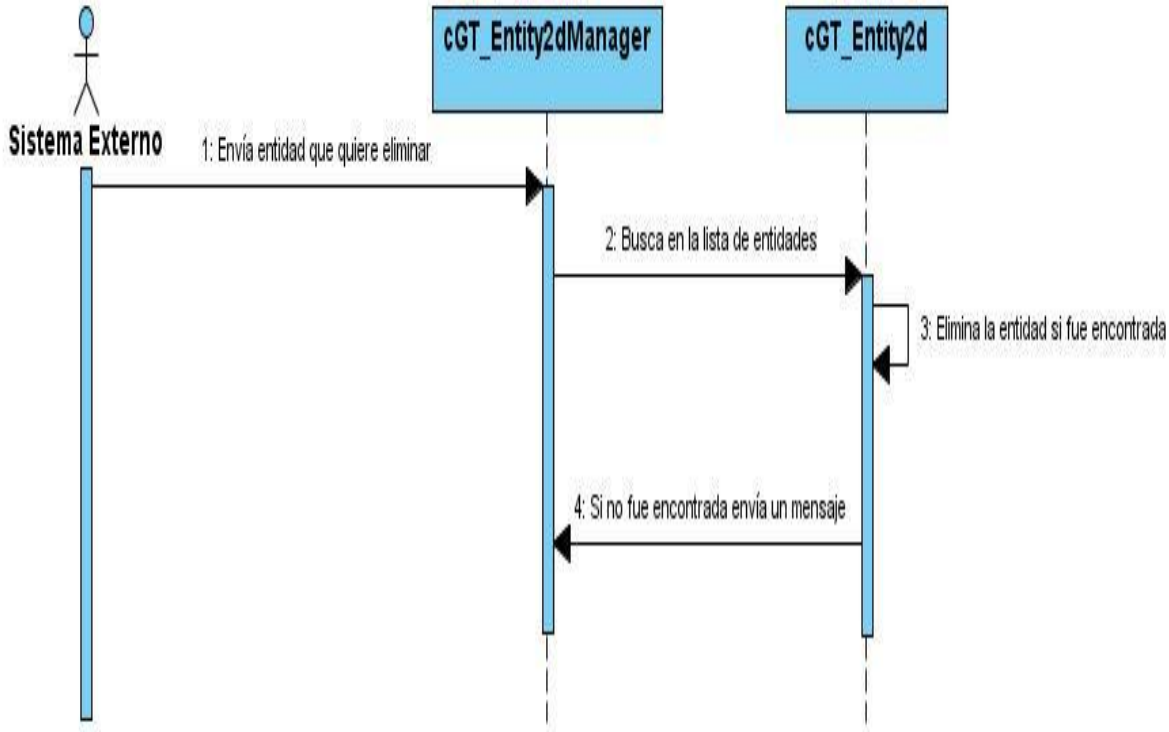
3.3.1 Diagrama de Secuencia Adicionar Animación.



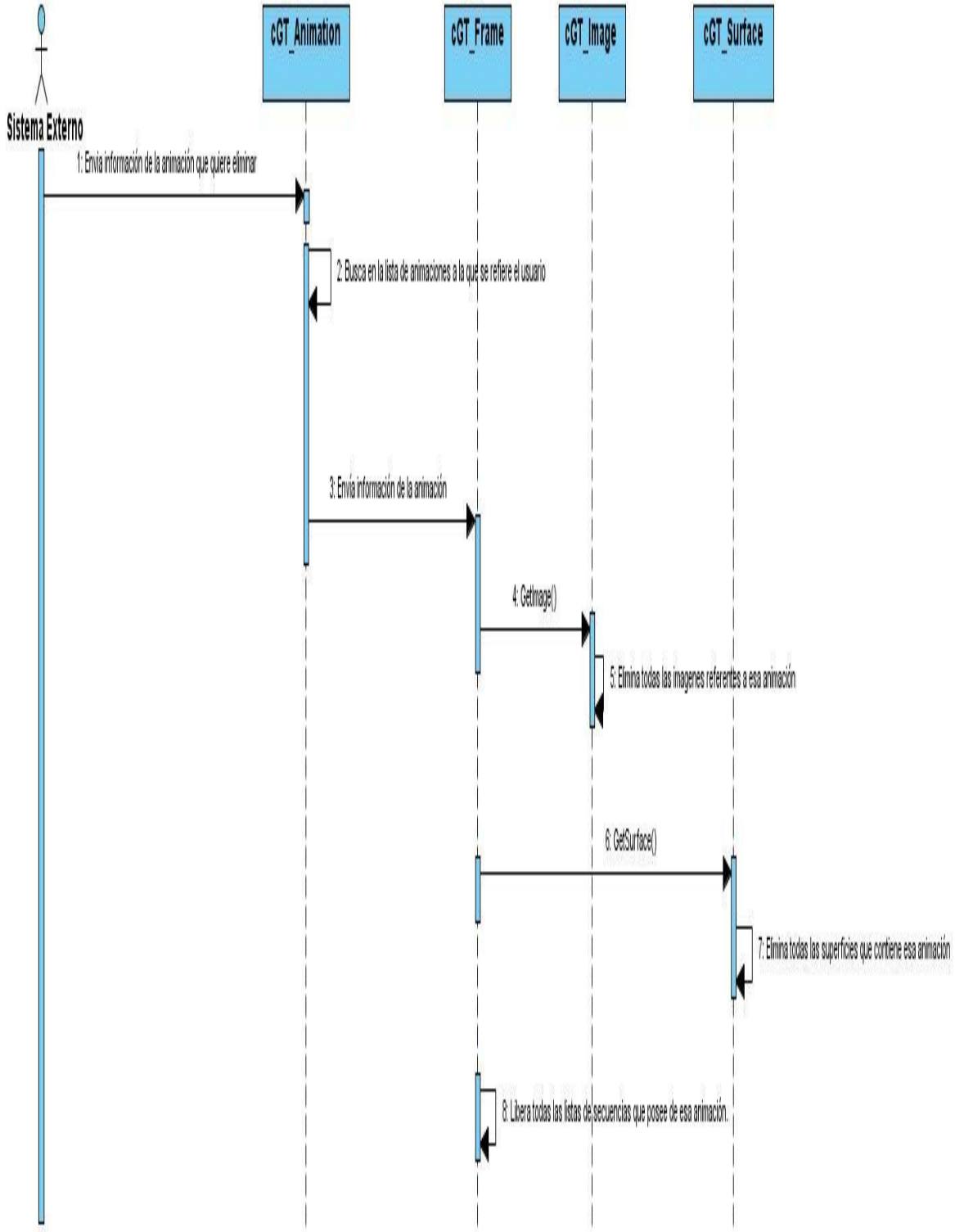
3.3.2 Diagrama de Secuencia Adicionar Entidad.



3.3.3 Diagrama de Secuencia de Eliminar una Entidad.

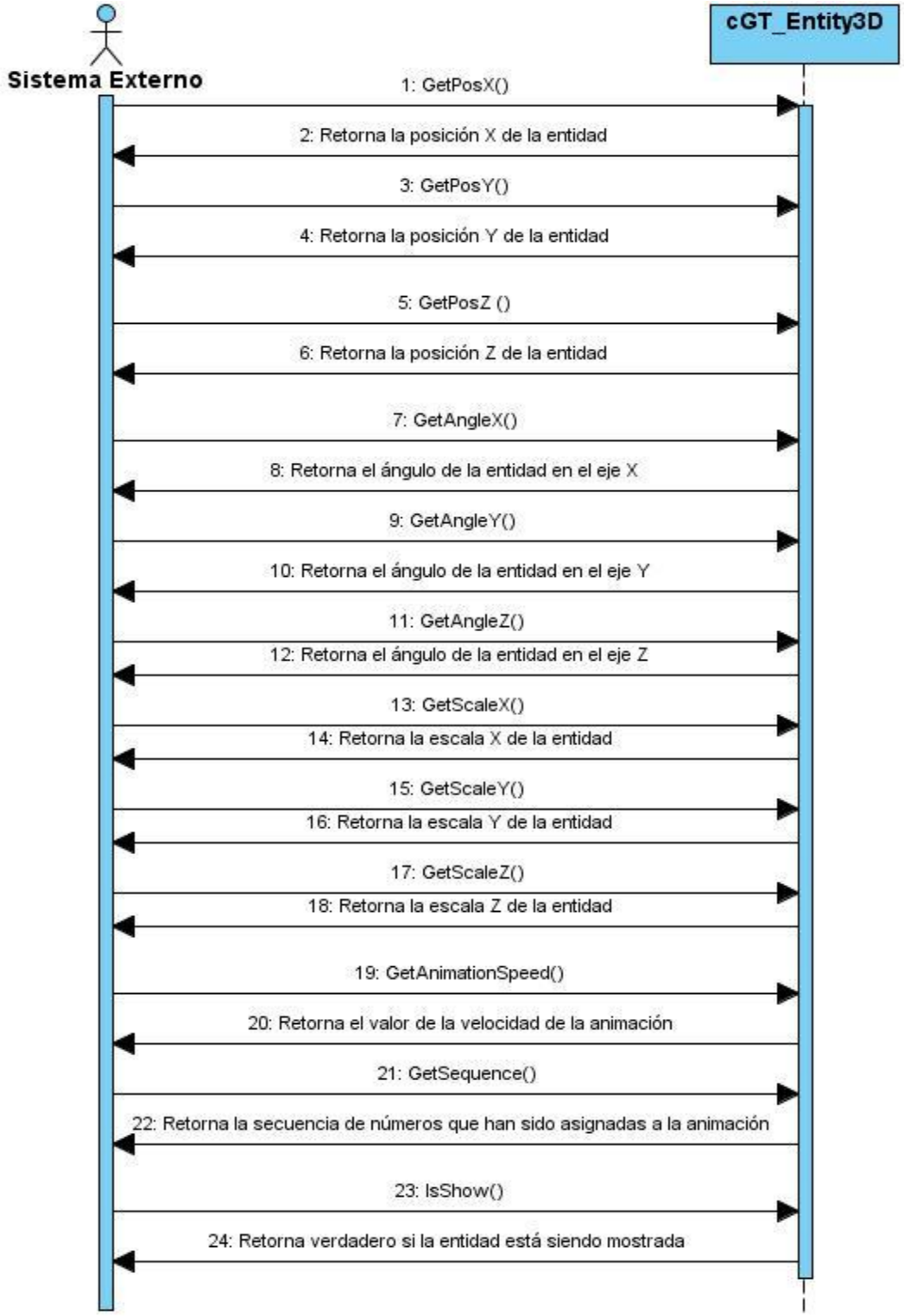


3.3.4 Diagrama de Secuencia de Eliminar una Animación.

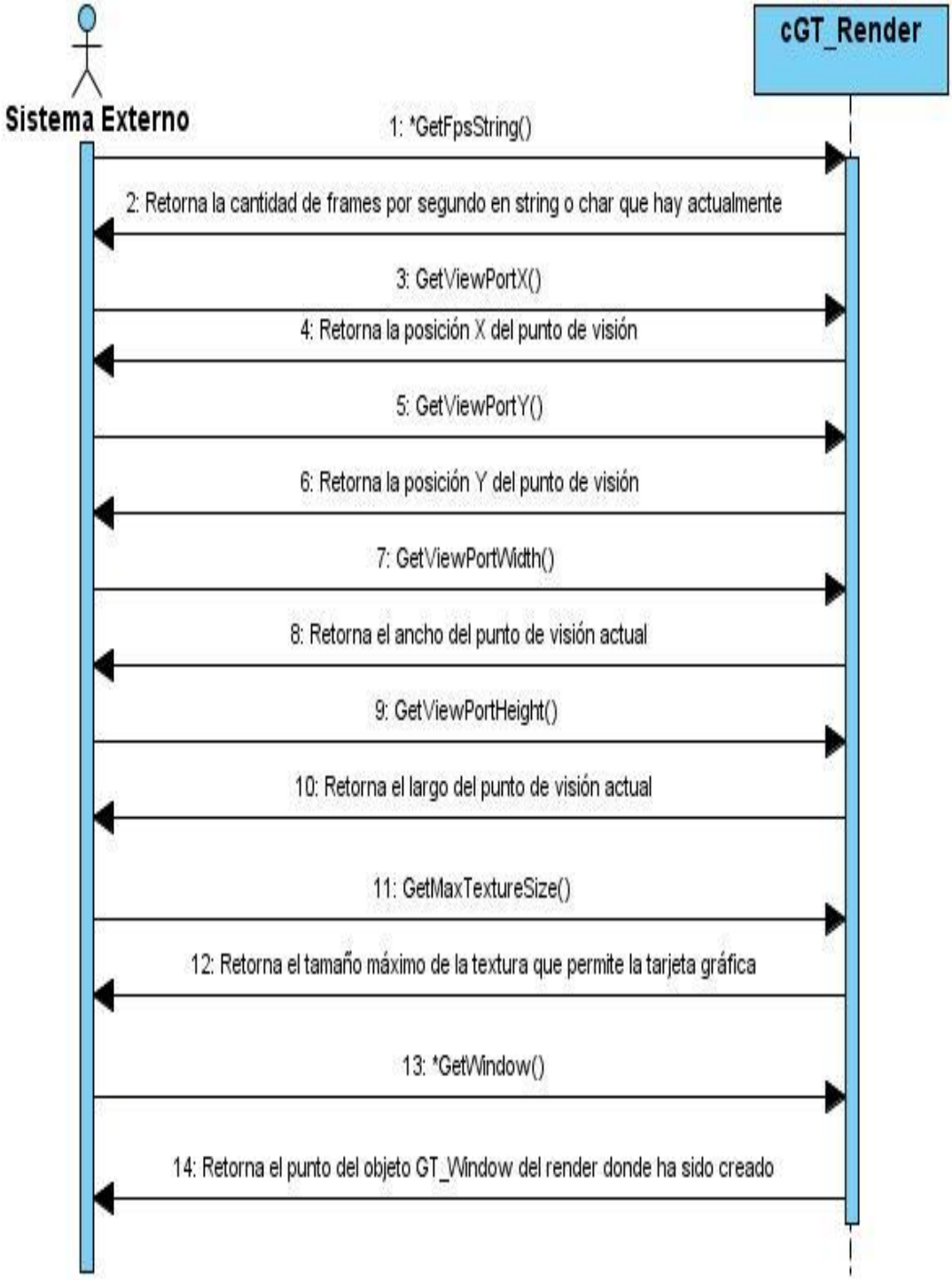


3.3.5 Diagrama de Secuencia de Cargar Entidad2D.

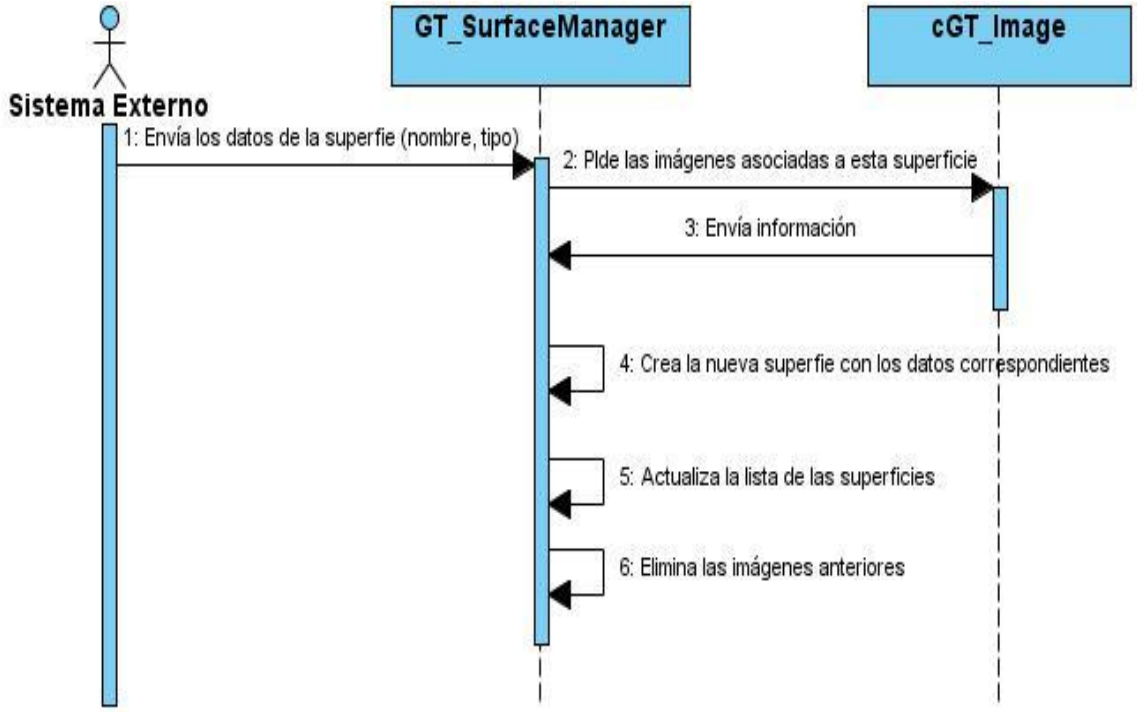
3.3.6 Diagrama de Secuencia de Cargar Entidad3D.



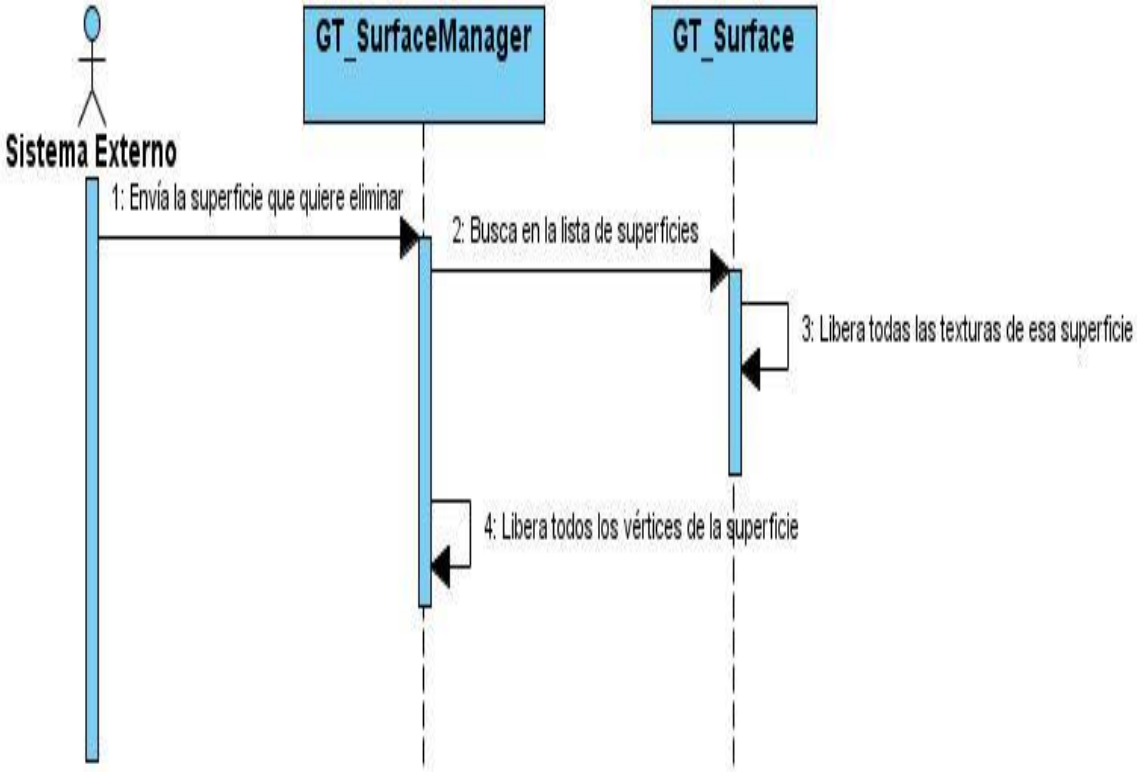
3.3.7 Diagrama de Secuencia de Cargar Juego.



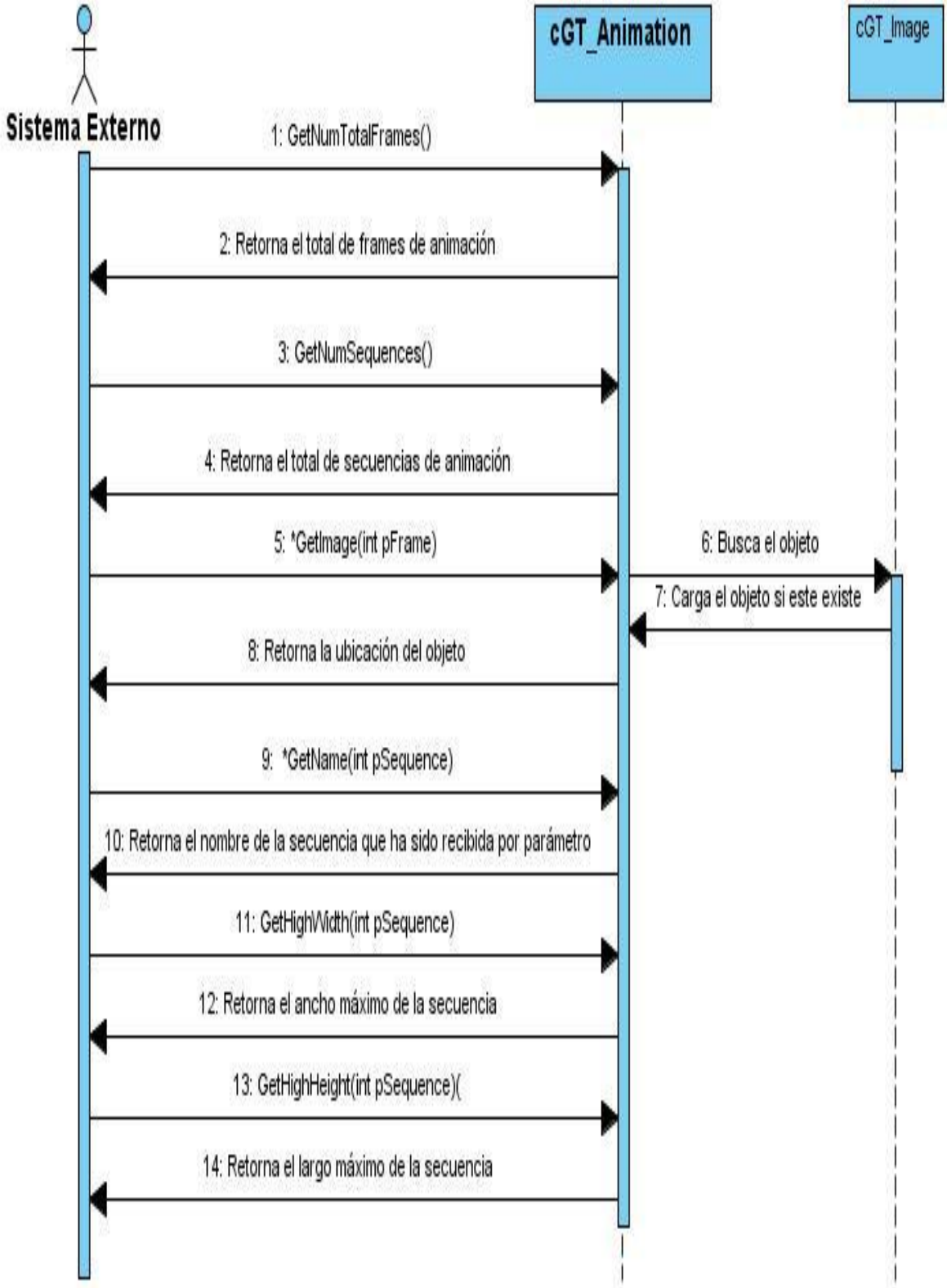
3.3.8 Diagrama de Secuencia de Adicionar Superficie.



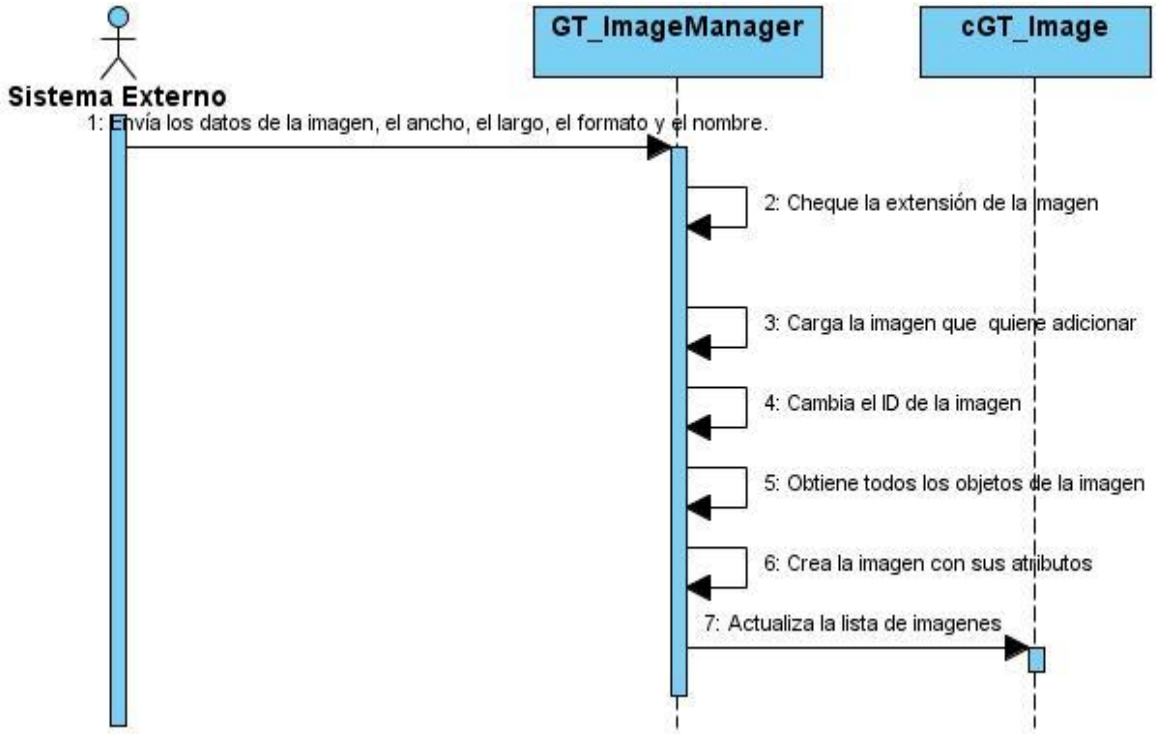
3.3.9 Diagrama de Secuencia de Eliminar Superficie.



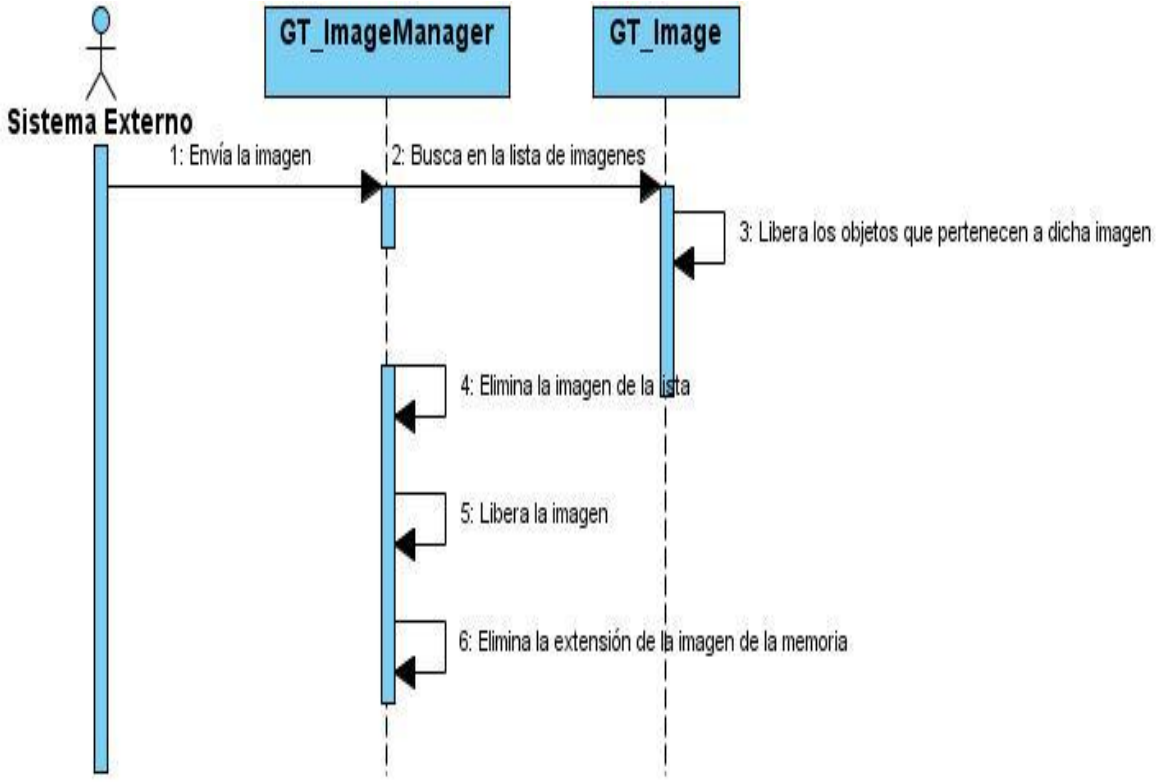
3.3.10 Diagrama de Secuencia Cargar Animación.



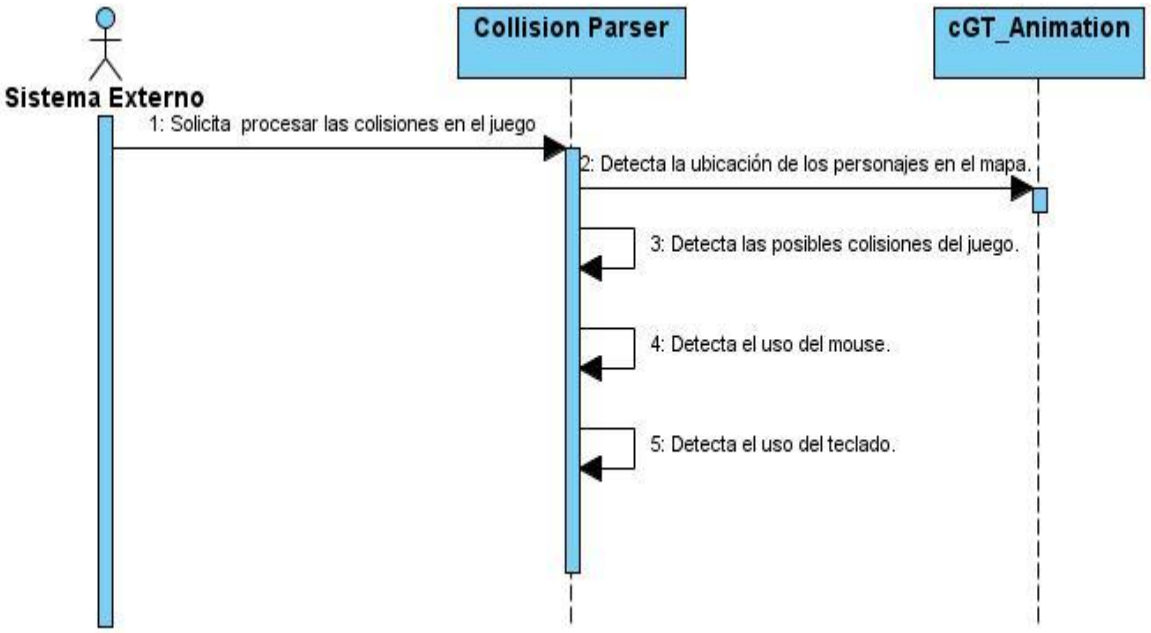
3.3.11 Diagrama de Secuencias Adicionar Imagen.



3.3.12 Diagrama de Secuencias Eliminar Imagen.



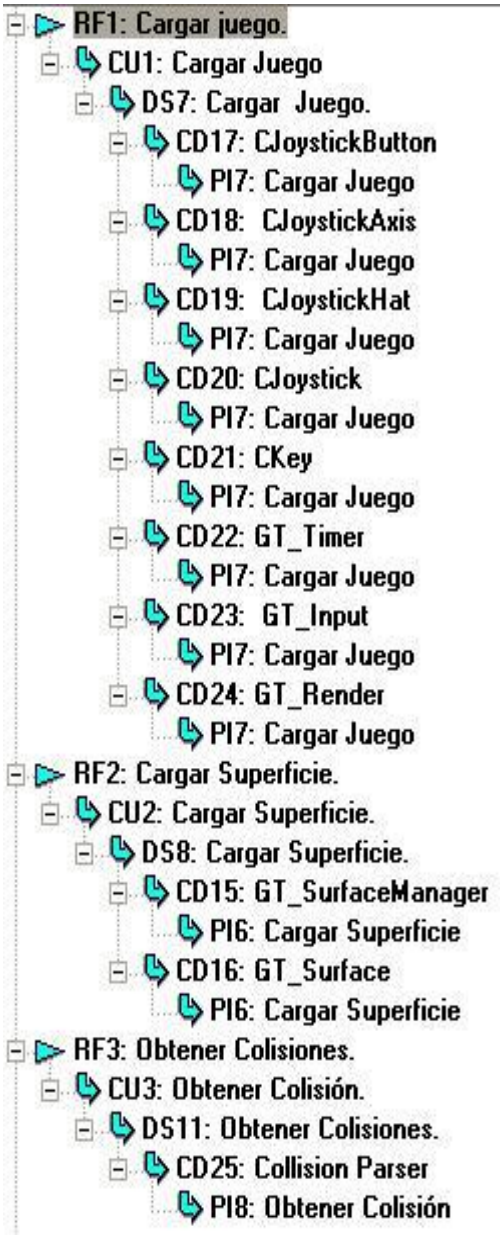
3.3.13 Diagrama de Secuencias Obtener Colisión.

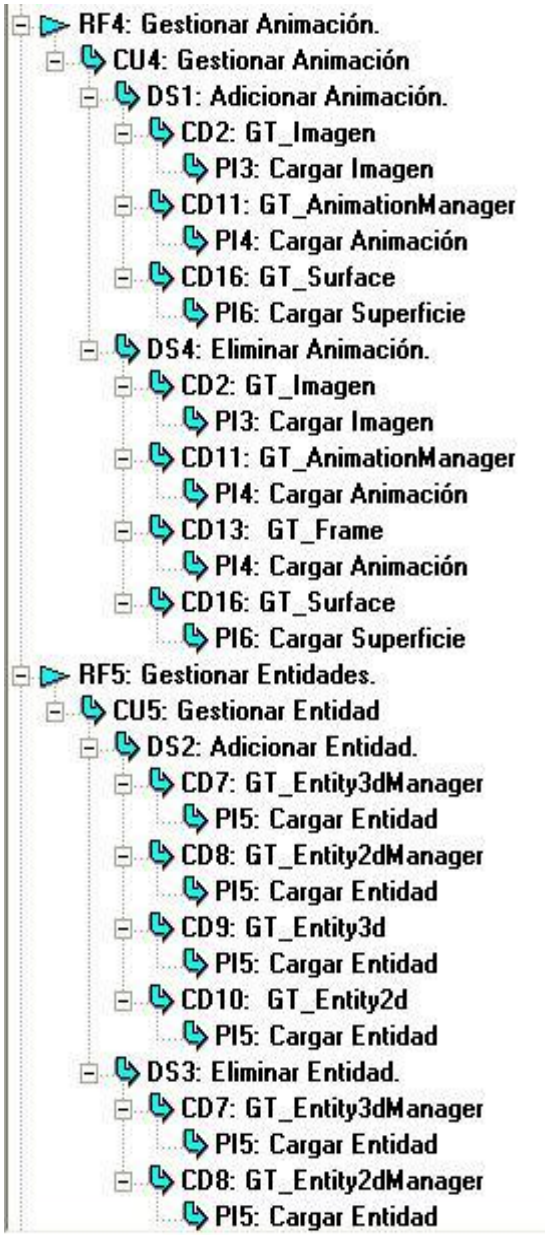


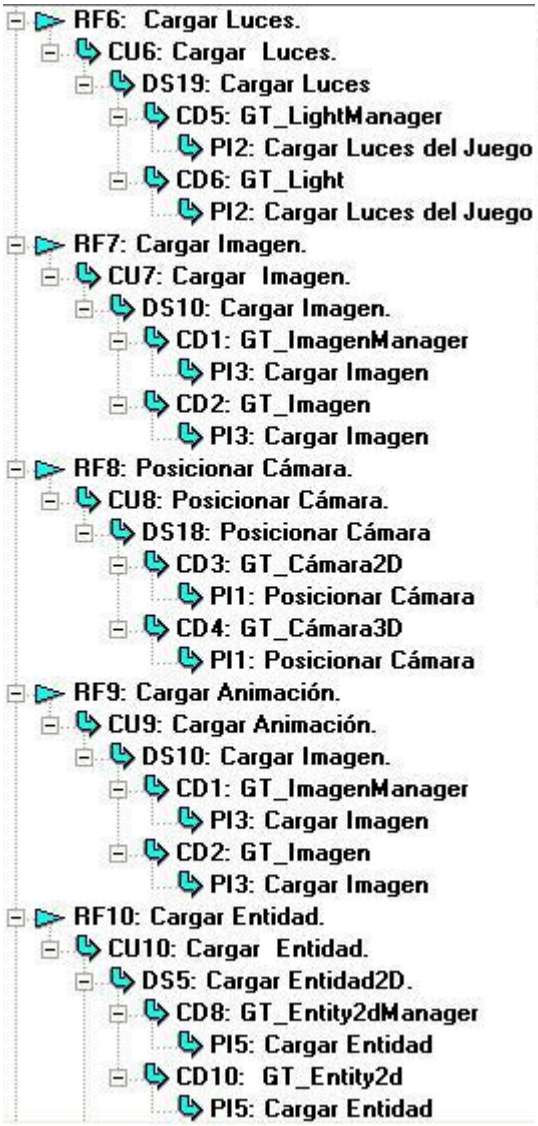
3.4 Matriz de Trazabilidad.

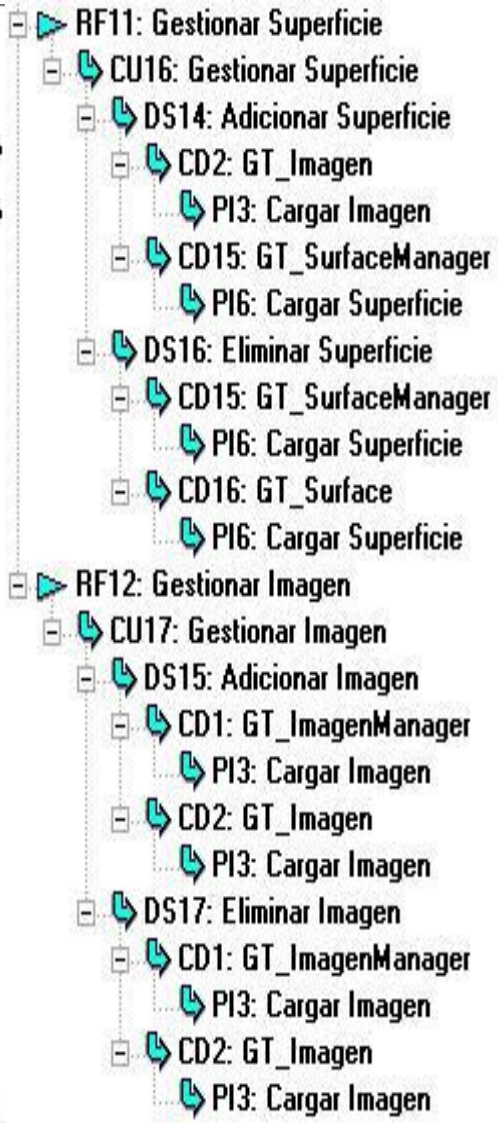
Para realizar la Matriz de Trazabilidad se utilizó la herramienta Rational RequisitePro para obtener un resultado más ilustrativo y completo.

3.4.1- Árbol de Trazabilidad.

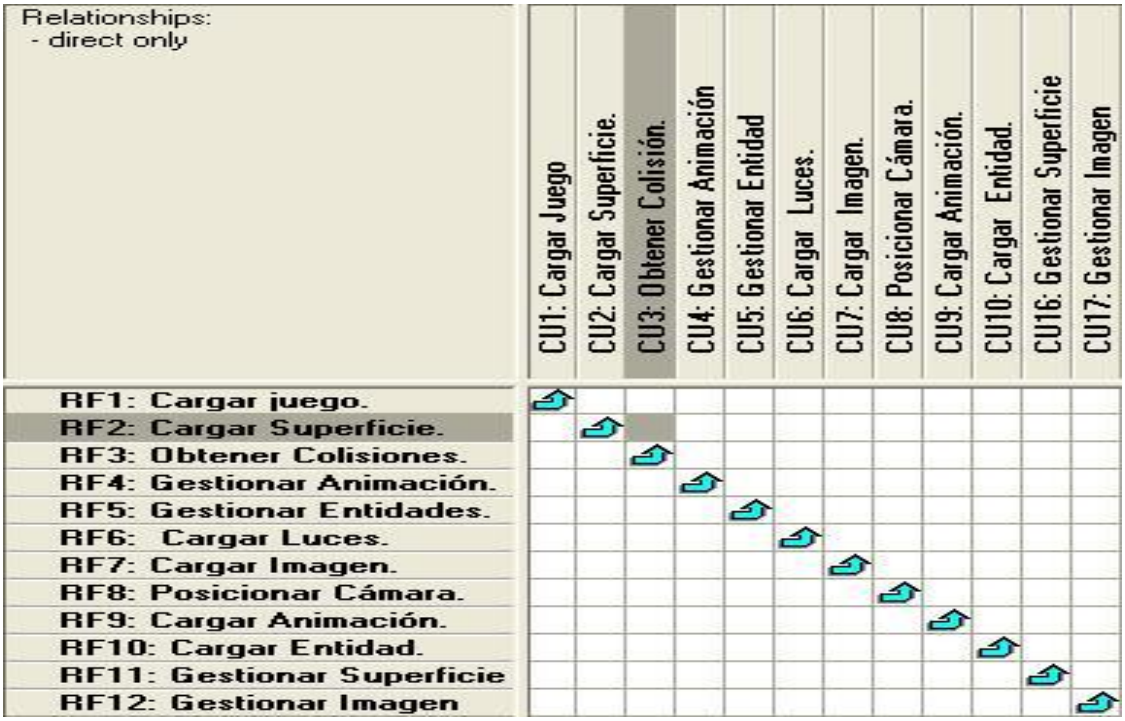




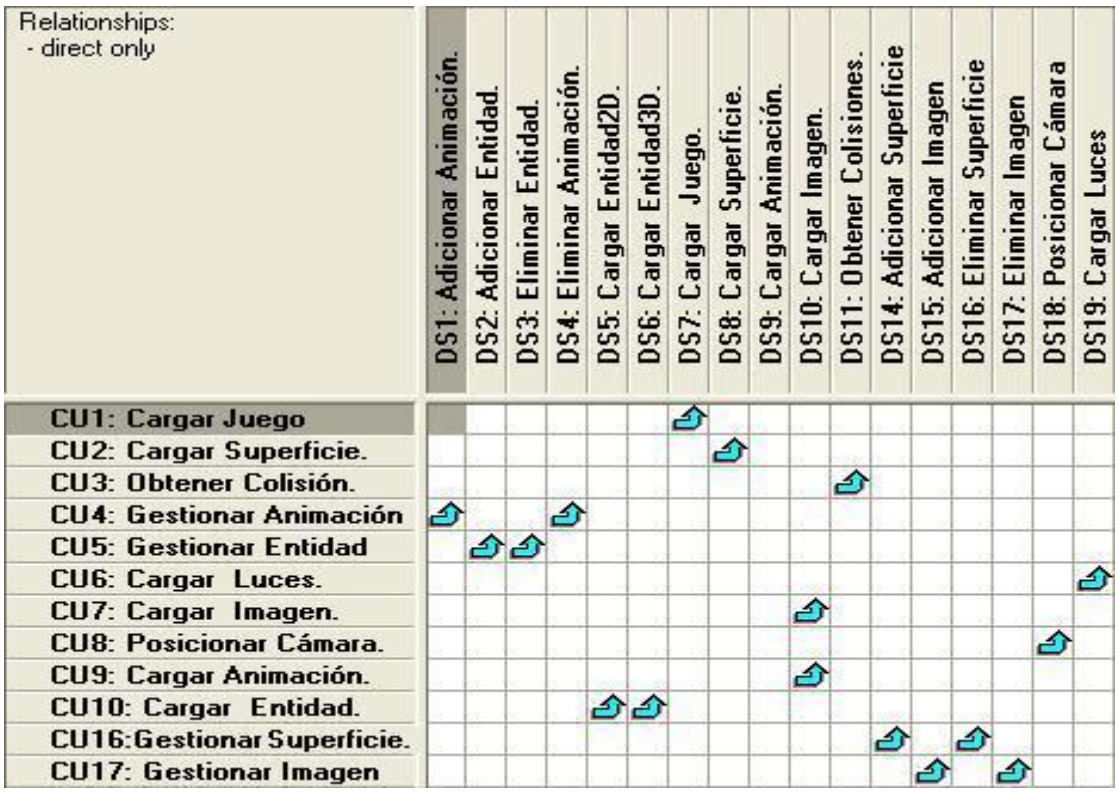

RF6: Cargar Luces.
CU6: Cargar Luces.
DS19: Cargar Luces
CD5: GT_LightManager
PI2: Cargar Luces del Juego
CD6: GT_Light
PI2: Cargar Luces del Juego
RF7: Cargar Imagen.
CU7: Cargar Imagen.
DS10: Cargar Imagen.
CD1: GT_ImagenManager
PI3: Cargar Imagen
CD2: GT_Imagen
PI3: Cargar Imagen
RF8: Posicionar Cámara.
CU8: Posicionar Cámara.
DS18: Posicionar Cámara
CD3: GT_Cámara2D
PI1: Posicionar Cámara
CD4: GT_Cámara3D
PI1: Posicionar Cámara
RF9: Cargar Animación.
CU9: Cargar Animación.
DS10: Cargar Imagen.
CD1: GT_ImagenManager
PI3: Cargar Imagen
CD2: GT_Imagen
PI3: Cargar Imagen
RF10: Cargar Entidad.
CU10: Cargar Entidad.
DS5: Cargar Entidad2D.
CD8: GT_Entity2dManager
PI5: Cargar Entidad
CD10: GT_Entity2d
PI5: Cargar Entidad


RF11: Gestionar Superficie
CU16: Gestionar Superficie
DS14: Adicionar Superficie
CD2: GT_Imagen
PI3: Cargar Imagen
CD15: GT_SurfaceManager
PI6: Cargar Superficie
DS16: Eliminar Superficie
CD15: GT_SurfaceManager
PI6: Cargar Superficie
CD16: GT_Surface
PI6: Cargar Superficie
RF12: Gestionar Imagen
CU17: Gestionar Imagen
DS15: Adicionar Imagen
CD1: GT_ImagenManager
PI3: Cargar Imagen
CD2: GT_Imagen
PI3: Cargar Imagen
DS17: Eliminar Imagen
CD1: GT_ImagenManager
PI3: Cargar Imagen
CD2: GT_Imagen
PI3: Cargar Imagen

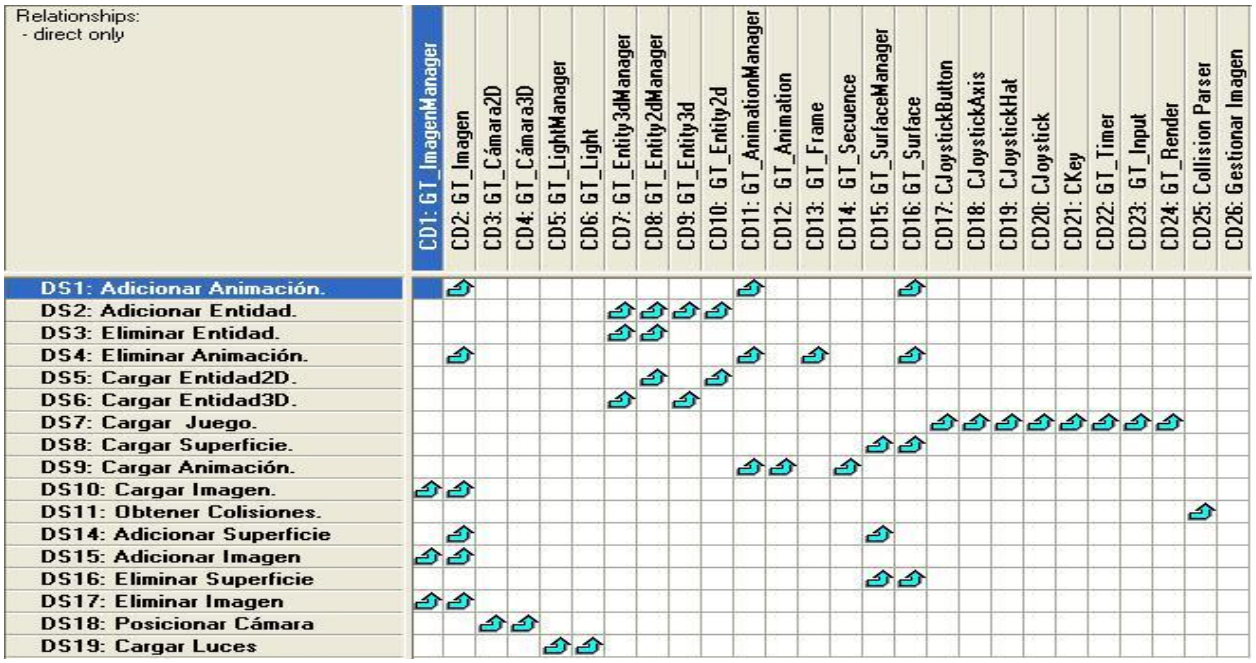
3.4.2- Relación de trazabilidad entre Requisito Funcional y Casos de Uso.



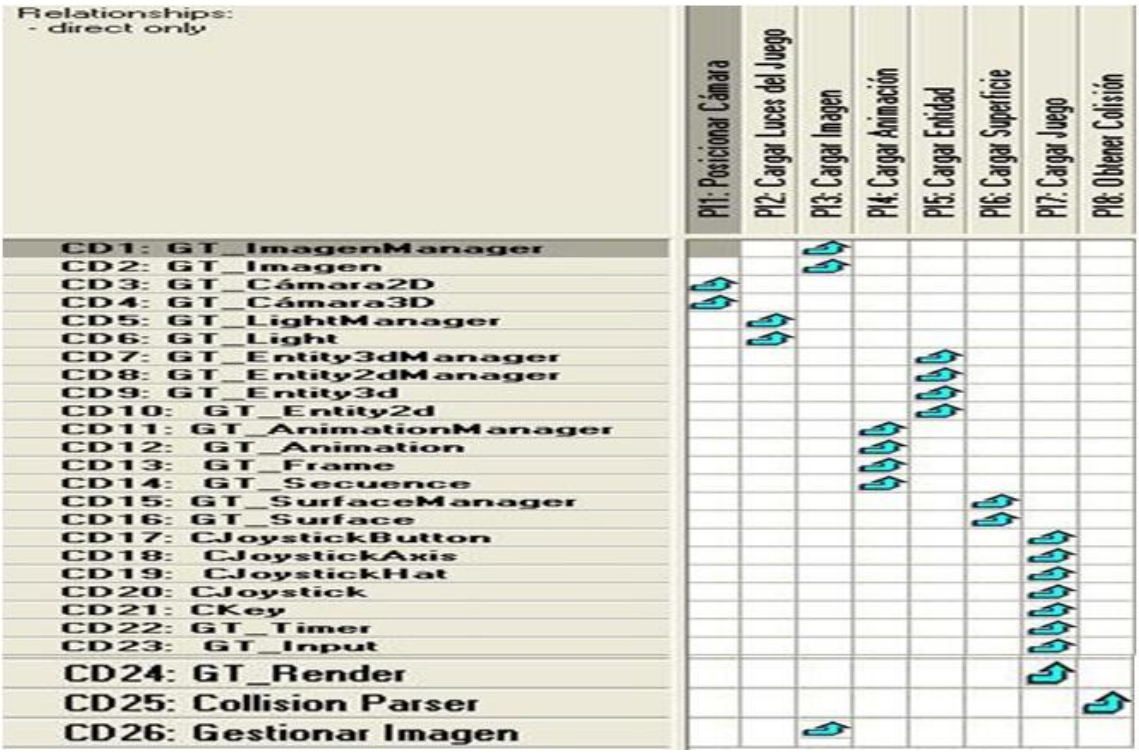
3.4.3- Relación de trazabilidad entre Casos de Uso y Diagrama de Secuencia.



3.4.4- Relación de trazabilidad entre Diagrama de Secuencia y Clases del Diseño.



3.4.5- Relación de trazabilidad entre Clases del Diseño y paquetes de Implementación.



Conclusiones Parciales

En este capítulo se aprovechó la información obtenida de las investigaciones sobre el Motor GT2D, para así hacer la extracción del Diagrama de Clase agrupado por paquetes según su funcionalidad, aplicando los patrones de GRASP y GoF como una forma de organización.

Luego de plasmar el diagrama de Clase se realizó la extracción de los Diagramas de Secuencias correspondiente a la misma, para así dar paso a las Matrices de Trazabilidad permitiendo generar el árbol de trazabilidad.

CONCLUSIONES.

La documentación obtenida describe el sistema permitiendo una mejor comprensión, reutilización y modificación de este. Para legar a este resultado se seleccionó la Metodología ágil Open UP por ser la más adecuada para aplicar el proceso de la Ingeniería Inversa, por su característica de ser aplicable a proyectos pequeños y que genera menos de 20 artefactos, luego de hacer la selección de metodología se plasmaron las definiciones que se creyeron importantes para desarrollar la presente investigación, exponiéndose el porqué del uso del lenguaje UML y de las herramientas Visual Parading y OSRMT.

Se realizó el modelo conceptual definiendo como universo las imágenes, superficies, colisiones, juego, animaciones, cámara, luces y entidades, se generaron los artefactos ingenieriles de requisitos funcionales y no funcionales, encontrandose en el mismo 12 requisitos funcionales y 5 categorías de requisitos no funcianales como son los Restricciones de diseño, Interfaz, Usabilidad, Soporte y los Requisitos Legales.

En la confección de los CU, se detectaron 12, de ellos se definieron como críticos a 10 de los cuales se le realizó la descripcion de Casos de Usos y sus respectivos descripciones.

Con la información obtenida se confeccionó el Diagrama de Paquetes agrupado según su funcionalidad, aplicando los patrones de GRASP y GoF como una forma de organización.

Luego de plasmar el diagrama de Clase se realizó la extracción de los Diagramas de Secuencias correspondiente a la misma, para así dar paso a la Matriz de Trazabilidad permitiendo generar el árbol de trazabilidad.

RECOMENDACIONES.

Se recomienda implementar las siguientes funcionalidades en el sistema:

- Que disponga de dos modos de edición: uno más básico, que consista en arrastrar y soltar acciones referidas a objetos, y otro más avanzado para quienes sepan programar y quieran hacer juegos más originales y trabajados.

- Que se pueda crear juegos que combinen personajes modelados en 3D con escenarios renderizados.

GLOSARIO DE TÉRMINOS.

Feedback: Equivale a retroalimentación o retroacción, y consiste en introducir los resultados obtenidos como datos para considerar al inicio del nuevo proceso, lo que permitirá rectificar si procede dicho proceso.

Fotograma: Son las imágenes individuales captadas por cámaras de video y registradas analógica o digitalmente.

Frames: Se denomina frame en inglés, a un fotograma o cuadro, una imagen particular dentro de una sucesión de imágenes que componen una animación.

Framework: Es una estructura conceptual y tecnológica de soporte, definida normalmente con artefactos o módulos de *software* concretos, con base en la cual otro proyecto de *software* puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Ingeniería Progresiva: Renovación o Reclamación, recupera la información del diseño de un software ya existente.

Joystick: Dispositivo de control de dos o tres ejes que se usa desde una computadora o videoconsola hasta un transbordador espacial o los aviones de caza.

Motor: Un motor de videojuego es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego.

Paneo: Acción de desplazarse en una imagen.

Render: Renderizado (render en inglés) es un término usado para referirse al proceso de generar una imagen desde un modelo.

Sprites: Tipo de mapa de bits dibujados en la pantalla de ordenador por hardware gráfico especializado.

BIBLIOGRAFÍA REFERENCIADA Y CONSUTADA.

1. Capitulo 4. Ingeniería Inversa.
2. Guatemala, 1997, Tesis: La ingeniería Inversa como un apoyo a las herramientas Case. Cambiando la mentalidad de los desarrolladores. Evelyn Mabell Contreras Sagastume, Herman Alfonso Reyes Girón. Universidad Francisco Marroquín, Facultad de Ingeniería de Sistemas Informática y Ciencias de la Computación.
3. McGraw Hill, 1997, Pressman, R. Ingeniería del Software: Un enfoque práctico. s.l.
4. 1990, Society, IEEE Computer. “std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology”..
5. 25 August 2010, All Bitmap 2.0, License: GPL3, http://www.mugenation.com/staff/nobun/download/allbmp20/index.html.
6. 3 Noviembre 2010, Guillermo Costamagna Quinteros http://www.tierradesoldados.com/?p=138.
7. 30 enero 2010, Fran G, http://www.gigle.net/game-maker-desarrolla-tu-propio-videojuego/.
8. Copyright 2007,Propiedad Intelectual Española: http://www.tutorial-enlace.net.
9. 'http://www.willydev.net/descargas/prev/TodoAgil.Pdf'.
10. 'http://www.avemundi.com/archivos/XP.doc'
11. 1993, Nosek, Palvia y. "A Field Examination of System Life Cycle Techniques and Methodologies.". .
12. 2004, al, Piattini et. " Una perspectiva de la ingeniería de software".
13. November 30, 2010, EPF 1.5.1.1 – License, http://epf.eclipse.org/wikis/openup/.
14. 7-07-08, Iván Garcerant, http://synergix.wordpress.com/2008/07/07/requisito-funcional-y-no-funcional/.
15. 12-10-08, Iván Garcerant, http://synergix.wordpress.com/category/casos-de-uso/.
16. 16-04-10 Iván Garcerant, http://synergix.wordpress.com/category/uml/.
17. 3 May 2011, www.visual-paradigm.com/
18. 16-04-10, http://synergix.wordpress.com/
19. 19 marzo 2011, Nicolás Tedeschi, http://arevalomaria.wordpress.com.

ANEXOS.

Representación gráfica del contenido anterior.

Anexo.1 Proceso de Ingeniería Inversa.

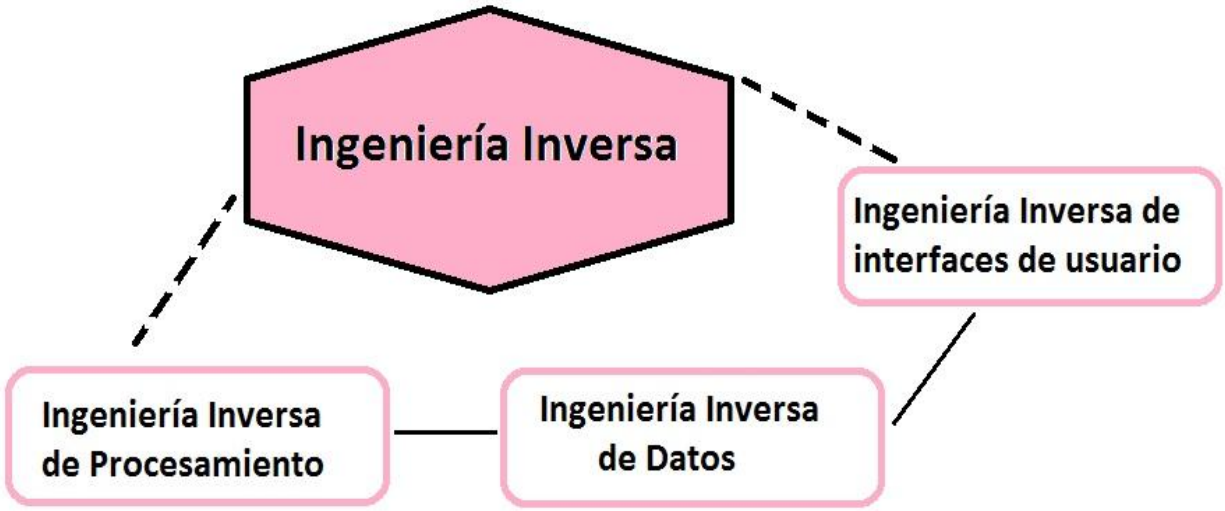


Figura.1 Proceso de Ingeniería Inversa.

Anexo.2 Proceso de restructuración del código.

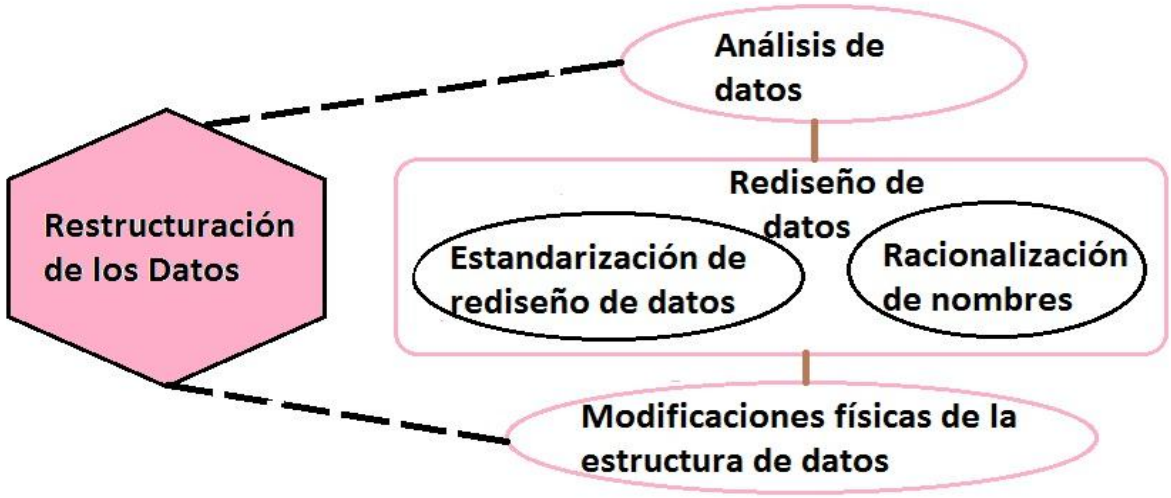
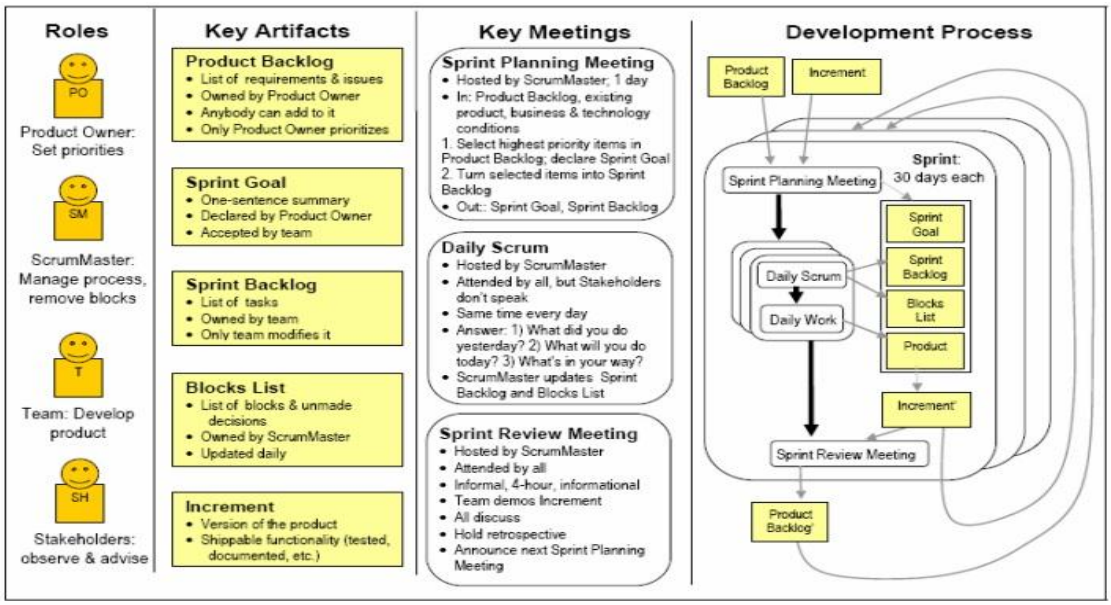


Figura.2 Proceso de restructuración del código.

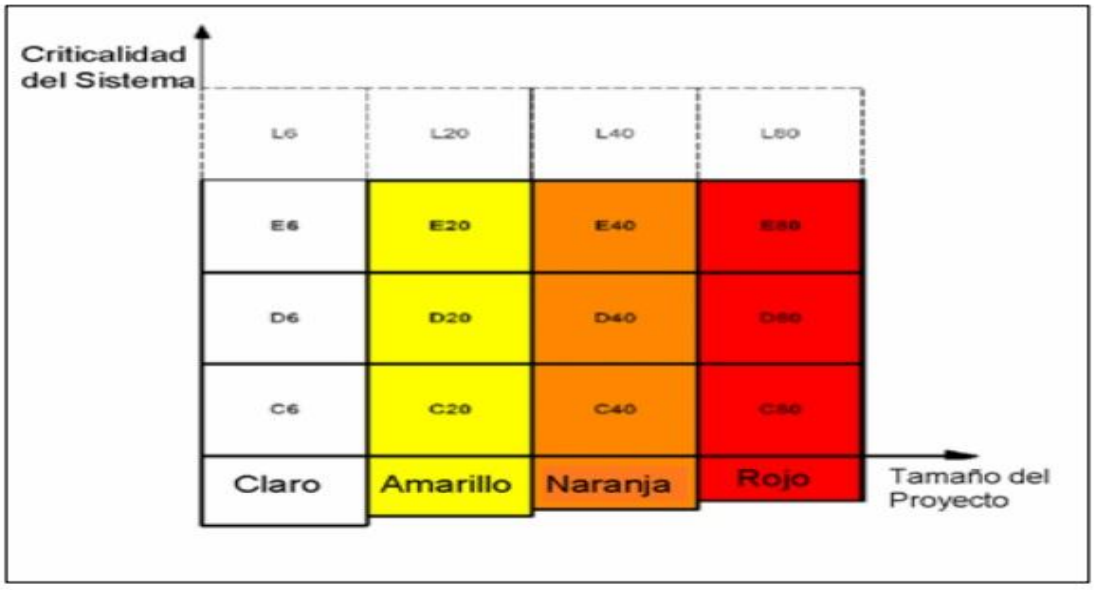
Anexo.3 Scrum [9].



Descripción de Roles, artefactos, reuniones y proceso de desarrollo de Scrum.

Figura.3 Scrum.

Anexo.4 Crystal [9].



Familia de Crystal Methods

Figura.4 Crystal.

Anexo.5 Ciclo de Vida de Open UP [13].

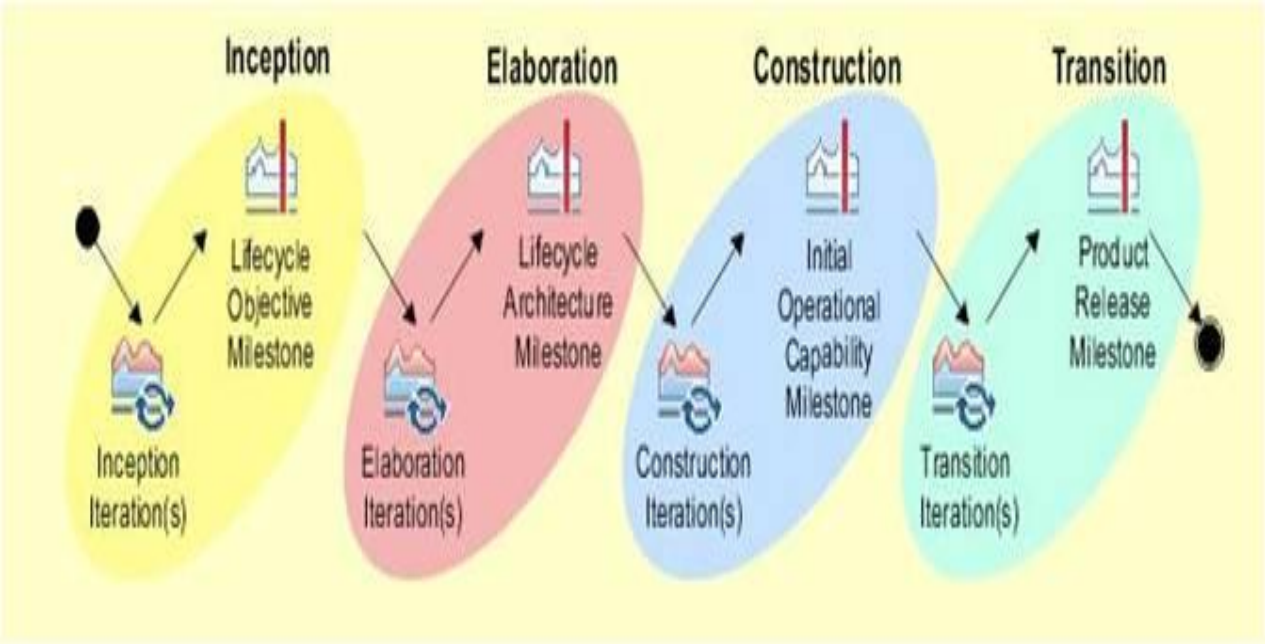


Figura.5 Ciclo de Vida de Open UP.

Anexo.6 Procesos en Reingeniería de Software [1].

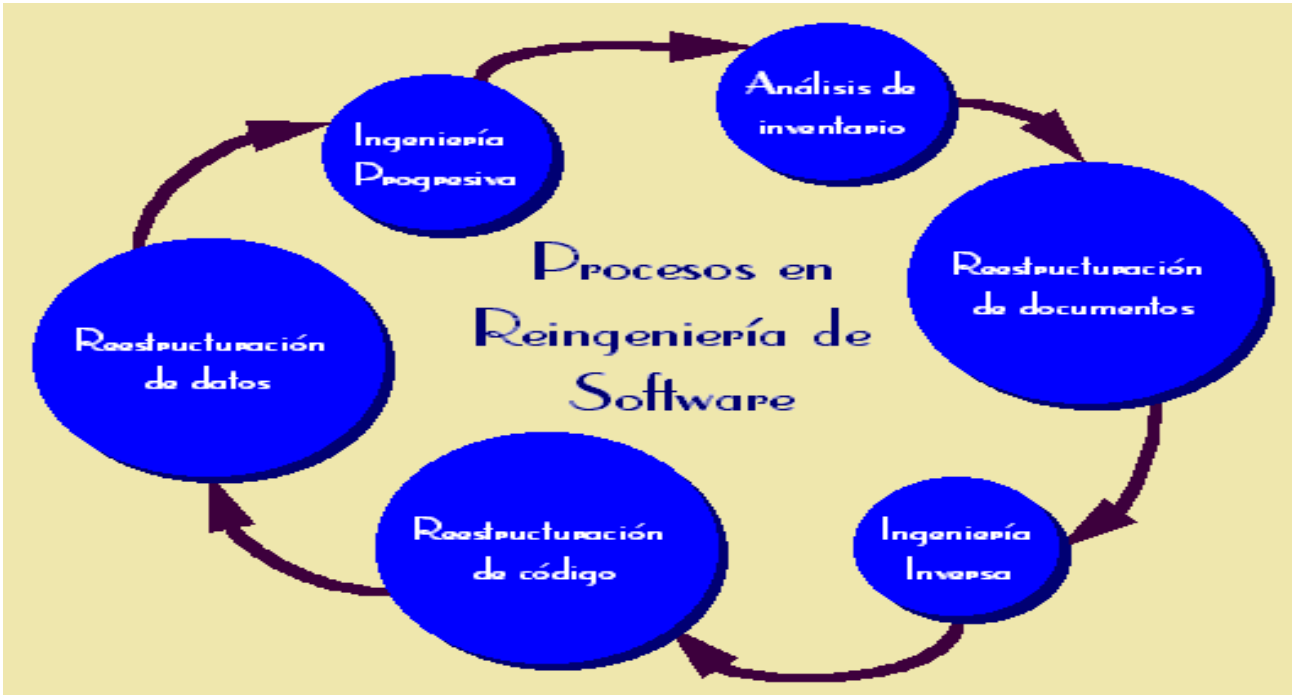


Figura.6 Procesos en Reingeniería de Software.

Anexo.7: Comparación entre herramientas.

Herramientas están M.U.G.E.N, Construct, Game Maker y RPG Maker XP	GT2D
▪ Representan los objetos visibles del juego ▪ Se colocan los objetos sobre el escenario y se le adjudican un comportamiento determinado. ▪ Tienen compatibilidad con las librerías DirectX. ▪ Disponen de menús, que definen desde los objetivos del juego a los comportamientos y controles de los personajes en según qué situaciones. Ver anexos figura 11, 12, 13, 14.	▪ Posee además un gestor de estados (máquina de estados) para la implementación de los niveles o pantallas del juego. ▪ Posee un gestor de sonidos que viene integrado con el núcleo del la herramienta. ▪ Posee un editor gráfico que permite pintar las colisiones directamente sobre los objetos. ▪ Posee un prototipo de estado genérico mediante el cual se acelera notablemente la integración final o parcial del proyecto. ▪ Posee además, algunos editores auxiliares para hacer el trabajo más fácil y eficiente como son el editor de mapa, editor de colisiones y editor de animaciones.

Figura 7: Comparación entre herramientas.

Anexo.8 Motor de desarrollo de Videojuegos RGB Maker.

Figura.8 RGB Maker.

Anexo.9 Motor de desarrollo de Videojuegos Game Maker.

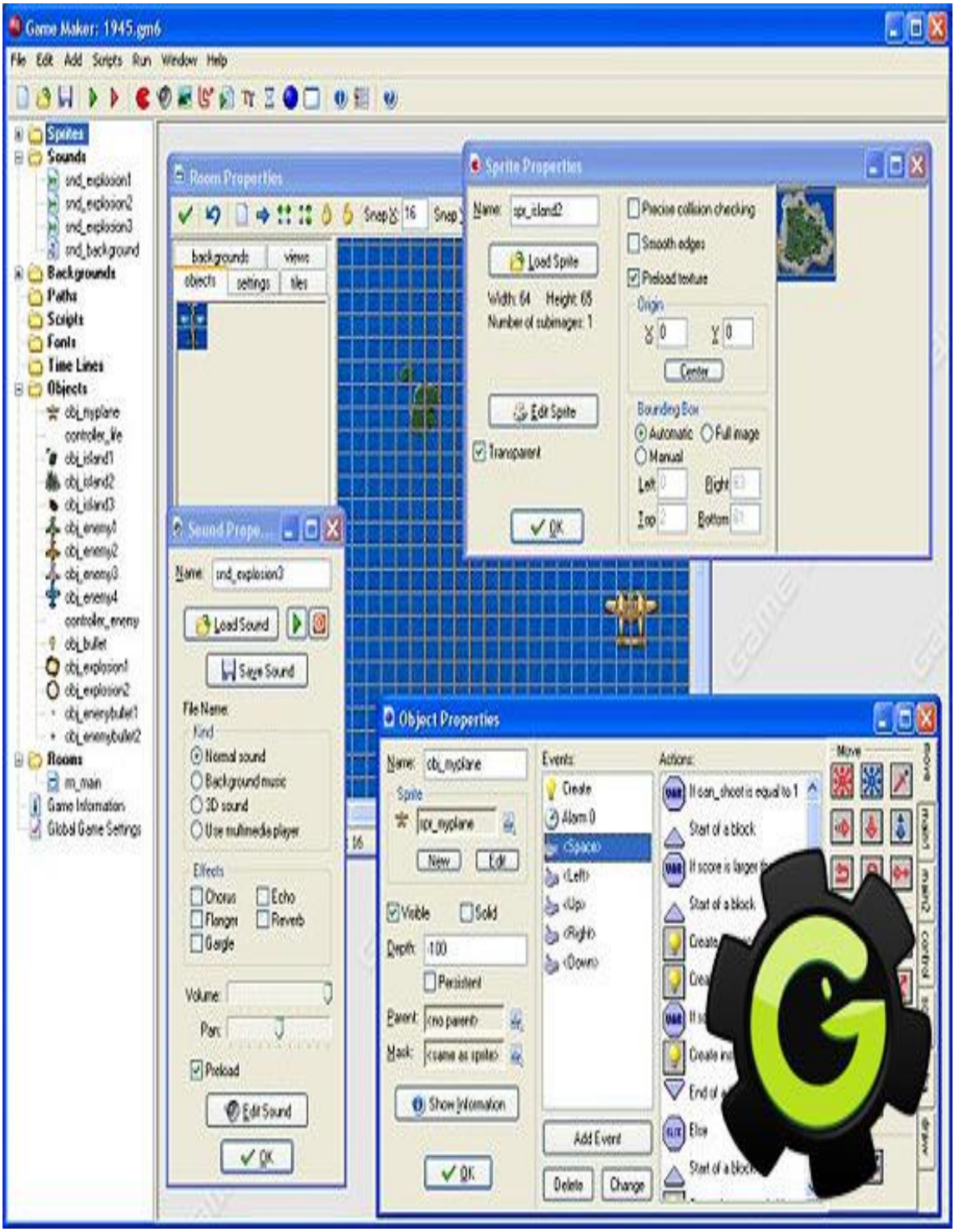


Figura.9 Game Maker.

Anexo.10 Motor de desarrollo de Videojuegos Constuct.

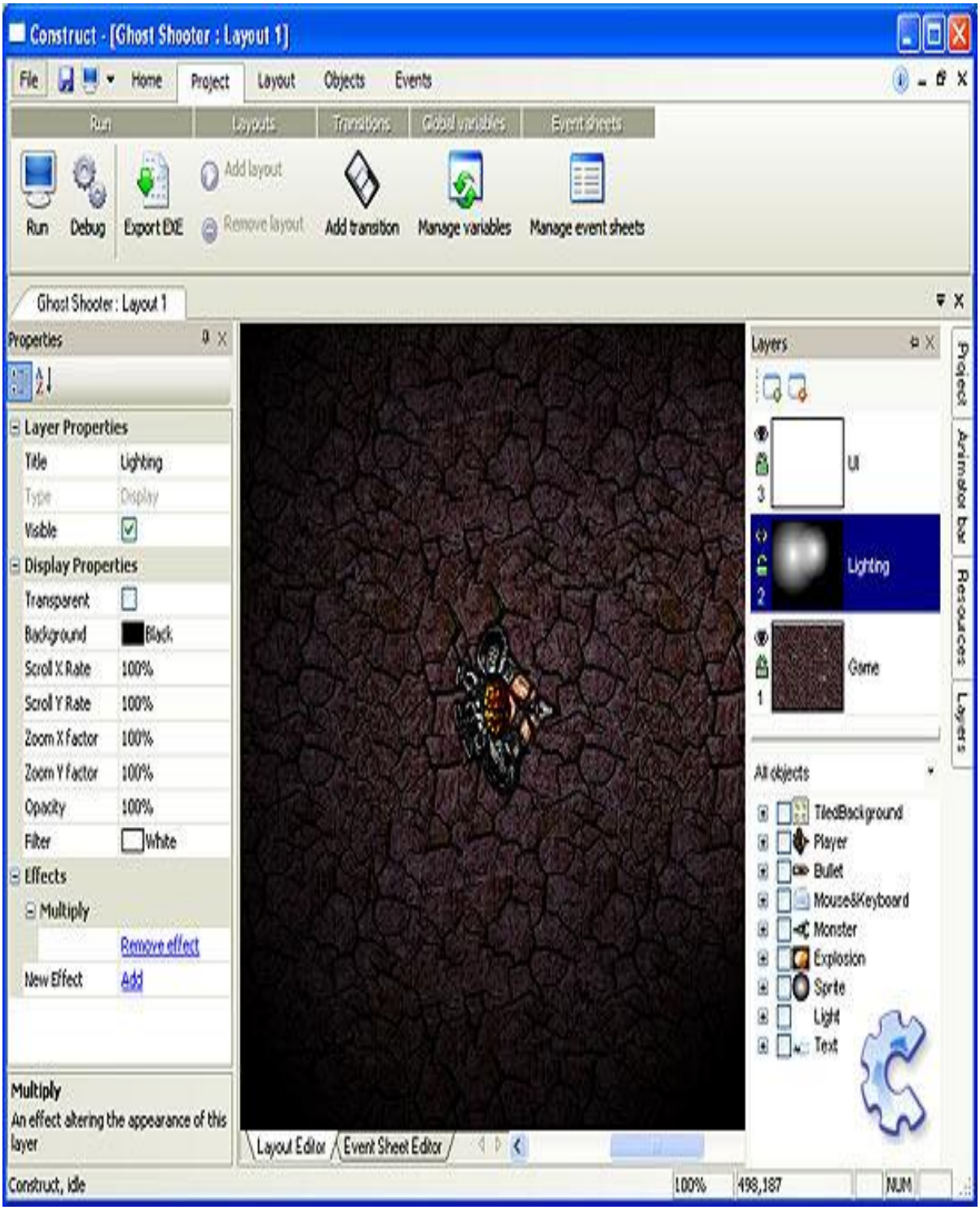
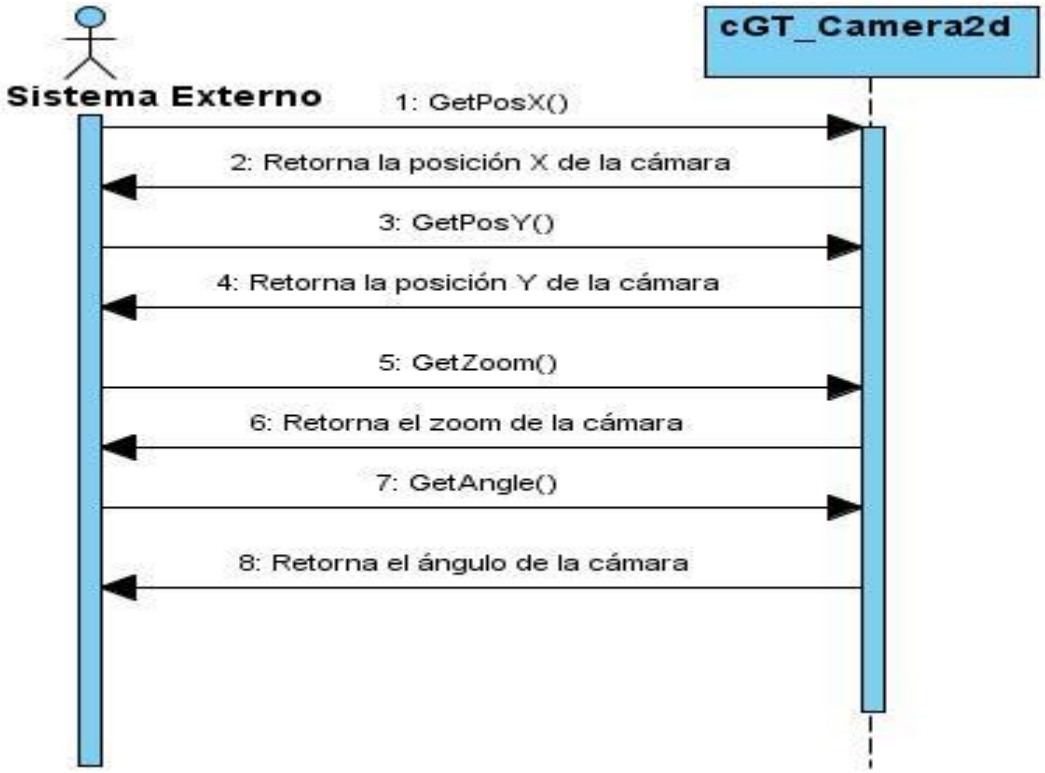


Figura.10 Constuct.

Anexo.11 Motor de desarrollo de Videojuegos Constuct MUGEN.

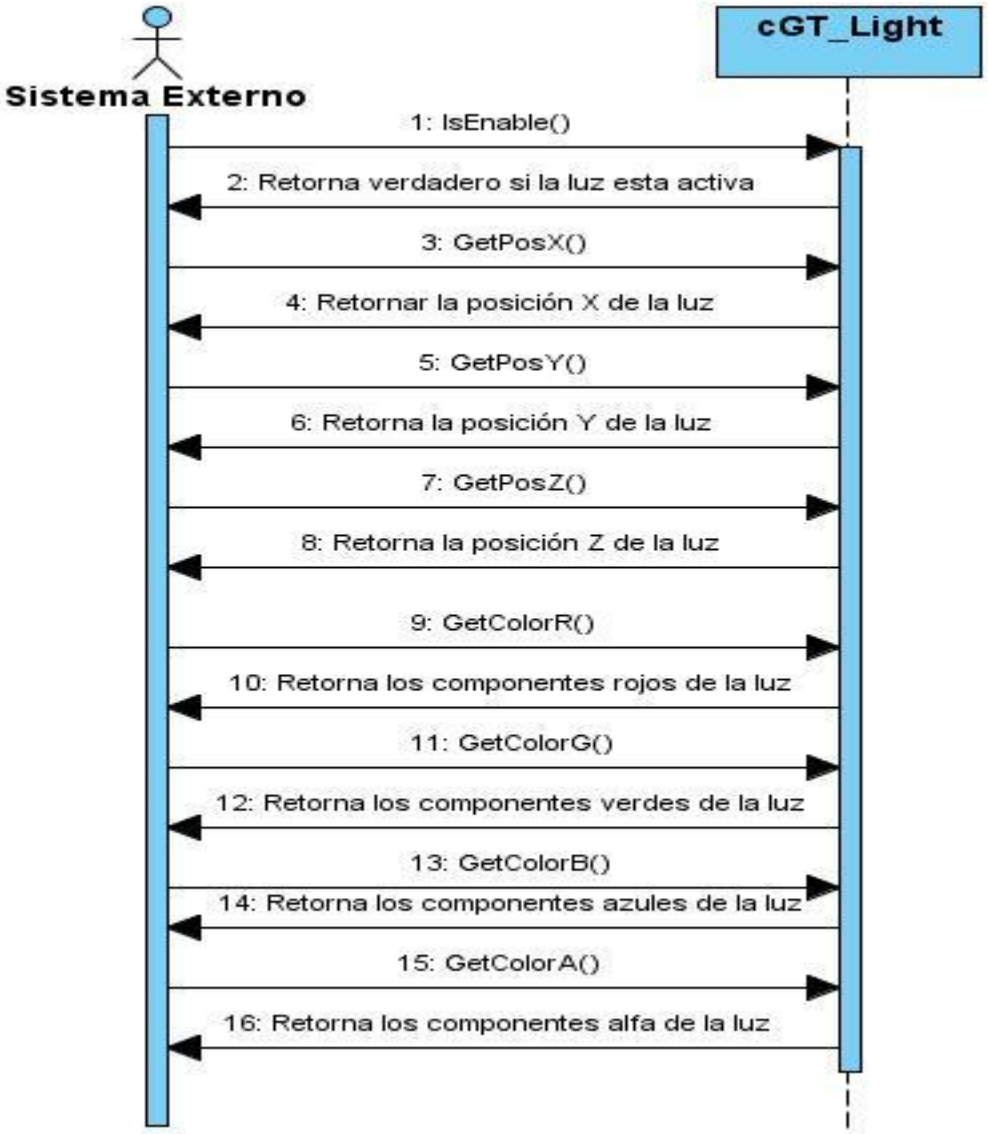
Figura.11 MUGEN.

Anexo. 12 Diagrama de Secuencia Posicionar Cámara 2D.



Anexo. 13 Diagrama de Secuencia Posicionar Cámara 3D.

Anexo. 14 Diagrama de Secuencia Cargar Luces.



Anexo 15: Descripción de la Clase Colisión.

	Nombre de la Clases: CollisionParser		
	Descripción: Clase que maneja las posibles colisiones del juego.		
	Tipo	**Nombre**	**Descripción**
Estructuras		structPixel	
		structVertex2d	
		structMatrix	
		structPoint	
		structBoundingCollision	

Anexo 16: Descripción de las Clases de Animación.

	Nombre de la Clases: GT_Animation		
	Descripción: Clase que maneja las imágenes en el juego, dándole las dimenciones de la pantalla		
	Tipo	Nombre	Descripción
Métodos	char	*GetName	Devuelve el nombre del archivo que contiene el script de animación.
	int	GetNumSequences	Retorna el total de secuencias de animación.
	int	GetNumTotalFrames	Retorna el total de frames de animación.
	GT_Image	*GetImage(int pFrame)	Retorna la ubicación del objeto, este método depende de la clase GT_Image la cual permite cargar el objeto o informa que no fue carg
	GT_Surface	*GetSurface(int pFrame)	Retorna la ubicación del objeto lo que en este caso es sobre la superficie este método trabaja con la clase GT_Surface la cual permite cargar el objeto o informa que no fue cargado.
	int	GetHighWidth(int pSequence)	Retorna el ancho máximo de la secuencia.
	int	GetHighHeight(int pSequence)	Retorna el largo máximo de la secuencia.
	int	GetNumFrames(int pSequence)	Retorna el número del frame que ha sido referenciado por la secuencia recibida por parámetro
	char	*GetName(int pSequence)	Retorna el nombre de la secuencia que ha sido recibida por parámetro
Estructuras que utiliza	vector <GT_Frame*>	*mVectorFrames	
	struct	structAnimation	Listado de las secuencias
Clases amigas		GT_AnimationManager	
		GT_Render	
		GT_Entity2dManager	

	Nombre de la Clases: GT_AnimationManager		
	Descripción: Clase que permite cargar una animación		
	Tipo	Nombre	Descripción
Métodos	bool	Init (GT_ImageManager *pImageManager, GT_SurfaceManager *pSurface	Carga una Animación.
	void	End()	Termina la Animación.
	bool	Delete (GT_Animation *pAn, bool pType)	Elimina una Animación.
	void	AddToList (GT_Animation *pNewAnimation)	Adiciona a una lista la Animación que creó.
	void	DelFromList (GT_Animation *pAn)	Elimina una Animación de la lista de animaciones.
	GT_Image*	LoadImage (char *pName)	Carga las imágenes que pertenencen a esa animación.

Anexo 17: Descripción de las Clases Entidad.

	Nombre de la Clases: GT_Entity2d		
	Descripción: Clase que maneja las entidades 2d.		
	Tipo	**Nombre**	**Descripción**
Métodos	GT_Surface	*GetSurface	si la entidad tiene una superficie asignada, retorna el punto de esta superficie.
	GT_Animation	*GetAnimation	si la entidad tiene una animación asignada, retorna el punto de esta animación.
	bool	IsShow	retorna verdadero si la entidad está siendo mostrada.
	int	GetSequence	retorna el número de secuencia que ha sido asignada a la animación
	int	GetNumReplays	retorna el número de repeticiones que tiene que hacer la animación, si el valor es igual o menor que 0, esto indica que es un bucle
	float	GetPosX	retorna la posición X de la entidad
	float	GetPosY	retorna la posición Y de la entidad
	int	GetPosZ	retorna la posición Z de la entidad
	float	GetAngleX	retorna el ángulo en el eje X de la entidad
	float	GetAngleY	retorna el ángulo en el eje Y de la entidad
	float	GetAngleZ	retorna el ángulo en el eje Z de la entidad
	float	GetScaleX	retorna la escala X de la entidad
	float	GetScaleY	retorna la escala Y de la entidad
	bool	GetBackCull	indica si la entidad está haciendo un backface donde 0 es no y 1 es si
	bool	GetMirrorX	indica si el espejo está horizontal donde 0 es no y 1 es si
	bool	GetMirrorY	indica si el espejo está vertical donde 0 es no y 1 es si
	GT_Filter	GetFilter	retorna el tipo de filtro utilizando la clase ::GT_Filter donde usa el objeto gráfico asignado a la entidad
	GT_Type	GetType	retorna el tipo, haciendo uso de la clase ::GT_Type donde usa el objeto gráfico asignado a la entidad
	byte	GetTintR	retorna donde es que está el color rojo de la entidad
	byte	GetTintG	retorna donde es que está el color verde de la entidad
	byte	GetTintB	retorna donde es que está el color azul de la entidad
	byte	GetTransparency	retorna la transparencia de la entidad
	byte	GetFadeR	retorna donde es que se desvanece el color rojo en la entidad
	byte	GetFadeG	retorna donde es que se desvanece el color verde en la entidad
	byte	GetFadeB	retorna donde es que se desvanece el color azul en la entidad
	byte	GetFadeA	retorna donde empieza a devaneserce la entidad
	GT_BlendingType	GetBlendSource	retorna el tipo de mezcla utilizando la clase ::GT_BlendingType para la fuente
	GT_BlendingType	GetDestSource	retorna el tipo de mezcla utilizando la clase ::GT_BlendingType para el destino
	float	GetHotSpotX	retorna la X del HotSpot de la entidad
	float	GetHotSpotY	retorna la Y del HotSpot de la entidad
	int	GetRegionX	retorna la posición X de la parte de arriba de la esquina izq de la región asignada de la entidad del método GT_Entity2d::SetRegion
	int	GetRegionY	retorna la posición Y de la parte de arriba de la esquina izq de la región asignada de la entidad del método GT_Entity2d::SetRegion
	int	GetRegionWidth	retorna el ancho de la región asignada por la entidad GT_Entity2d::SetRegion
	int	GetRegionHeight	retorna el largo de la región asignada por la entidad GT_Entity2d::SetRegion
	bool	IfWrap	retorna 1(verdadero) se la imagen esta repartida en los ejes X y Y
	int	GetLineX1	retorna la posición en X, del 1er punto de la primitiva de la línea
	int	GetLineY1	retorna la posición en Y, del 1er punto de la primitiva de la línea
	int	GetLineX2	retorna la posición en X, del 2do punto de la primitiva de la línea
	int	GetLineY2	retorna la posición en Y, del 2do punto de la primitiva de la línea
	int	GetRadius	retorna el radio de la primitiva utilizando la clase ::GT_REGULAR_POLY
	int	GetNumSides	retorna la cantidad de lados que tiene la primitiva utilizando la clase ::GT_REGULAR_POLY
	float	GetPolyAngle	retorna el ángulo de la primitiva utilizando la clase ::GT_REGULAR_POLY
	GT_Point	*GetPolyPoints	retorna el punto del arreglo en el cual se dibujo la primitiva utilizando la clase ::GT_POLY2D
	int	GetNumLines	retorna el número de la línea con el cual se dibujo la primitiva utilizando la clase ::GT_POLY2D
	GT_Align	GetAlign	retorna la alineación del texto utilizando la clase ::GT_Align
	int	GetCharSpacing	retorna el espacio adicional que hay entre los caracteres utilizando la clase GT_Font
	int	GetLineSpacing	retorna el espacio entre las líneas utilizando la clase GT_Font
	char	*GetText	retorna el texto de la entidad
	bool	IsShowCollisionAreas	retorna verdadero si la colisión esta siendo mostrada
	bool	IsShowGridAreas	retorna verdadero si la red de área esta siendo mostrada

	Nombre de la Clases: GT_Entity3d		
	Descripción: Clase que maneja las entidades 3d.		
	Tipo	Nombre	Descripción
Métodos	GT_3dMesh	*Get3dMesh	si la entidad tiene una superficie asiganda, retorna el punto de esa superficie.
	bool	IsShow	retorna verdadero si la entidad está siendo mostrada.
	int	GetSequence	retorna el número de secuencia que ha sido asignada a la animación.
	float	GetAnimationSpeed	retorna el valor de la velocidad de la animación.
	float	GetTransitionSpeed	retorna el valor de la velocidad de la trasición.
	float	GetPosX	retorna la posición X de la entidad.
	float	GetPosY	retorna la posición Y de la entidad.
	int	GetPosZ	retorna la posición Z de la entidad.
	float	GetAngleX	retorna el ángulo en el eje X de la entidad.
	float	GetAngleY	retorna el ángulo en el eje Y de la entidad.
	float	GetAngleZ	retorna el ángulo en el eje Z de la entidad.
	float	GetScaleX	retorna la escala X de la entidad.
	float	GetScaleY	retorna la escala Y de la entidad.
	float	GetScaleZ	retorna la escala Z de la entidad.
	bool	GetBackCull	indica si la entidad esta haciendo un backface donde 0 es no y 1 es si.
	GT_Filter	GetFilter	retorna el tipo de filtro utilizando la clase ::GT_Filter donde usa el objeto gráfico asignado a la entidad.
	byte	GetTintR	retorna donde es que está el color rojo de la entidad.
	byte	GetTintG	retorna donde es que está el color verde de la entidad.
	byte	GetTintB	retorna donde es que está el color azul de la entidad.
	byte	GetTransparency	retorna la transparencia de la entidad.
	byte	GetFadeR	retorna donde es que se desvanece el color rojo en la entidad.
	byte	GetFadeG	retorna donde es que se desvanece el color verde en la entidad.
	byte	GetFadeB	retorna donde es que se desvanece el color azul en la entidad.
	byte	GetFadeA	retorna donde empieza a devaneserce la entidad.
	GT_BlendingType	GetBlendSource	retorna el tipo de mezcla utilizando la clase ::GT_BlendingType para la fuente.
	GT_BlendingType	GetDestSource	retorna el tipo de mezcla utilizando la clase ::GT_BlendingType para el destino.
Clases amigas		GT_Entity3dManager;	

	Nombre de la Clases: GT_Entity2dManager		
	Descripción: Clase que carga las entidades 2d.		
	Tipo	Nombre	Descripción
Métodos	bool	Add(GT_Entity2d *pNewEntity2d)	Adiciona una entidad.
	bool	Add(int pLayer, GT_Entity2d *pNewEntity2d)	Adiciona una entidad.
	bool	Delete(GT_Entity2d *pEn)	Elimina una entidad.
	void	AddToList	Adicionar una entidad a una lista.
Variables	bool	m0k	
	GT_Render	*mRender	

	Nombre de la Clases: GT_Entity3dManager		
	Descripción: Clase que carga las entidades 3d.		
	Tipo	Nombre	Descripción
Métodos	bool	Add (GT_Entity3d *pNewEntity3d)	Adiciona una entidad.
	bool	Delete(GT_Entity3d *pEn)	Elimina una entidad.
	void	RenderEntities3d	Carga la entidad.
	void	AddToList	Adicionar una entidad a una lista.
Variables	bool	m0k	
	GT_Render	*mRender	

Anexo 18: Descripción de las Clases Imagen.

	Nombre de la Clases: GT_Image		
	Descripción: Clase que controla las características de las imágenes		
	Tipo	Nombre	Descripción
Métodos	bool	IsImageLoaded	controla si se está utilizando una imagen.
	float	GetWidth	retorna el ancho de la imagen.
	float	GetHeight	retorna el largo de la imagen.
	float	GetBpp	retorna la cantidad de byte por píxeles de la imagen.
	char	GetFormatInt	retorna el formato de la imagen en entero utilizando la clase ::GT_Format.
	char	*GetFormatChar	retorna el formato de la imagen en texto string utilizando la clase ::GT_Format.
	char	*GetExtension	retorna la extención que tiene la imagen cuando está almacenada en un archivo. Por ejemplo si la imagen fuera "loover.jpg" la función retornaria "jpg".
	string	*GetName	retorna el nombre en string o char.
	float	*GetPointer	retorna el punto de la memoria el cual forma la imagen. Esta función es muy utilizada para acceder directamente a la imagen, modificarla o obtenerla.
Estructura		structImage	
Clases amigas		GT_ImageManager;	

	Nombre de la Clases: GT_ImageManager		
	Descripción: Clase que controla las imágenes		
	Tipo	Nombre	Descripción
Métodos	bool	Init ()	Carga las imágenes.
	bool	Add (GT_Image *pNewImage, char *pName)	Adiciona una imagen.
	void	AddToList (GT_Image *pNewImage)	Adiciona una imagen a la lista.
	bool	Delete (GT_Image *pIm)	Elimina una imagen.

Anexo 19: Descripción de las Clases Juego.

	Nombre de la Clases: GT_Render		
	Descripción: Clase que se encarga de correr eljuego		
	Tipo	Nombre	Descripción
Métodos	int	GetViewPortX	retorna la posición X del punto de visión.
	int	GetViewPortY	retorna la posición Y del punto de visión.
	int	GetViewPortWidth	retorna el ancho del punto de visión actual.
	int	GetViewPortHeight	retorna el largo del punto de visión actual.
	char	*GetVersion	retorna la versión actual del Direct3d
	char	*GetVendor	retorna el nombre del proveedor de la tarjeta gráfica
	char	*GetRenderer	retorna el nombre de la tarjeta gráfica
	int	GetMaxTextureSize	retorna el tamaño máximo de la textura que permite la tarjeta gráfica
	LPDIRECT3D9	GetDirect3d	retorna el punto de Direct3d
	IDirect3DDevice9	*GetDevice	retorna el punto del dispositivo Direct3D
	void	*GetFpsString	retorna la cantidad de de frames por segundo en string o char que hay actualmente
	int	GetFpsInt	retorna la cantidad de de frames por segundo en int que hay actualmente
	GT_Window	*GetWindow	retorna el punto del objeto GT_Window del render donde ha sido creado
Clases amigas		GT_Entity2dManager;	
		GT_Input;	
Estructuras		StrucutFrustumPlane	
		infoStruct	

Anexo 20: Descripción de las Clases Superficie.

	Nombre de la Clases: GT_Surface		
	Descripción: Clase que controla las superficies.		
	Tipo	**Nombre**	**Descripción**
Métodos	GT_Type	GetTypeInt	retorna el tipo de la superficie, utilizando la clase ::GT_Type.
	GT_Quality	GetQualityInt	retorna la calidad de la superficie ::GT_Quality.
	int	GetNumTextures	retorna el número de la textura que la superficie esta usando.
	int	GetWidth	retorna el ancho de la superficie.
	int	GetHeight	retorna el largo de la superficie.
	int	GetBlocksX	retorna el número de bloques o cuandrado de ancho en que la superficie esta
	int	GetBlocksY	retorna el número de bloques o cuandrado de largo en que la superficie esta d
	int	GetNumBlocks	retorna el número de bloques o cuandrado en que la superficie esta dividida.
	int	GetWidthBlock	retorna el ancho de cada cuadrado el cual la superficie esta dividida.
	int	GetHeightBlock	retorna el largo de cada cuadrado el cual la superficie esta dividida.
	bool	IsHaveSurface	retorna 1 si la superficie esta cargada.
	char	*GetTypeChar	retorna el tipo de superficie en string o char.
	char	*GetQualityChar	retorna la calidad de la superficie en string o char::GT_Quality.
	bool	IsHaveGrid	retorna 1 si la superficie tiene una cuadricula asignada.
Estructura		structTextures	
Clases amigas		GT_SurfaceManager	
		GT_Render	

	Nombre de la Clases: GT_SurfaceManager		
	Descripción: Clase que cargar las superficies.		
	Tipo	**Nombre**	**Descripción**
Métodos	bool	Add (GT_Surface *pNewSurface, char *pName, GT_Type pType, GT_Quality pQuality)	Adiciona una superficie.
	bool	Delete (GT_Surface *pSu)	Elimina una superficie
	bool	Init (GT_ImageManager *pImageManager, GT_Render *pRender)	Carga una superficie