

Universidad Tecnológica Centroamericana

Facultad de Ingeniería y Arquitectura

Proyecto de Graduación

Unitec Tactics Engine:

Motor para video juegos de rol y estrategia

Previo a la obtención del Titulo

Ingeniero en Sistemas Computacionales

Presentado por:

20641045 Miguel Chicas

Asesor:

Ing. Iván de Jesús Déras

Campus San Pedro Sula

Junio, 2017

Autorización

AUTORIZACIÓN DEL AUTOR(ES) PARA LA CONSULTA, LA REPRODUCCIÓN PARCIAL O TOTAL, Y

PUBLICACIÓN ELECTRÓNICA DEL TEXTO COMPLETO DE TESIS DE GRADO.

Señores

CENTRO DE RECURSOS PARA EL APRENDIZAJE Y LA INVESTIGACION (CRAI)

San Pedro Sula

Estimados Señores:

La presentación del documento de tesis forma parte de los requerimientos y procesos establecidos de graduación para alumnos de pregrado de UNITEC.

Yo, Miguel Angel Chicas Amador, de San Pedro Sula autor del trabajo de grado titulado: Unitec Tactics Engine: Motor para video juegos de rol y estrategia, presentado y aprobado en el año 2017, como requisito para optar al título de Profesional de Ingeniero en Sistemas Computacionales, autorizo a:

Las Bibliotecas de los Centros de Recursos para el Aprendizaje y la Investigación (CRAI) de la Universidad Tecnológica Centroamericana (UNITEC), para que, con fines académicos, pueda libremente registrar, copiar y usar la información contenida en él, con fines educativos, investigativos o sociales de la siguiente manera:

Los usuarios puedan consultar el contenido de este trabajo de grado en la sala de estudio de la biblioteca y la página Web de la universidad.

Permita la consulta y la reproducción, a los usuarios interesados en el contenido de este trabajo, para todos los usos que tengan finalidad académica, ya sea en formato CD o digital desde Internet, Intranet, etc., y en general para cualquier formato conocido o por conocer.

De conformidad con lo establecido en el artículo 19 de la Ley de Derechos de Autor y de los Derechos Conexos; los cuales son irrenunciables, imprescriptibles, inembargables e inalienables. Es entendido que cualquier copia o reproducción del presente documento con fines de lucro no está permitida sin previa autorización por escrito de parte de los principales autores.

En fe de lo cual, se suscribe la presente acta en la ciudad de San Pedro Sula a los [##] días del mes de [##########] de dos mil [#########]

[Fecha de la defensa]

Miguel Chicas

20641045

Hoja de Firmas

Los abajo firmantes damos fe, en nuestra posición de miembro de Terna, Asesor y/o jefe Académico y en el marco de nuestras responsabilidades adquiridas, que el presente documento cumple con los lineamientos exigidos por la Facultad de Ingeniería y Arquitectura y los requerimientos académicos que la Universidad dispone dentro de los procesos de graduación.

Ing X

Asesor UNITEC

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Ing X | Ing X | Ing X |
| Coordinador de Terna | Miembro de Terna | Miembro de Terna |
|  |  |  |

Ing X

Jefe Academico de ¿? UNITEC

Dedicatoria

Dedico este proyecto a mi familia, quien siempre me ha apoyado en todo, y a todo el que me ha apoyado en algún momento, gracias.

Resumen Ejecutivo

El presente documento describe el desarrollo del programa de software Unitec Tactics Engine, el cual fue creado con la finalidad facilitar el desarrollo de juegos de rol de estrategia. Unitec Tactics Engine provee un sistema que comunica diferentes partes del juego, y controla las acciones de cada parte según las instrucciones provistas por el usuario. La idea principal es que los desarrolladores puedan crear y utilizar sus propios módulos para incluir funcionalidades que no se provean con los módulos originales.

Unitec Tactics Engine permite crear juego de rol de estrategia mediante la manipulación de sus recursos de datos. Si bien Unitec Tactics Engine está pensado para juegos de rol de estrategia, es lo suficientemente flexible para acomodar otros tipos de juegos mediante la modificación de sus módulos o la inclusión de módulos nuevos. También se puede cambiar comportamientos menores sin necesidad de cambiar módulos mediante la modificación de los “script”.

Unitec Tactics Engine fue desarrollado en C++ utilizando el estándar C++14 y utilizando solamente código estándar de C++, asegurando su compatibilidad con diversos compiladores a través de múltiples plataformas y es construido con CMake para asegurar la estandarización del proceso de construcción a lo largo las antes mencionadas múltiples plataformas.

Índice

Contenido

[1 Introducción 1](#_Toc488308912)

[2 Planteamiento del Problema 3](#_Toc488308913)

[2.1 Definición del problema 3](#_Toc488308914)

[2.2 Antecedentes 3](#_Toc488308915)

[2.3 Objetivos 8](#_Toc488308916)

[2.3.1 Objetivo General 8](#_Toc488308917)

[2.3.2 Objetivos Específicos 8](#_Toc488308918)

[2.4 Justificación 8](#_Toc488308919)

[3 Marco Teórico 9](#_Toc488308920)

[3.1 Video juego 9](#_Toc488308921)

[3.1.1 Juego 9](#_Toc488308922)

[3.1.2 Video juego 10](#_Toc488308923)

[3.1.3 Juego de Rol (RPG) 12](#_Toc488308924)

[3.1.4 Juego de Estrategia 15](#_Toc488308925)

[3.1.5 Juego de Rol Táctico (TRPG) / Juego de Rol de Estrategia (SRPG) / Juego de Rol de Simulación (SRPG) 16](#_Toc488308926)

[3.2 Programación 18](#_Toc488308927)

[3.2.1 Algoritmo 19](#_Toc488308928)

[3.2.2 Lenguaje de programación 20](#_Toc488308929)

[3.2.3 C 21](#_Toc488308930)

[3.2.4 C++ 21](#_Toc488308931)

[3.2.5 C++11 y C++14 28](#_Toc488308932)

[3.2.6 Lua 38](#_Toc488308933)

[3.3 Bibliotecas 39](#_Toc488308934)

[3.3.1 La Biblioteca Estándar de C++ 40](#_Toc488308935)

[3.3.2 Sol2 40](#_Toc488308936)

[3.3.3 Simple and Fast Multimedia Library 41](#_Toc488308937)

[3.4 Herramientas de manutención de software 41](#_Toc488308938)

[3.4.1 Software de construcción 41](#_Toc488308939)

[3.4.2 Compilador 44](#_Toc488308940)

[3.5 Entorno Integrado de Desarrollo 44](#_Toc488308941)

[3.5.1 QtCreator 45](#_Toc488308942)

[4 Desarrollo del proyecto 46](#_Toc488308943)

[4.1 General 46](#_Toc488308944)

[4.2 El ambiente de desarrollo 48](#_Toc488308945)

[4.3 Unitec Tactics Engine 48](#_Toc488308946)

[4.4 Sol y Lua 50](#_Toc488308947)

[4.5 Limitantes 51](#_Toc488308948)

[5 Conclusiones 52](#_Toc488308949)

[6 Recomendaciones 52](#_Toc488308950)

# Introducción

El presente documento describe en detalle los conocimientos necesarios para llevar a cabo el proyecto Unitec Tactics Engine. Entender los conceptos que este documento presenta es de gran importancia para la comprensión tanto conceptual como técnica del proyecto, sin embargo, se recomienda al lector que investigue más acerca de los temas presentados en este documento, debido a que la información sobre estos es demasiado extensa para incluirla en su totalidad en este documento.

Unitec Tactics Engine es, un motor para crear juegos, su principal trabajo es mantener los diversos sistemas de un juego comunicados para crear una experiencia cohesiva. El desarrollador del juego alimenta a Unitec Tactics Engine con instrucciones llamadas “scripts”, y con los recursos necesarios ya sea audio, video, imágenes o cualquier otro recurso que se necesite para que su juego funcione, y Unitec Tactics Engine se encarga de hacer que los sistemas que se han agregado interactúen como esta descrito en el script.

El capítulo dos se describen las razones por las cuales se llevó a cabo este proyecto, incluyendo soluciones alternativas al proyecto que si bien se asemejan no cumplen completamente con los requisitos del mismo. En este capítulo también se definen los objetivos del proyecto y el por que es importante que se haya creado.

En el capítulo tres se presenta la base teórica utilizada para poder desarrollar el proyecto. El capítulo tres incluye definiciones, ejemplos e ilustraciones que ayudan al lector a comprender los elementos que se tomaron en consideración al momento de desarrollar el proyecto. Desde que es un juego, hasta las partes del lenguaje que se utilizaron, todo esto se puede encontrar en el capítulo tres.

El capítulo cuatro describe como se llevó a cabo el desarrollo del proyecto, en este capítulo se encuentran los detalles de implementación, incluyendo el diseño del sistema como tal y las consideraciones que se deben tomar al trabajar con el proyecto. También se incluye como se debe preparar el ambiente de desarrollo, y que tomar en cuenta al momento de querer hacer funcionar el proyecto.

En el capítulo cinco se muestran las conclusiones a las que se llegaron para este proyecto. En este capítulo se observa si se cumplieron los objetivos y de que manera se cumplieron. Este capítulo muestra los resultados del proyecto de manera resumida.

En el capítulo seis se describen recomendaciones para las personas que deseen extender el proyecto. Cabe destacar que este proyecto fue diseñado con extensibilidad en mente así que se le invita a quien le interese a que lo extienda de la manera que vea conveniente.

# Planteamiento del Problema

La cantidad de motores para la creación de juegos tácticos de rol (TRPG), también conocidos como juegos estratégicos de rol (SRPG), es muy limitada. Esto no sería un problema si los motores que existen para dicha tarea fuesen lo suficientemente flexibles como para poder crear una variedad de TRPG. Unitec Tactics Engine busca resolver este problema brindando flexibilidad, tanto a la hora de crear un juego TRPG con los módulos provistos, y dando la oportunidad de extender dichos módulos en caso de ser insuficientes para dicha tarea.

## Definición del problema

Se desea crear un juego TRPG, sin embargo, la opción más viable en el momento no cumple con todas las necesidades que puedan tener diversos tipos de desarrolladores. ¿Es posible crear un programa de software que pueda simplificar la creación de este tipo de juegos y que se pueda extender para acomodar nuevas necesidades?

## Antecedentes

Existen en el mercado motores para el desarrollo de juegos TRPG y juegos de rol en general, como ya antes se había mencionado estos no resuelven el problema, se explica el por que no lo hacen a continuación.

Existen motores para la creación de juegos de rol extensibles, estos son accesibles al momento de crear juegos de rol normales, pero requieren de código externo para poder crear juegos TRPG. Algunos ejemplos de esto tipos de motor son RPG BOSS, el cual al momento de esta publicación se encuentra abandonado, Smile Game Builder que es reciente en el mercado y RPG Maker, un motor con mucha madurez, una comunidad activa y de nombre reconocido en el ámbito.





Ilustración 1. Logos de RPG Maker MV, Smile Game Builder: RPG Edition y RPG BOSS

Fuente:(“RPG Maker MV | RPG Maker | Make Your Own Video Games!”, s/f, “RPGBoss Engine”, s/f, “SMILE GAME BUILDER - a RPG creation software.”, s/f)

RPG Maker es el ejemplo más destacable de los motores para RPG, el cual en su ultima versión hasta el momento, RPG Maker MV, posee una capacidad de extensión bastante considerable, debido a que, a diferencia de sus antecesores que estaban desarrollados en RUBY, RPG Maker MV esta desarrollado en JavaScript, y permite modificar los scripts existentes del juego, así como agregar nuevos fácilmente. Un dato importante de RPG Maker MV es que su motor de dibujado esta limitado a dos dimensiones, cosa que puede ser un detrimento si se desea hacer un juego con terreno 3D y/o personajes 3D. Con una comunidad activa y más de 24 años en el mercado RPG Maker es una opción muy robusta para la tarea. La Ilustración 1 muestra un script en desarrollo cuyo objetivo es poder crear juegos TRPG en RPG Maker MV.



Ilustración 2. Ejemplo de un script en desarrollo para juegos TRPG en RPG Maker MV.

Fuente:(Lecode Works, 2017)

Entrando al mundo de motores para juegos TRPG, existen dos motores candidatos a suplir la necesidad que se ha presentado, Tethical, el cual busca emular el estilo de batalla de Final Fantasy Tactics y Battle of Westnoth, que es, además del motor, un juego completo.

Tethical fue concebido como un motor para poder imitar el estilo de batallas de Final Fantasy Tactics en el foro de Final Fantasy Hacktics, un foro dedicado a modificar imágenes originales del juego Final Fantasy Tactics en 2011 por Jean-André Santoni. El proyecto está hecho en Python con el motor Panda3D como base. Tethical provee ambientes 3D con personajes 2D como se muestra en ILUSTRACION. Debido a que su objetivo primordial es emular el estilo de Final Fantasy Tactics, no es fácilmente extensible. Tethical no se encuentra en desarrollo en el momento, teniendo su última actualización en Julio 29 del 2013.



Ilustración 3. El objetivo de Tethical es emular el estilo de batallas de Final Fantasy Tactics.

Fuente: (Jean-André Santoni, 2011)

The Battle of Westnoth es un juego TRPG de fuente abierta. El enfoque de The Battle of Wesnoth es crear un mundo alrededor de su historia, es decir que en primer lugar es un juego y no un motor para crear juegos TRPG. Aun así, debido a que el juego tiene la capacidad de utilizar contenido creado por usuarios, esto incluye “plugins” que cambian cómo funciona el juego. Otro dato destacable es que The Battle of Wesnoth es desarrollado en C++, es extensible y su comunidad es activa.



Ilustración 4. Un mapa en The Battle of Wesnoth

Fuente: (White, 2003)

## Objetivos

### Objetivo General

Desarrollar un programa de software que facilite el desarrollo de los juegos TRPG, el cual se pueda extender para acomodar las necesidades del desarrollador, ya sea mediante la modificación de scripts o la creación de nuevas piezas de software que se puedan integrar fácilmente al software.

### Objetivos Específicos

1. Comprender que se necesita para poder crear un motor de videojuegos, especialmente de los juegos tácticos de rol.
2. Desarrollar un programa de software que pueda crear la lógica necesaria y la retroalimentación adecuada para poder desarrollar juegos TRPG.
3. Establecer un flujo de trabajo apropiado para la utilización del programa de software desarrollado, tanto desde el punto de vista del desarrollador de módulos como el desarrollador de scripts.

## Justificación

Es necesaria la existencia de este programa de software, ya que, como se ha observado, en este punto en el tiempo, las alternativas para crear juegos TRPG son limitadas. La cantidad de juegos notables en este género son limitados, con el software correcto se puede crear una oportunidad para exponer al mercado más de juegos TRPG y conseguir una audiencia mayor para este tipo juegos.

# Marco Teórico

## Video juego

### Juego

Para definir un video juego se debe empezar por definir que es un juego.

Según la Real Academia Española se define un juego como : “Ejercicio recreativo o de competición sometido a reglas, y en el cuales gana o se pierde” (“juego”, s/f).

La página del diccionario Merriam-Webster define juego como una competencia física o mental conducida en acorde ciertas reglas, en la cual los participantes se encuentran directamente en oposición unos a otros.(“Definition of GAME”, s/f).

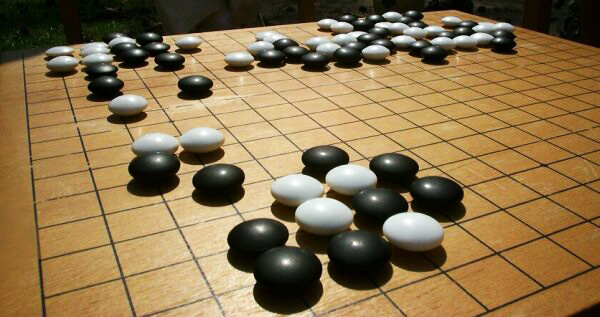


Ilustración 5. Juego de Go.

Fuente: (Donarreiskoffer, 2004)

Un juego es un sistema formal, cerrado, que subjetivamente representa un subconjunto de la realidad. Por cerrado quiere decir que es autosuficiente, el modelo del mundo recreado por el juego es internamente completo. Las reglas de un juego propiamente diseñado cubren todas las posibilidades que se puedan encontrar en el juego. Estos son formales debido a que las reglas son explicitas y transparentes. Los juegos son sistemas en el sentido completo de la palabra, una colección de partes que interactúan de diversas maneras entre sí y son objetivamente irreales por qué no recrean las situaciones físicamente, sin embargo, las situaciones son reales para quien los juega (Crawford, Chris, 1984).

Se pueden comparar los juegos con otros tipos de entretenimiento de la siguiente manera. Un juego no es un acertijo, debido a que existen agentes que interactúan directamente sobre el jugador. Existen agentes sean estos otros jugadores u otros factores que conflictúan con el objetivo de otros jugadores. A diferencia de una historia, que es lineal en naturaleza, los juegos ofrecen ramificaciones de posibilidades. También, un juego es formal en cuanto a su reglamento, mientras que a un juguete se le puede dar el uso que se desee (Crawford, Chris, 1984).

### Video juego

La Real Academia Española define videojuego como : “Juego electrónico que se visualiza en una pantalla” (“videojuego”, s/f).

El diccionario de Oxford en línea define un video juego como un juego que se juega manipulando imágenes producidas por un programa de computadora en un monitor u otro tipo de visualización. (“video game - definition of video game in English | Oxford Dictionaries”, s/f)

El primer juego de computadora del que se tiene conocimiento fue creado para la tesis del en ese entonces candidato a Doctorado de la universidad de Cambridge Alexander S. Douglas. Douglas, fascinado por la Electronic Delay Storage Automatic Calculator (EDSAC) agrego a esta un juego de tres en línea al cual llamó Noughts and Crosses, complementando su tesis sobre la interacción computador-humano. […](Goldberg, 2011).

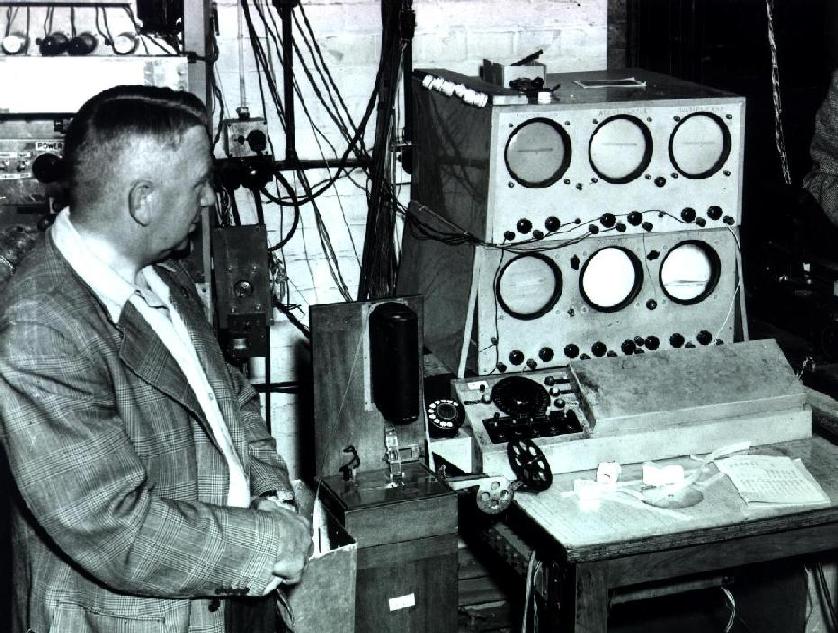


Ilustración 6. Panel de control de la EDSAC

Fuente: (Cambridge Computer Laboratory, s/f)

Otro intento temprano se le puede atribuir a William Higinbotham, quien trabajo en el Proyecto Manhattan, desarrollando los interruptores de tiempo que hicieron explotar en el momento correcto la bomba atómica. En 1958, luego de la guerra, se convirtió en el jefe de la división de instrumentación del Laboratorio Nacional de Brookhaven. Debido a que el laboratorio muestra ciertas exhibiciones al año, Higinbotham decide crear una exhibición interactiva, con la ayuda de Robert Dvorak, dan nuevo propósito a un osciloscopio y crean el juego tenis para dos. Este fue un éxito con los visitantes, apareciendo incluso el año siguiente, pero ni Dvorak ni Higinbotham, ni nadie más en ese momento pensó más de este incidente […]. (Donovan, 2010)

En febrero de 1962, “Spacewar!” fue creado por Steve Russell y sus compañeros del MIT. Este juego consistía de dos puntos verdes que flotaban en gravedad cero, disparándose en el espacio exterior. Esta experiencia inspiro a Nolan Bushnell, quien después de algunos momentos bajos en su vida decidió que la única manera de hacer dinero era ser emprendedor. En 1970 Bushnell empezó a construir su primera máquina de árcade. En junio de 1972 Bushnell fundó Atari inspirado en la idea de traer los juegos que había conocido hasta el momento a las masas (Donovan, 2010).

En 1966, Ralph Baer, un ingeniero de alto éxito, tuvo una epifanía mientras esperaba a un cliente. Baer, quien su trabajo hasta el momento involucraba radares y electrónicos anti aéreos y anti submarinos, imaginó una caja que pudiese conectarse a cualquiera de los millones de televisores estadounidenses y jugar todo tipo de juegos, la llamo inicialmente “Channel Let’s Play!” (¡Canal Juguemos!) (Donovan, 2010).

### Juego de Rol (RPG)

El término “juego de rol” (RPG) representa una variedad de juegos. Este término se ha expandido para incluir cualquier juego en el que el jugador controla un personaje en el mundo del juego y lo desarrolla a través del curso del juego. Los juegos de rol difieren de otras formas de entretenimiento debido a que sus jugadores crean sus propias experiencias a través de los personajes que representan (Tresca, 2011).

En un juego típico de Calabozos y Dragones, los jugadores parten en una aventura, donde el maestro del calabozo controla los eventos, ya sea por sí mismo o siguiendo una publicación previa. Una serie de aventuras interconectadas se le llama una campaña. El maestro del calabozo se encarga de determinar los sucesos dependiendo de las estadísticas de los personajes y de tablas de resultados. Los jugadores ruedan dados para determinar que sucede en dichos eventos. En la Ilustración 3. se puede observar un juego de calabozos y dragones llevándose a cabo.



Ilustración 7. Mapa de un juego de Calabozos y Dragones

Fuente: (Moroboshi, 2005)

En la Ilustración 4. se puede observar el manual del juego Final Fantasy, donde se describen varios aspectos del juego, incluyendo las estadísticas de los personajes, y vistas previas de los armamentos, artículos y la pantalla de batalla.



Ilustración 8. Página del manual del videojuego Final Fantasy para Nintendo Entertainment System

Fuente: (Nintendo, 1990)

Si bien es raro encontrar dos RPG que utilicen la misma fórmula, usando el concepto de familiaridad, se puede definir ciertas características que los juegos de rol poseen. La mayor parte de los RPG contiene algún tipo de combate, haciendo énfasis en la táctica a utilizar, y formas de dar soporte al combate. Tienen también lugares en donde se pueden adquirir servicios ya sea para mejorar los personajes o sus posesiones. Otras características comunes pueden ser acertijos y laberintos. Al final el único factor común que podes realmente encontrar a través de los RPGs es un sistema de estadísticas que gobierna lo que pueden hacer los personajes. […] Los RPGs necesitan el azar y un sistema de promoción formal que mejore las estadísticas de los personajes (Barton, 2008).

### Juego de Estrategia

Un juego de estrategia es aquel en el que el rol del jugador es el de un general o un ser similar a un dios que controla la acción de ciertas unidades desde la distancia. El jugador se encuentra sobre la acción, la estrategia surge en contexto de guerra, y las tácticas desde un contexto de batalla (Barton, 2008). Un ejemplo por excelencia de este género es el Ajedrez. El ajedrez es un juego de dos jugadores donde cada jugador controla dos ejércitos de 16 piezas. Se juega en un tablero de 64 casillas. Los jugadores toman turnos haciendo movimientos, el juego termina cuando un jugador inmoviliza la pieza del rey del oponente (Emms, 2003). En la Ilustración 5. se puede apreciar las diferentes piezas de ajedrez existentes.



Ilustración 9. Piezas de Ajedrez, de izquierda a derecha: Rey, Torre, Reina, Peon, Caballero y Alfil

Fuente: (Light, 2004)

Los video juegos de estrategia nacen de sus primos cercanos los juegos de mesa. Comparados con otros géneros de juegos, sus reglas suelen ser simples y se prestan para el análisis de la efectividad de las mismas. Como regla general un juego de estrategia puro tiende a ser por turnos. El jugador tiende a considerar sus movimientos posibles tratando de aprovechar al máximo sus recursos. Uno de los mayores beneficios de computarizar los juegos de estrategia es la capacidad de un ordenador de poder manejar imparcialmente conjuntos de reglas complejas que aburrirían a un ser humano tratando de seguirlas (Rollings & Adams, 2003). Un ejemplo de video juego de estrategia es Wargroove que se puede apreciar en la Ilustración 6., en el cual de uno a cuatro jugadores controlan un ejército y su capitán (“Wargroove - FAQ”, s/f).



Ilustración 10. Un mapa de Wargroove.

Fuente: (“Wargroove - Announcing Wargroove!”, 2017)

### Juego de Rol Táctico (TRPG) / Juego de Rol de Estrategia (SRPG) / Juego de Rol de Simulación (SRPG)

En un juego de rol de simulación, las batallas son el énfasis del juego. Los personajes se desarrollan de manera muy similar a cualquier juego de rol. Las batallas se desarrollan en mapas que contienen diferentes tipos de obstáculos, como elevación, edificios, agua o cualquier cosa que pueda existir en la localidad. Se puede elegir que personajes tomaran parte en la batalla y que llevaran con ellos. Es importante decidir a quién llevar a la batalla y su posicionamiento. Para ganar una batalla, es necesario cumplir con la condición de gane, normalmente esto significa inhabilitar a todos los personajes enemigos o a uno en específico. La exploración y la interacción de los personajes suele ser secundaria (Hollinger & Ratkos, 1998).

Un ejemplo moderno de juego de rol de estrategia es Disgaea 2: Dark Hero Days descrito en la ilustración 7, el combate es el centro de enfoque de este juego. En el turno correspondiente, el jugador y el oponente toman turnos para mover unidades y ejecutar acciones con todas las unidades que posea, hasta un máximo de 10 por batalla. Ambos bandos luchan hasta que todos los personajes del bando contrario sean vencidos. El mapa de Disgaea 2: Dark Hero Days consiste de una cuadricula con diferentes obstáculos y elevaciones. Los movimientos de los personajes pueden ser bloqueados por otros personajes a menos que el personaje pueda volar o haciendo que un personaje arroje a otro. Los personajes pueden actuar con diferentes tipos de armamentos cada uno con rango y propiedades propias, además de varios tipos de magia y otras habilidades (Ross, Harbick, De Angelus, Meyers, & Warriner, s/f).



Ilustración 11. Disgaea 2: Dark Hero Days, se pueden apreciar los elementos de un juego de rol y de un juego de estrategia.

Fuente: (Nipon Ichi Software, 2006)

## Programación

Programar es tomar secuencias lógicas y ejecutarlas. La programación de la computadora permite ejecutar tareas de manera eficiente, rápida y exacta. La computadora no tiene la capacidad de analizar los problemas y buscar una solución, es deber del programador analizar el problema y desarrollar la serie de pasos lógicos para poder solucionarlo. Una vez la computadora tiene esta información es posible replicar la solución de manera rápida y concisa cuando sea necesario. A la serie de pasos lógicos que solucionan el problema se le llama algoritmos (Dale & Weems, 2007).

Una metáfora que describe de una manera comprensible el desarrollo de software es la construcción de software. Construir software implica varias etapas de planeación, preparación y ejecución que varían según lo que se construye. Por ejemplo, construir una casa para un perro no requiere mucho menos planeación y es mucho más permisiva con los errores, por otra parte, construir una casa es un proceso que lleva mucha planeación, y los errores son mucho más costos. Una complejidad mayor implica un mayor costo tanto en construcción como en desarrollo de software. Otra similitud es que no es deseable volver a hacer lo que ya este hecho, a menos que sea por una razón muy específica, por desempeño, por ejemplo. Ambos se benefician de una planeación apropiada y sufren cuando se necesita un cambio estructural drástico. Finalmente, mientras más grande el proyecto, mas es el impacto de los errores, y mayor la necesidad de ser cuidadosos (McConnell, 2004).

### Algoritmo

El termino algoritmo proviene de la traducción al latín del apellido de Mohammed al-Kowarizmi, quien utilizase reglas paso a paso para operar números decimales. Es así como definimos que un algoritmo es una serie de pasos necesarios para resolver un problema. Los algoritmos no dependen del lenguaje de programación y es un aspecto muy importante de la programación (Joyanes Aguilar, 2006).

Joyanes Aguilar, (2006) describe los pasos para la resolución, mostrado en la Ilustración 8, de problemas de la siguiente manera:

1. Diseño del algoritmo, el cual no debe ser ambiguo y debe conducir a la solución del problema
2. Expresar el algoritmo como un programa, es decir codificar el algoritmo en un lenguaje de programación.
3. Ejecución y validación, la computadora procede a ejecutar el algoritmo y se verifica la salida.

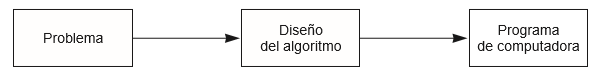


Ilustración 12. Resolución de problemas. (Joyanes Aguilar, 2006)

### Lenguaje de programación

En la computadora, los datos —cualquiera que sea su forma— se almacenan y emplean en códigos binarios, cadenas de “unos” y “ceros”. Las instrucciones y datos se almacenan en la memoria de la computadora por medio de estos códigos binarios. Si usted examinara los códigos binarios que representan instrucciones y datos en la memoria, no podría indicar la diferencia entre ellos; se distinguen sólo por la manera en que los usa la computadora. Esto hace posible que la computadora procese sus propias instrucciones como una forma de datos. (Dale & Weems, 2007, p.8)

Al principio solo era posible programar las computadoras en instrucciones integradas de la máquina, a esto se le llama lenguaje de máquina o código de máquina. EL código de máquina se introducía en código binario, esta era una tarea repetitiva y propensa a errores. Luego se introdujeron los lenguajes ensamblador para reducir los errores y facilitar la programación. Los lenguajes ensamblador coinciden uno a uno con las instrucciones de la máquina. Un paso más adelante se encuentran los lenguajes de alto nivel, los cuales se asemejan más a el lenguaje humano (Dale & Weems, 2007).

Un programa llamado compilador traduce los programas escritos en algunos lenguajes de alto nivel (C++, Pascal, FORTRAN, COBOL, Modula-2 y Ada, por ejemplo) en lenguaje de máquina. Si usted escribiera un programa en un lenguaje de alto nivel, puede ejecutarlo en cualquier computadora que tenga un compilador apropiado. Esto es posible porque la mayoría de los lenguajes de alto nivel están estandarizados, lo que significa que existe una descripción oficial del lenguaje. (Dale & Weems, 2007, p.8)

### C

C es un lenguaje de programación de alto nivel de propósito general, gracias a su eficacia y potencia es conocido como el lenguaje de programación de sistemas por excelencia. Es soportado por prácticamente todos los fabricantes de sistemas operativos, así como entidades educativas. C++, Java y C#, los tres lenguajes de programación más populares de la primera década del siglo XXI son hijos directos de C (Dale & Weems, 2007). La Ilustración 9 muestra un programa sencillo de C que muestra “Hello World” en la pantalla.

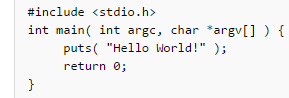


Ilustración 13. Programa “Hola Mundo” en C

Fuente: (“Hello world/Newbie - Rosetta Code”, s/f)

### C++

C++ es una extensión de C con características más potentes. Estrictamente hablando, es un superconjunto de C. Al igual que sucede con Java y C# que son superconjuntos de C++. El ANSI C estándar no sólo define el lenguaje C sino que también define una biblioteca de C estándar que las implementaciones de ANSI C deben soportar. C++ también utiliza esa biblioteca, además de su propia biblioteca estándar de clases. (Joyanes Aguilar, 2006, p.41)

#### Fundamentos

Debido a esto casi todo programa escrito para C, es correcto en C++, pero no viceversa. Si bien esto es cierto, rara vez un programa de C++ se escribe similar a C. Esto se debe a que C++ provee características propias más potentes que C (Joyanes Aguilar, 2006).

Una función en C++ es una estructura del código que permite descomponer el programa en partes. Todo programa de C++ debe tener la función “main”, con esta se inicia el programa, “main” puede invocar otras funciones las cuales al terminar retornan el control a “main” (Dale & Weems, 2007). El programa “Hola Mundo” se puede visualizar en la Ilustración 10, es uno de los programas más sencillos que existen.



Ilustración 14. Programa “Hola Mundo” de C++ utilizando la librería “iostream”

Fuente: (“Hello world/Newbie - Rosetta Code”, s/f)

Un programa de computadora opera sobre datos, en C++ estos datos deben ser de un tipo específico, algunos de estos tipos son tan comunes que ya se encuentran definidos previamente en el lenguaje, otros tipos se pueden definir por el usuario, estos son llamados tipos definidos por el usuario (Dale & Weems, 2007).

C++ fue diseñado para soportar la abstracción de datos, programación orientada a objetos y programación genérica, pero no está diseñado para forzar ningún tipo de estilo. Cada nombre y expresión tiene un tipo que determina que operaciones se pueden hacer con ese tipo de dato. C++ posee un conjunto de instrucciones para selección y para ciclos. C++ tiene soporte para apuntadores y para arreglos. C++ provee un mecanismo para poder agrupar datos y funciones relacionadas en nombres de espacio (namespaces) separados. También soporta la compilación separada, es decir que se pueden compilar partes del código por separado. C++ posee un conjunto de módulos de excepción, aunque se prefiere el manejo de errores a usar este. C++ soporta tipos concretos y abstractos y funciones virtuales y jerarquía de datos (Stroustrup, 1997). Los tipos fundamentales de C++ se pueden observar en la Tabla 1.

Tabla 1. Tipos fundamentales C++

|  |  |  |
| --- | --- | --- |
| Tipo | Ejemplos | Descripción |
| Booleano | bool | Un tipo booleano, este puede tener solamente dos valores, true (verdadero) y false (falso) |
| Carácter | char, wchar\_t, | Un tipo para describir un carácter. |
| Integrales | int, short, long | Un tipo para describir números enteros. |
| Punto Flotante | float, double, long doublé | Un tipo que representa un numero en formato de punto flotante. |
| Vacío | void | Un tipo que indica que una función no tiene valor de retorno. |

Fuente: (Stroustrup, 1997)

#### Apuntadores

Los apuntadores son tipos que contienen la dirección de una variable, el objeto mas pequeño que se puede apartar memoria para es un char. Un apuntador se denota con un asterisco “\*”, el cual también se utiliza para dereferenciar apuntadores (Stroustrup, 1997). Se puede ver en la Ilustración 11 un ejemplo de apuntadores.

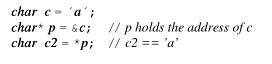


Ilustración 15. Apuntadores en C++

Fuente: (Stroustrup, 1997)

#### Arreglos

Para un tipo T, T[tamaño] representa un arreglo con una cantidad de elementos igual a tamaño, y se pueden acceder desde el índice 0 al índice tamaño – 1 (Stroustrup, 1997). Un ejemplo se puede ver en la Ilustración 12.

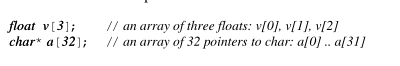


Ilustración 16. Arreglos en C++

Fuente: (Stroustrup, 1997)

#### Referencias

En C++ existen las referencias, que son un nombre alternativo para un objeto, normalmente se utilizan para argumentos en funciones y valores de retorno. La notación x& significa una referencia a x. Las referencias deben ser inicializadas al momento de ser creadas, la excepción a esto es si se utiliza la palabra reservada “extern”. Un dato importante que tener en cuenta es que una referencia no es un objeto manipulable (Stroustrup, 1997). Un ejemplo de referencia se puede observar en la Ilustración 13.

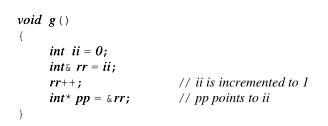


Ilustración 17. Referencia en C++

Fuente: (Stroustrup, 1997)

#### Funciones

La manera típica de hacer algo en C++ es llamar una función que se encargue de ello. El usuario define la función y luego se puede utilizar. La declaración de la función le el nombre a la función y el número y tipo de argumentos que esta acepta. Se puede observar un ejemplo declaración de funciones en la Ilustración 14.



Ilustración 18. Declaración de funciones en C++

Fuente: (Stroustrup, 1997)

#### Tipos definidos por el usuario

El objetivo del concepto de clases de C++ es proveerle ar programador con una herramienta para crear nuevos tipos que puedan ser utilizados a tal conveniencia como los tipos nativos del lenguaje. Un tipo es una representación concreta de un concepto. Por ejemplo, el tipo “float” con sus operaciones aritméticas provee un concreto aproximado del concepto de un número real (Stroustrup, 1997). Un ejemplo de la declaración de una clase puede verse en la Ilustración 15.

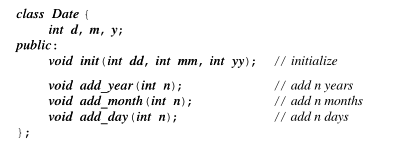


Ilustración 19. Declaración de una clase en C++

Fuente: (Stroustrup, 1997)

C++ propone que una clase es un tipo definido por el usuario, y que las clases deberían modelar conceptos en ambos el mundo del programa y del desarrollador. Debido a que un concepto no existe en aislamiento, debe existir una manera de agregar relaciones entre clases (Stroustrup, 1997).

#### Clases derivadas

Uno de los conceptos básicos que C++ busca modelar es en el que uno o varios conceptos pueden generalizarse bajo otros conceptos, es decir los conceptos heredan ciertas características de objetos más simples o generales. A esto se le llama herencia. Las clases derivadas son un super-conjunto de la clase base, conteniendo todo lo que la clase base contiene y agregando sus propias características.

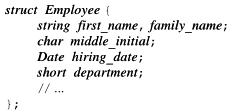
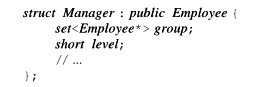
 

Ilustración 20. De izquierda a derecha, la clase base Employee y la clase derivada Employee

Fuente: (Stroustrup, 1997)

#### Sobrecarga de operadores

C++ soporta un concepto llamado sobrecarga de operadores, esto permite agregar comportamiento personalizado a los operadores, dando la oportunidad de utilizar los operadores para dar al desarrollador una manera más conveniente de notación y manipulación de los objetos. La Ilustración 17 muestra cómo se pueden sobrecargar los operadores, mientras que la Ilustración 18 muestra que operadores se pueden sobrecargar.

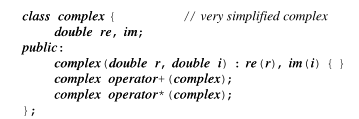


Ilustración 21. Sobrecarga de operadores en C++

Fuente: (Stroustrup, 1997)

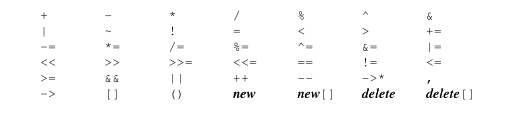


Ilustración 22. Operadores que se pueden sobrecargar C++

Fuente: (Stroustrup, 1997)

#### Plantillas

Conceptos independientes deberían ser representados independientemente y combinados solamente cuando se necesite. Si este principio es violado se crean dependencias innecesarias o conceptos no relacionados terminan juntos. Las plantillas proveen una manera de mantener una alta gama de conceptos separados y fácilmente combinarlos. Las plantillas proveen soporte directo para programación genérica, gracias a que pueden utilizar tipos como parámetros (Stroustrup, 1997).

Una clase plantilla se declara con el prefijo “template” seguido de “<”, se procede a agregar los parámetros que se utilizaran para la plantilla, cabe destacar que el tipo de los parámetros no está limitado a tipos, sin embargo debe de ser constante, luego se cierra con “>” y se procede a declarar la clase plantilla.



Ilustración 23. Declaración de una clase plantilla

Fuente: (Stroustrup, 1997)

### C++11 y C++14

Hay muchas herramientas nuevas en C++11 y C++14. Las declaraciones “auto”, ciclos “for” basados en rango, expresiones lambda, referencias a los valores de derecha (rvalue), “nullptr” para indicar apuntadores nulos en lugar de 0, enumeraciones con alcance, mover en lugar de copiar. La característica más predominante de C++11 es probablemente las semánticas de movimiento. El fundamento de las semánticas de movimiento que distingue los valores de derecha (rvalue) de los valores de izquierda (lvalue). Los rvalue conceptualmente se refiere a objetos que pueden ser movidos mientras que en general los lvalue no pueden ser movidos (Meyers, 2015).

El concepto de “auto”, aunque simple, es más sutil de lo que se pueda pensar. Las variables “auto” obtienen su tipo del inicializador, esto quiere decir que siempre deben ser inicializadas. Debido a su naturaleza de deducción de tipos, esta puede representar tipos que solo el compilador conoce. En C++14 “auto” puede incluso ser utilizado como parámetro de expresiones lambda. Otra ventaja de “auto” es que debido a que utiliza el tipo preciso del objeto, se evitan errores cuando diferentes implementaciones de tipos que son parecidos no coinciden en sistemas diferentes (Meyers, 2015). La Ilustración 20 muestra ejemplos del uso de “auto”.

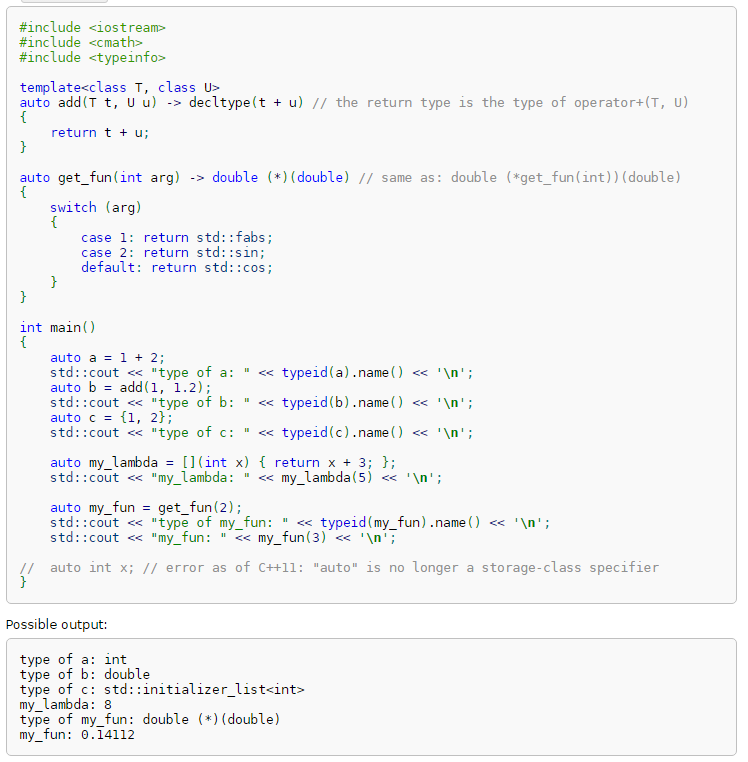


Ilustración 24. Ejemplo de utilización de auto

Fuente: (“auto specifier (since C++11) - cppreference.com”, 2017)

C++11 introduce inicialización uniforme, esta inicialización utiliza llaves para expresar la sintaxis de inicialización. Se puede utilizar para la inicialización de miembros de datos estáticos, para inicializar con un conjunto de datos un objeto y para inicializar objetos no copiables (Meyers, 2015). La Ilustración 21 muestra ejemplos de inicialización uniforme.

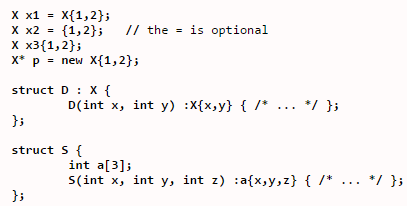


Ilustración 25. Inicialización uniforme

Fuente:(Stroustrup, 2012)

En versiones anteriores de C++, se utilizaba “0” o el macro “NULL” para indicar un apuntador nulo. Los compiladores de C++ toman esto como un apuntado nulo solamente en última instancia, así que C++11 introduce el valor “nullptr”. El valor “nullptr” es de tipo “std::nullptr\_t”, el cual implícitamente se convierte a todo tipo de apuntador bruto, de esta manera se evita que se utilicen sobrecargas equivocadas a la hora de utilizar nulo como parámetro. Otra ventaja es que al momento de intentar deducir un tipo se evita la ambigüedad que proveería utilizar “0” o “NULL”(Meyers, 2015).

C++11 ofrece una alternativa a los “typedef”, se le llaman “aliases”. Para declarar un alias se utiliza la palabra reservada “using” seguido del nombre del alias, un símbolo de igual y el tipo al que se le desea dar el alias. Una ventaja del “alias” sobre el “typedef” es que el alias puede usarse con plantillas directamente sin necesidad de una solución alternativa (Meyers, 2015). La Ilustración 22 muestra uno de los usos de “using” como alias.

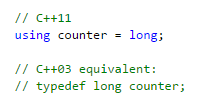


Ilustración 26. Using como equivalente a typedef

Fuente: (Microsoft, 2017a)

Como regla general, declarar un nombre dentro de llaves limita su visibilidad, este no es el caso con las enumeraciones normales de C++, debido a esto se creó la enumeración con alcance, comúnmente llamada clases de enumeración. Las enumeraciones con alcance contaminan menos el entorno de desarrollo y no se pueden convertir implícitamente a tipos integrales, lo cual hace que la intención del desarrollador se mantenga al solamente poder compararse con el mismo tipo. Un aspecto nuevo par ambos tipos de enumeraciones es la capacidad de declararlas previo a definirlas, en el caso de las enumeraciones sin alcance se debe de especificar que tipo interno se desea que sea (Meyers, 2015). La declaración de una enumeración con alcance puede observarse en la Ilustración 23.

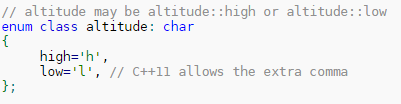


Ilustración 27. Declaración de una enumeración con alcance

Fuente: (“enumeration declaration - cppreference.com”, 2017)

Las funciones borradas son otro aspecto de C++11, su propósito es evitar que se llamen funciones particulares. Se puede borrar una función simplemente igualándola a “delete”. En C++98 se colocaba la función como privada y se igualaba a 0, hacer esto generaba un error al querer utilizar la función al momento de hacer el enlazamiento, utilizando “delete” se puede detectar en tiempo de compilación. Otra ventaja de “delete” es que se puede utilizar para eliminar cualquier función, incluyendo sobrecargas especificas (Meyers, 2015). Un ejemplo del borrado de funciones puede verse en la Ilustración 24.



Ilustración 28. Ejemplo de funciones borradas

Fuente: (“Declaring functions - cppreference.com”, 2017)

C++11 agrega la palabra reservada “override” para poder indicar al compilador que la función que se está declarando es un reemplazo de la función de la base. Sin esto es posible crear funciones en una clase derivada muy parecidas a la original pero no exactamente iguales, en cuyo caso no actúa como se piensa que debería actuar la función (Meyers, 2015). Se puede ver un ejemplo del uso de “override” en laIlustración 25.

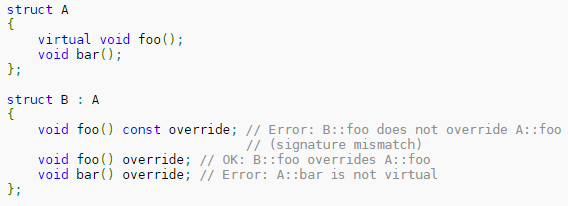


Ilustración 29. Utilizacion de “override”

Fuente: (“override specifier (since C++11) - cppreference.com”, 2017)

Los apuntadores en C++ son un concepto muy poderoso, y así como es de poderoso puede ser peligroso si no se tratan con cuidado. C++11 presenta la oportunidad de mantener el poder de los apuntadores en bruto con los apuntadores inteligentes. Los apuntadores inteligentes saben a quién pertenece el objeto apuntado y cuando y como destruirlo. Entre los apuntadores inteligentes existen los “auto\_ptr”, los “unique\_ptr”, los “shared\_ptr” y los “weak\_ptr” (Meyers, 2015). Los apuntadores inteligentes reducen la complejidad de manejar el tiempo de vida de los objetos como se muestra en la Ilustración 26.

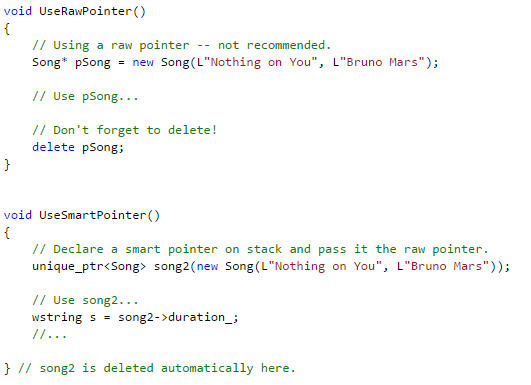


Ilustración 30 Manejo del ciclo de vida de un objeto con apuntadores en bruto y con apuntadores inteligentes.

Fuente: (Microsoft, 2017e)

Los “auto\_ptr” son una implementación previa a C++11 y se deben evitar, debido a que los otros apuntadores inteligentes hacen el trabajo de “auto\_ptr” mucho mejor. Cuando se necesita un apuntador inteligente, la primera opción es el “unique\_ptr”. El “unique\_ptr” tiene un tamaño por defecto igual al del apuntador en bruto para la mayor parte de las operaciones. El “unique\_ptr” tiene la peculiaridad de que no puede ser copiado, esto se debe a que al copiarlo el nuevo “unique\_ptr” apuntaría al mismo lugar que el original. Al momento de ser destruido un “unique\_ptr” no nulo destruye a lo que sea que este apuntando en ese momento. Se debe tener en cuenta que si bien por defecto un “unique\_ptr” es del mismo tamaño que un apuntador en bruto, el “unique\_ptr” crece cuando entran en juego destructores personalizados (Meyers, 2015). La Ilustración 27 muestra cómo crear un “unique\_ptr” y transferir la posesión del objeto a un nuevo “unique\_ptr”

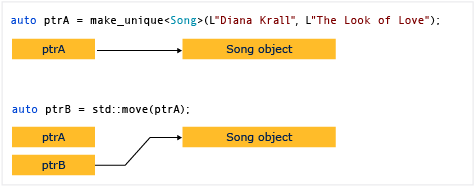


Ilustración 31. Transferencia de posesión entre dos instancias de “unique\_ptr”

Fuente: (Microsoft, 2017c)

La recolección de basura es una de las conveniencias de la programación moderna, pero suele ocurrir en momentos poco oportunos. C++11 introduce un concepto que permite que la colección de basura sea automática y a la vez predecible, los apuntadores compartidos o “shared\_ptr”. Como su nombre lo indica los “shared\_ptr” son apuntadores que no tienen posesión del objeto al que apuntan, pero entre todos los que apuntan al mismo objeto ayudan a asegurar su destrucción cuando ya no es necesario. A diferencia de los “unique\_ptr”, los “shared\_ptr” suelen tener un apuntador hacia el objeto y uno hacia el conteo de referencias de ese objeto, el cual es dinámicamente reservado, véase la Ilustración 28. El costo de reservar memoria dinámicamente par aun “shared\_ptr” puede ser evitado utilizando “std::make\_shared”, pero no siempre se puede utilizar. Cabe destacar que es más eficiente mover un “shared\_ptr” que copiarlo (Meyers, 2015). Las formas de inicializar un “shared\_ptr” se pueden observar en la Ilustración 29.

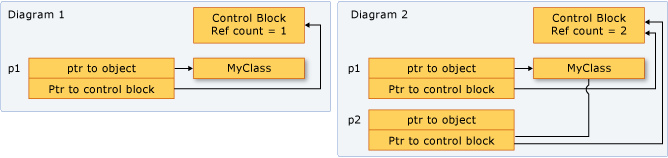


Ilustración 32. Varios “shared\_ptr” que apuntan al mismo objeto.

Fuente: (Microsoft, 2017b)

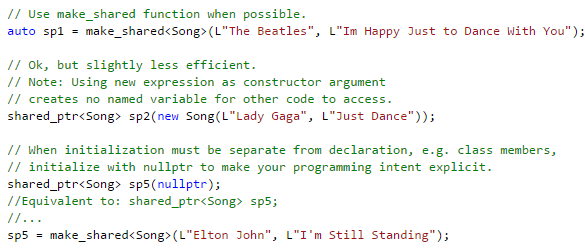


Ilustración 33. Formas de inicialización de un “shared\_ptr”

Fuente: (Microsoft, 2017b)

El cuarto tipo de apuntador inteligente es el “weak\_ptr”. Este tipo de apuntador inteligente puede quedar colgando, es decir que el apuntador apunte a un objeto invalido, esto se debe a que su uso es diferente al de los que se han mencionado con anterioridad. A diferencia de un “shared\_ptr”, el “weak\_ptr” no aumenta el conteo de referencias, debido a esto una vez que un objeto no contenga más referencias se elimina, dejando al “weak\_ptr” colgado, de esta manera el “weak\_ptr” no interfiere con el ciclo de vida del objeto, pero si se quiere utilizar el objeto se puede obtener un “shared\_ptr” a través de un “weak\_ptr”. Usos potenciales de este tipo de apuntador incluyen listas de observadores, prevención de ciclos de “shared\_ptr” y hacer caches (Meyers, 2015).

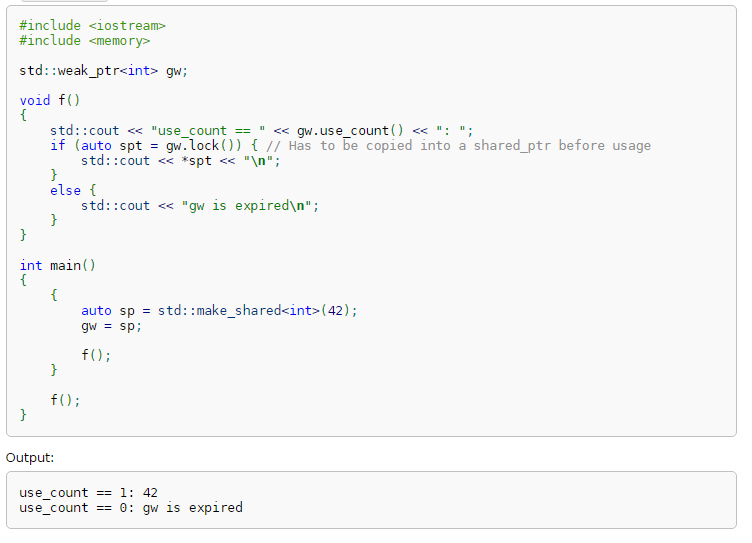


Ilustración 34. Ejemplo de utilización de un “weak\_ptr”.

Fuente: (“std::weak\_ptr - cppreference.com”, 2017)

Las expresiones lambda son nuevas en C++11, estas proveen una forma muy conveniente de crear objetos de función. Una expresión lambda es esencialmente eso, una expresión. Una cerradura es el objeto creado por una lambda, y dependiendo del tipo de captura de la lambda, puede contener copias o referencias del ambiente en que se creó la lambda. En general se utilizan para crear argumentos para funciones que aceptan funciones como parámetros, se puede ver esto en la Ilustración 31 (Meyers, 2015).

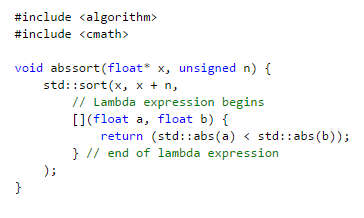


Ilustración 35. Expresión lambda como parámetro de la función “std::sort”

Fuente: (Microsoft, 2017d)

### Lua

Lua es un lenguaje de scripting diseñado específicamente para integrase con software escrito en C++ y C. Su nombre significa Luna en portugués. Lua ofrece estructuras dinámicas, distancia del hardware y facilidad para hacer pruebas. La mayor parte del poder de Lua proviene de sus bibliotecas, esto se debe a que Lua fue diseñado con extensibilidad en mente. Mas importante que su extensibilidad, Lua soporta un desarrollo basado en componentes, los componentes son compilados en lenguajes estáticamente tipificados y Lua funciona como el pegamento que los une (Ierusalimschy, 2016).

La principal y única estructura existente en Lua son las tablas. Las tablas son poderosas y se utilizan para representar arreglos, conjuntos, archivos y otros tipos de estructuras de datos simples. Incluso se utilizan para representar paquetes y objetos. En Lua las tablas no son valores ni variables, son objetos. La Ilustración 36 muestra cómo se inicializan las tablas en Lua.

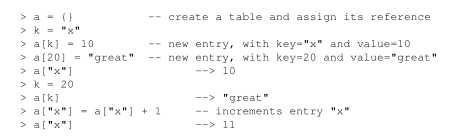


Ilustración 36. Formas de inicializar tablas en Lua

Fuente: (Ierusalimschy, 2016)

En Lua las funciones son el mecanismo principal de abstracción de enunciados y expresiones. Las funciones pueden llevar a cabo una tarea específica o computar un valor y retornarlo. Lua puede utilizar funciones definidas tanto en Lua como en C, o cualquiera que fuese el lenguaje de programación del programa anfitrión. Las funciones pueden o no llevar paréntesis si tienen al menos un argumento, si no tienen ningún argumento entonces se debe denotar con paréntesis. (Ierusalimschy, 2016).

Por defecto las variables de Lua son globales, todas las variables locales deben declarase como locales, esto hace que su alcance este limitado al bloque donde son declarados. Un bloque es el cuerpo de una estructura de control de flujo(Ierusalimschy, 2016).

## Bibliotecas

Las bibliotecas proveen funciones predefinidas para la programación modular. Las bibliotecas permiten a los desarrolladores escribir código más corto y modular. Estas también permiten la fácil actualización del programa al poder cambiar solamente el módulo que se necesite (Simsek, 2004). Las bibliotecas son diseñadas para empaquetar funcionalidades similares en una sola unidad y pueden ser dinámicas o estáticas. Las bibliotecas estaticas se compilan en como parte del programa que la utilice, mientras que la biblioteca dinámica se carga en memoria cuando el programa es cargada (Jones, 2008).

### La Biblioteca Estándar de C++

La librería estándar de C++ provee soporte para las características del lenguaje como ser manejo de memoria e información en tiempo de ejecución de los tipos. Provee información sobre aspectos definidos por la implementación del lenguaje, como por ejemplo el flotante mas grande. Provee funciones que no pueden ser implementadas óptimamente para cada sistema, como por ejemplo la función sqrt, también provee utilidades que el desarrollador puede utilizar y confiar en su portabilidad, como listas, mapas, funciones de ordenamiento y flujo de entradas y salidas. También provee convenciones a seguir para la extensión de las utilidades que provee y el fundamento para las otras bibliotecas (Stroustrup, 1997).

La biblioteca estándar de C++ provee contenedores tales como list y queue. También tiene utilidades generales para el manejo de funciones, tiempo y memoria, soporte para iteradores, algoritmos genéricos estándar, una colección para el diagnóstico de la aplicación, manejo de cadenas, manejo de entradas y salidas, globalización, soporte para el lenguaje y soporte para datos numéricos entre otras cosas.

### Sol2

Sol es una biblioteca de C++ que une Lua con C++, soporta las versiones de Lua mayores 5.1. Sol está diseñado para ser de fácil uso y fácil implementación en un proyecto. Soporta la recuperación y la colocación de múltiples tipos. Soporta la unión con expresiones lambdas y funciones. Un tipo intermediario para revisar la existencia de variables. Sol tiene un API simple que completamente abstrae el API de Lua, incluyendo la habilidad de manejar errores. Manejo de tablas con el operador operator[], representación de datos de usuario y limpieza de datos garantizada. También soporta tablas, tablas anidadas e iteración de tablas (Derpstorm, el 7 de enero de 2016/2017).

### Simple and Fast Multimedia Library

SFML es una biblioteca que provee una interfaz simple a varios de los componentes del ordenador, para facilitar el desarrollo de aplicaciones multimedia y juegos. Está compuesta de cinco módulos, estos son: El módulo de system, el módulo de window, el módulo de graphics, el módulo dee audio y el módulo de network. SFML es multiplataforma, puede correr en Windows, Linux y OS X, con planes para incluir Android e iOS.

## Herramientas de manutención de software

### Software de construcción

Los programas de software se construyen a partir de sus archivos fuente, también conocido como código fuente. Una herramienta de construcción de software utiliza el código fuente y pasos predefinidos para ejecutar herramientas para crear el programa. A los archivos que se utilizan para tal tarea se les conoce como archivos de construcción (build files). Las herramientas de construcción deben poder construir el programa y conocer el orden en que se deben construir las partes y poder reconstruirlas independientemente de plataforma. Las herramientas de construcción generan ejecutables que pueden ser ejecutados en otras plataformas, a esto se le conoce como compilación cruzada (cross-compiling) (Doar, 2005).

Escribir código fuente es sencillo hasta que la cantidad de código empieza a crecer. Para mantener manejable el código se divide en partes, hacer esto crea dependencias y es en ese momento que pueden surgir problemas. Para reducir las dependencias se han creado diversos mecanismos, como ser archivos de encabezado, encapsulamiento e interfaces. El orden en que se construye el programa juega parte importante en la construcción satisfactoria de un programa, mantener este orden no es tarea fácil, pero las herramientas de construcción de software nos ayudan a hacer justamente esto, la tabla 1 ilustra que acciones se deben tomar al haber cambios. (Doar, 2005).

Tabla 2. Razones por las cuales un proyecto debe ser reconstruido

|  |  |
| --- | --- |
| Tipo de cambio | Acción que debe tomar la herramienta |
| Nuevos archivos agregados | El desarrollador debe agregar los archivos nuevos a los archivos de construcción y revisar si están correctos. |
| El contenido de los archivos cambió | Reconstruir el archivo; detectar si la compilación falo o fue un éxito. |
| Un archivo depende de un archivo que cambió. | Reconstruir los archivos afectados. |
| Un archivo ahora depende de un archivo que ya existe. | Reconstruir el árbol de dependencias. Verificar si el archivo puede ser encontrado correctamente. |
| Un archivo ahora depende de un nuevo archivo. | Reconstruir el árbol de dependencias. Verificar la existencia del archivo nuevo o si será creado como parte de la construcción. |
| Un archivo ya no depende de otro. | Reconstruir el árbol de dependencias. Verificar si el archivo antiguo causa algún problema o si debería ser eliminado. |
| Un archivo ahora depende de un archivo generado. | Asegurar que el árbol de dependencias cause la creación del archivo generado antes de que se necesite. |
| Un archivo ahora depende de un archivo generado cuyo archivo fuente ha sido cambiado. | Volver a generar los archivos generados necesarios. |

(Doar, 2005)

#### Make

Make es la herramienta de construcción original, y la más popular con desarrolladores de C y C++. Las dependencias en make se especifican explícitamente en archivos llamados convencionalmente “Makefile” o “makefile” y son escritos en su propio lenguaje de make (Doar, 2005).



Ilustración 37. Un “makefile” sencillo

Fuente: (Maxwell, s/f)

#### CMake

Cmake es una familia de herramientas de fuente abierta y de plataforma cruzada diseñada para construir, probar y empaquetar software. Es utilizada para controlar el proceso de compilación del software utilizando una plataforma simple y archivos de configuración independientes del compilador para generar “makefiles” nativos y espacios de trabajo que pueden utilizar el compilador de la elección del usuario (“cmake.org”, consultado el 12/6/2017).

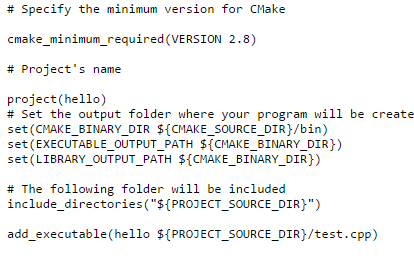


Ilustración 38. Archivo “CMakeLists.txt” de un proyecto

Fuente: (Nguyen, 2014)

### Compilador

Compilar es el proceso en el que el código fuente de un programa se traduce a objetos, normalmente el objeto se encuentra en código de máquina. Un compilador es un programa que hace esto, el resultado de una compilación suele ser un programa ejecutable. (Joyanes Aguilar, 2006).

#### GCC y G++

GCC por sus siglas en ingles quiere decir colección de compiladores de GNU (GNU Compiler Collection), es una distribución integrada de compiladores para varios lenguajes de programación importantes. Estos incluyen en este momento C, C++, Java, Fortran, Ada, Objective-C y Objective-C++. El compilador GNU para C++ es G++, el cual genera directamente código de máquina (“G++ and GCC - Using the GNU Compiler Collection (GCC)”, consultado el 12/6/2017).

## Entorno Integrado de Desarrollo

Un entorno de desarrollo integrado (IDE por sus siglas en inglés) está diseñado para facilitar el trabajo de desarrollo. Los IDEs contienen características como:

* Una consola para ingresar comandos.
* Editor de código fuente.
* Explorador de objetos.
* Interfaces para compilar y correr el software.
* Manejo de proyectos.
* Asistencia de depuramiento.
* Editores de objetos.
* Integración con la documentación (Verzani, 2011).

### QtCreator

QtCreator es un entorno de desarrollo integrado de plataforma cruzada para el desarrollo de aplicaciones de escritorio, empotradas o móviles. (“http://doc.qt.io/qtcreator/”, consultado el 12/6/2017). Se puede ver el espacio de trabajo de QtCreator en la Ilustración 9.

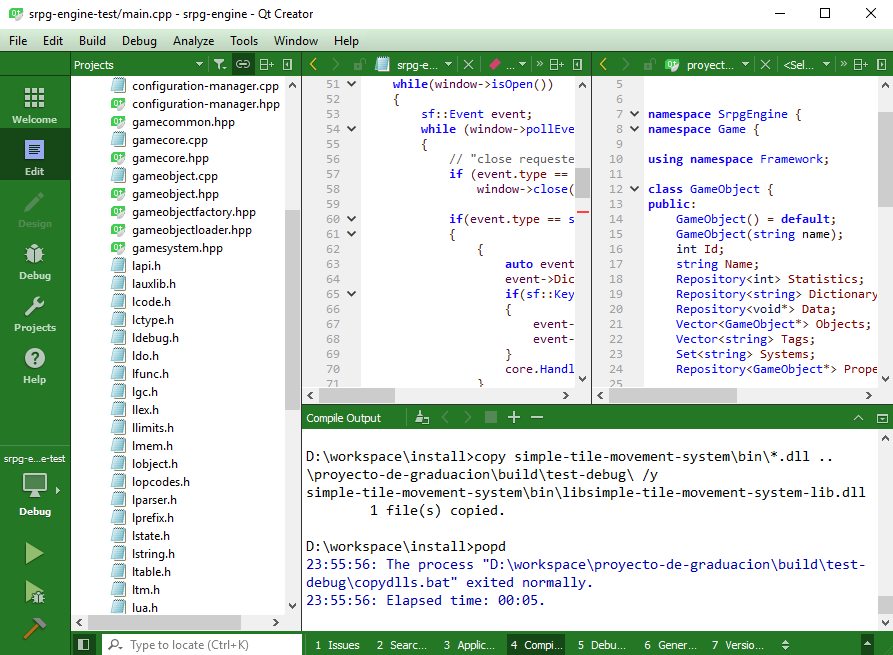


Ilustración 39. Ejemplo de un espacio de trabajo en QtCreator

Fuente: (Qt Corporation, 2017)

# Desarrollo del proyecto

## General

El presente proyecto busca crear un programa de software que se pueda utilizar para desarrollar juegos de rol de estrategia (SRPG). A este tipo de programa se le llama motor o engine. El núcleo del software maneja diferentes sistemas los cuales manejan lógica, sonidos, entradas y salidas. Los objetos que deben manejar estos sistemas están definidos en un archivo el cual se carga al iniciar, este comportamiento puede cambiar según los sistemas que se carguen en el nucleo.

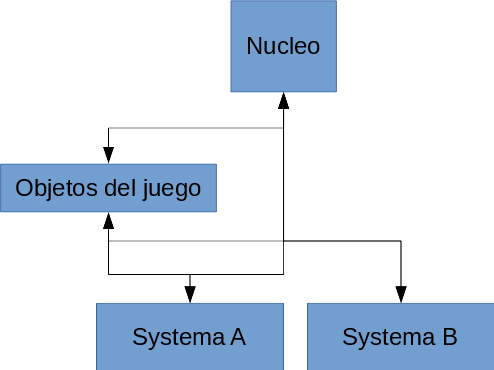


Ilustración 40. Relación sistema, núcleo, objetos del juego.

El núcleo se encarga de manejar la vida de los objetos y la comunicación entre sistemas. Cada parte del sistema se comunica a través de llamadas a eventos, las cuales maneja el núcleo. Cada sistema puede ser tan grande como se desee, pero el propósito principal de hacerlo de esta manera es que exista segregación de responsabilidades, y que los sistemas sean independientes los unos de los otros. El Núcleo también cuenta con representación para entradas comunes de controladores.

Cada sistema puede controlar independientemente su estado, y tiene acceso al estado del núcleo. El sistema puede crear un evento que se propague a través del núcleo o a través de si mismo. De esta manera cambios que se perciban locales no se propagan por todos los sistemas. Esto también desacopla los sistemas haciéndolos independientes de otros sistemas.

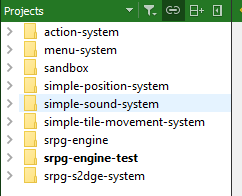


Ilustración 41. Algunos de los sistemas planeados para el proyecto

Cabe destacar que los sistemas desarrollados como ejemplo utilizan el lenguaje de scripting Lua para lograr separar los comportamientos atribuibles al juego en desarrollo y los comportamientos del motor.

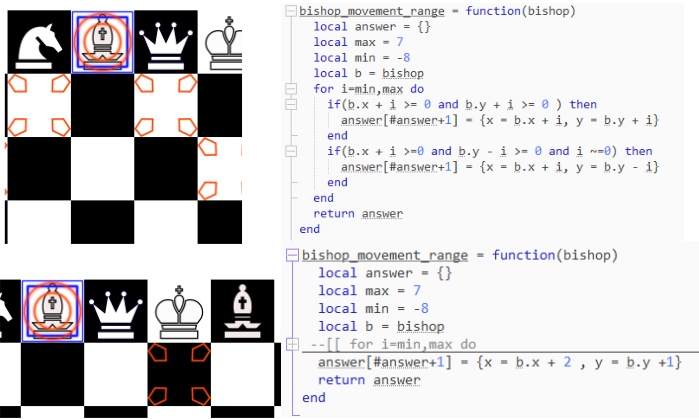


Ilustración 42. Ejemplo de cambios de comportamiento en el juego provistos por Lua

## El ambiente de desarrollo

El proyecto fue desarrollado en Windows y solamente se utilizan librerías, y herramientas que son multi-plataforma. El código fuente puede apreciarse en el enlace <https://github.com/Immac/srpg-engine>. Este repositorio incluye el código fuente y recursos utilizados para poder hacer el proyecto.

Para compilar el proyecto se utilizó el compilador MingW 4.3. Es importante que el compilador que se utilice para compilar el proyecto soporte al menos C++14.

El Ambiente Integrado de Desarrollo (IDE) que se utilizó para este proyecto fue QtCreator, sin embargo, no se utilizó la distribución personalizada de MingW que se encuentra normalmente con el IDE, tampoco se utilizó QMake, que es la herramienta de construcción estándar de QtCreator, se prefirieron herramientas que no estuvieran ligadas directamente a un editor en específico para proveer la libertad de utilizar el editor que se prefiriera por futuros desarrolladores de este proyecto. Para editar y probar los scripts de Lua se utilizó ZeroBrane Studio.

Una vez compilado el programa se deben colocar los recursos a utilizar en el directorio donde se encuentra el ejecutable.

## Unitec Tactics Engine

Unitec Tactics Engine es un motor para juegos de estrategia de rol cuyo objetivo principal es la extensibilidad del mismo. La idea detrás de este motor es que pueda ser extendido para cumplir con los fines que el desarrollador requiera. En base a esto, los módulos, o sistemas como se le llaman internamente, no se comunican directamente unos con otros, lo que hacen es modificar objetos que tienen datos de interés para los sistemas, el intercambio de estos datos hace que los sistemas puedan funcionar juntos sin saber que el otro existe, una segunda capa de comunicación que poseen los sistemas es mediante a eventos que se manejan desde el núcleo del motor.

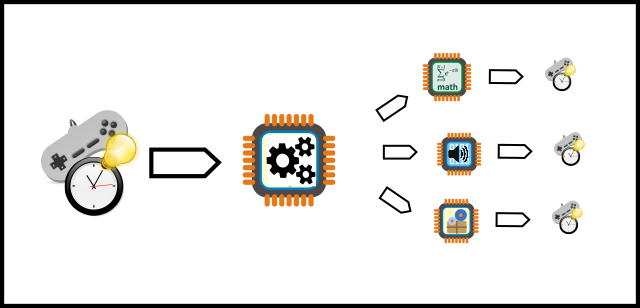


Ilustración 43. Los eventos se alimentan al núcleo el cual los distribuye, y puede crear nuevos eventos.

La unidad básica de movimiento de datos en el proyecto es el GameObject, a este le es posible contener toda la información posible, debido a que el intercambio de datos en los GameObject es la forma más común de comunicación de los sistemas se debe tener cuidado al utilizar los datos guardados en los GameObject.



Ilustración 44. La clase GameObject

Unitec Tactics Engine carga los objetos desde un script de lua, esto permite que la creación de los objetos sea dinámica y programable. Luego se deben cargar los sistemas en el núcleo, y se procede con el ciclo principal de juego, donde cada cierta cantidad de tiempo el núcleo actualizara la lógica del programa para que los sistemas lleven a cabo sus funciones.

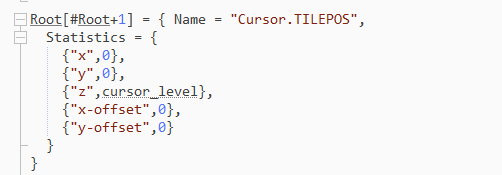


Ilustración 45. Definición del GameObject del sistema de TILEPOS para Cursor.

El núcleo carga todos los objetos en la raíz (Root) del archivo de game-objects.lua. Luego estos se utilizan para armar los objetos en C++ GameObject. Estos objetos incluyen eventos, datos sobre acciones, personajes e incluso la interfaz gráfica del juego.

Junto con el núcleo se desarrollaron 3 módulos, el de graficos simples denominado simple 2d graphics system (s2dge), el de sonido, simple sound system (SSS), el de cuadricula denominado tile positioning system (tilepos), el ultimo cual también contiene los datos para el sistema de acciones.

## Sol y Lua

Uno de los más grandes retos al inicio del proyecto fue el API de Lua, debido a que es un API de C basado en stacks. Para darle claridad a Lua, la librería Sol, encapsula el extraño API de Lua en una forma más familiar para los desarrolladores cuya experiencia es con orientación a objetos.

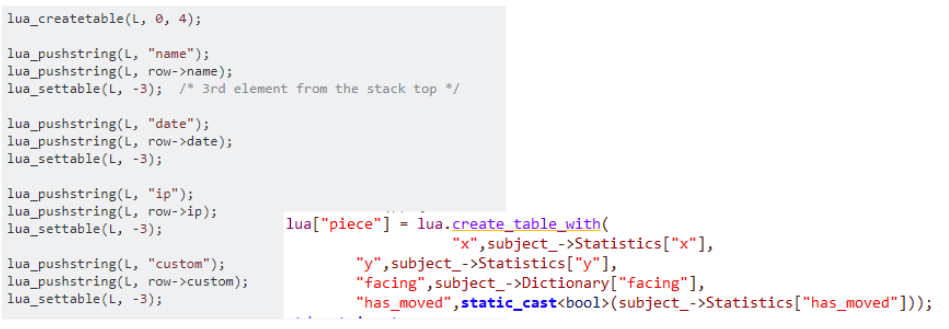


Ilustración 46.Crear una tabla con el Api de Lua (izquierda) y con Sol(derecha)

## Limitantes

Debido a problemas con el tiempo disponible para el autor del proyecto debido a circunstancias fuera del control del autor, no se pudieron cumplir todas las metas que se plantearon de la manera que se hubiese deseado. Esto incluye más ejemplos de utilización de la librería y esquinas que se tuvieron que cortar para poder cumplir con la fecha de entrega. En específico la segregación del sistema de acciones no se logró, aun si se implementó en otro sistema lo ideal fuese que cada parte del sistema se pueda separar para utilizar diferentes partes dependiendo del proyecto que se desee crear. Además de que se subestimo la cantidad de trabajo que toma este tipo de proyecto.

# Conclusiones

Con este proyecto se comprende que la cantidad de planeación y esfuerzo necesarios para crear un motor de juegos es bastante considerable, la cantidad de casos a considerar es substancial y se requiere bastante atención al detalle para poder verificar que funcione como se debe.

Se creo un motor para el desarrollo de juegos de estrategia de rol, si bien todos los componentes no se terminaron, es posible hacer un juego, aunque simple.

Se pudo establecer una metodología de trabajo y un flujo de cómo se debe trabajar en él, aunque debido al tiempo corto no se pudo realizar en su totalidad el flujo deseado.

# Recomendaciones

Para poder extender este proyecto se recomienda adquirir un conocimiento considerable de C++ moderno, en especial deducción automática de clases y de expresiones lambda. También se recomienda seguir patrones de diseño establecidos para resolver complejidad innecesaria que existe en el proyecto. El encapsulamiento es uno de los patrones en específico que se recomienda que se utilice con vehemencia.

Si se utiliza para desarrollar juegos, téngase en cuenta que, para el momento de esta publicación, si bien es posible crear juegos, debido a que no se desarrollaron todos los módulos necesarios, la operabilidad es limitada.

Bibliografía

auto specifier (since C++11) - cppreference.com. (2017, junio 21). Recuperado el 25 de junio de 2017, a partir de http://en.cppreference.com/w/cpp/language/auto

Barton, M. (2008). *Dungeons and Desktops: The History of Computer Role-Playing Games* (1a. ed.). A K Peters, Ltd.

Cambridge Computer Laboratory. (s/f). Computer Laboratory – The Relics Project: Archive photos. Recuperado el 13 de junio de 2017, a partir de http://www.cl.cam.ac.uk/relics/archive\_photos.html#Copyright\_Licencing

CMake. (s/f). Recuperado el 12 de junio de 2017, a partir de https://cmake.org/

Crawford, C. (1984). *The art of computer game design*. Osborne/McGraw-Hill.

Dale, N., & Weems, C. (2007). *Programación y resolución de problemas con C++* (Cuarta Edición). McGraw-Hill Interamericana.

Declaring functions - cppreference.com. (2017, junio 20). Recuperado el 25 de junio de 2017, a partir de http://en.cppreference.com/w/cpp/language/function%23Deleted\_functions

Definition of GAME. (s/f). Recuperado el 10 de junio de 2017, a partir de https://www.merriam-webster.com/dictionary/game

Derpstorm, T. P. (2017). *sol2: Sol v2.0 - a C++ <-> Lua API wrapper with advanced features and top notch performance - is here, and it’s great! Documentation:* C++. Recuperado a partir de https://github.com/ThePhD/sol2 (Original work published el 7 de enero de 2016)

Doar. (2005). *Practical Development Environments*. O’Reilly.

Donarreiskoffer. (2004). *English: Go board, at a Go-weekend, Hoge Rielen, Belgium*. Recuperado a partir de https://commons.wikimedia.org/wiki/File:Go\_board.jpg

Donovan, T. (2010). *Replay: The History of Video Games* (1a. ed.). Yellow Ant.

Emms, J. (2003). *Concise Chess, the compact guide for beginners.* (1a. ed.). Gloucester Publishers plc.

enumeration declaration - cppreference.com. (2017, marzo 31). Recuperado el 25 de junio de 2017, a partir de http://en.cppreference.com/w/cpp/language/enum

G++ and GCC - Using the GNU Compiler Collection (GCC). (s/f). Recuperado el 12 de junio de 2017, a partir de https://gcc.gnu.org/onlinedocs/gcc-4.0.2/gcc/G\_002b\_002b-and-GCC.html

Goldberg, H. (2011). *All your base are belong to us: how 50 years of videogames conquered pop culture* (1a. ed.). Three Rivers Press.

Hello world/Newbie - Rosetta Code. (s/f). Recuperado el 23 de junio de 2017, a partir de http://rosettacode.org/wiki/Hello\_world/Newbie#C

Hollinger, E., & Ratkos, J. (1998). *Final Fantasy Tactics: Prima’s official strategy guide*. Rocklin, California: Prima Publishing.

Ierusalimschy, R. (2016). *Programming in Lua* (4a.). Feisty Duck Digital.

Jean-André Santoni. (2011). *Tethical: Move algo improvements + battle coordinates*. Recuperado a partir de https://www.youtube.com/watch?v=7M66QDrFb4w

Jones, M. (2008, agosto 20). Anatomy of Linux dynamic libraries. Recuperado el 27 de junio de 2017, a partir de http://www.ibm.com/developerworks/library/l-dynamic-libraries/index.html

Joyanes Aguilar, L. (2006). *Programación en C++: algoritmos, estructuras de datos y objetos* (Segunda Edición). España: McGraw-Hill España.

juego. (s/f). Recuperado el 10 de junio de 2017, a partir de http://dle.rae.es/?id=MaS6XPk

Lecode Works. (2017). *LeTBS 0.5 and 0.6 Showcase*. Recuperado a partir de https://www.youtube.com/watch?v=8ZQ8r5tHgXs

Light, A. (2004). *English: Chess pieces – left to right: king, rook, queen, pawn, knight and bishop.* Recuperado a partir de https://commons.wikimedia.org/wiki/File:ChessSet.jpg

Maxwell, B. A. (s/f). A Simple Makefile Tutorial. Recuperado el 25 de junio de 2017, a partir de http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

McConnell, S. (2004). *Code Complete: A practical book of software construction* (2a. ed.). Redmond, Washington: Microsoft Press.

Meyers, S. (2015). *Effective Modern C++: 42 specific ways to improve your use of C++11 and C++14* (1a. ed.). 1005 Gravenstein Highway North, Sebastopol, CA 95472: O’Reilly Media Inc.

Microsoft. (2017a). Aliases and typedefs (C++). Recuperado el 25 de junio de 2017, a partir de https://msdn.microsoft.com/en-us/library/dn467695.aspx

Microsoft. (2017b). How to: Create and Use shared\_ptr Instances. Recuperado el 25 de junio de 2017, a partir de https://msdn.microsoft.com/en-us/library/hh279669.aspx

Microsoft. (2017c). How to: Create and Use unique\_ptr Instances. Recuperado el 25 de junio de 2017, a partir de https://msdn.microsoft.com/en-us/library/hh279676.aspx

Microsoft. (2017d). Lambda Expressions in C++. Recuperado el 25 de junio de 2017, a partir de https://msdn.microsoft.com/en-us/library/dd293608.aspx

Microsoft. (2017e). Smart Pointers (Modern C++). Recuperado el 25 de junio de 2017, a partir de https://msdn.microsoft.com/en-us/library/hh279674.aspx

Moroboshi. (2005). *The picture show the map of a Dungeons and Dragons game, I (“Rocco Pier Luigi”, user:Moroboshi took this pictures the 13 July 2005*. Recuperado a partir de https://commons.wikimedia.org/wiki/File:Dungeons\_and\_Dragons\_game.jpg

Nguyen, A. T. (2014). Tutorial 1: Let’s start with CMake | Learning CMake: A beginner’s guide. Recuperado el 25 de junio de 2017, a partir de https://tuannguyen68.gitbooks.io/learning-cmake-a-beginner-s-guide/content/chap1/chap1.html

Nintendo. (1990). *Final Fantasy Explorer’s Handbook*. Nintendo Of America Inc.

Nipon Ichi Software. (2006). *Disgaea 2: Dark Hero Days*.

override specifier (since C++11) - cppreference.com. (2017, septiembre 6). Recuperado el 25 de junio de 2017, a partir de http://en.cppreference.com/w/cpp/language/override

Qt Corporation. (2017). *QtCreator Screen Capture*.

Qt Creator Manual. (s/f). Recuperado el 12 de junio de 2017, a partir de http://doc.qt.io/qtcreator/

Rollings, A., & Adams, E. (2003). *Andrew Rollings and Ernest Adams on Game Design*. New Riders Publishing.

Ross, I., Harbick, K., De Angelus, C., Meyers, N., & Warriner, C. (s/f). *Disgaea 2: Dark Hero Days. Official Strategy Guide*. Double Jump.

RPG Maker MV | RPG Maker | Make Your Own Video Games! (s/f). Recuperado el 27 de junio de 2017, a partir de http://www.rpgmakerweb.com/products/programs/rpg-maker-mv

RPGBoss Engine. (s/f). Recuperado el 27 de junio de 2017, a partir de http://rpgboss.com/#features

Simsek, B. (2004). Libraries, 4.

SMILE GAME BUILDER - a RPG creation software. (s/f). Recuperado el 27 de junio de 2017, a partir de http://smilegamebuilder.com/en/

std::weak\_ptr - cppreference.com. (2017, diciembre 4). Recuperado el 25 de junio de 2017, a partir de http://en.cppreference.com/w/cpp/memory/weak\_ptr

Stroustrup, B. (1997). *The C++ Programming Language* (3a. ed.). New Jersey: Addison-Wesley Professional.

Stroustrup, B. (2012). Software Development for Infrastructure. *Computer*, *45*(1), 47–58. https://doi.org/10.1109/MC.2011.353

Tresca, M. J. (2011). *Of Fantasy Role-Playing Games* (1a. ed.). McFarland & Company, Inc.

Verzani, J. (2011). *Getting Started with RStudio* (1a. ed.). United States of America: O’Reilly Media Inc.

video game - definition of video game in English | Oxford Dictionaries. (s/f). Recuperado el 10 de junio de 2017, a partir de https://en.oxforddictionaries.com/definition/video\_game

videojuego. (s/f). Recuperado el 10 de junio de 2017, a partir de http://dle.rae.es/?id=bmnbNU7

Wargroove - Announcing Wargroove! (2017, febrero 28). Recuperado el 16 de junio de 2017, a partir de http://wargroove.com/announcing-wargroove/

Wargroove - FAQ. (s/f). Recuperado el 16 de junio de 2017, a partir de http://wargroove.com/faq/

White, D. (2003). *The Battle of Westnoth*.