Análisis y comparativa de un sistema de Gameplay Fundations

Jordi Borraz
Nil Muns
Ricard Ruiz
Héctor Vergel

Índex

Introducción

LÖVE fue lanzado el 13 de enero de 2008 por la misma compañía llamada *LOVE*. Este es un motor gratuito con código abierto, es multiplataforma y fue publicado bajo la licencia conocida como *zlib*, una licencia para hacer videojuegos 2D. El framework está escrito en C++, pero utiliza Lua como lenguaje de *script*.
Algunos juegos creados con LÖVE son: *BLUE REVOLVER*, *Move Or Die* o *Warlock's Tower*. Además, este motor está disponible para estas plataformas: Microsoft Windows, Linux, macOS, iOS, y Android.

Requisitos del Sistema

Tomando como punto inicial el lenguaje de programación de Lua (el utilizado por LÖVE) es un lenguaje que tiene unos requisitos muy bajos para poder ejecutarse, como volverá a mencionarse más en detalle posteriormente en este documento. Empleando un lenguaje vulgar, podría decirse que "lo tira hasta mi lavadora", debido a que los requisitos mínimos para poder utilizar Lua son los siguientes:

- Sistema operativo: cualquier sistema que soporte la API de ANSI C.
- Procesador: cualquiera compatible con x86 y x86-64.
- RAM: 64 Mb.
- Espacio en disco: 1 Mb.

Aunque todos estos requisitos son muy bajos, cuando hablamos de LÖVE tenemos que tener en cuenta que las necesidades aumentan un poco, sin embargo no es necesario tener un gran poder computacional para ejecutarlo.
Los requisitos mínimos de LÖVE son los siguientes:

- Sistema operativo: Windows, MacOS, Linux, Android, iOS.
- Procesador: cualquiera compatible con x86 y x86-64.
- RAM: 512 Mb.
- Espacio en disco: 20 Mb.
- Tarjeta gráfica: cualquier tarjeta compatible con OpenGL 2.1 o posterior.
- Versión de Lua: 5.1, 5.2 o 5.3.
- Biblioteca de tiempo de ejecución de Microsoft Visual C++ 2015.

Implementación del Sistema

Ahora analizaremos la implementación del Sistema de *scripting* (Lua) según la documentación.

Lua es un lenguaje de *script* potente y ligero y está implementado como una biblioteca escrita en C limpio (ANSI C y C++). Una característica a mencionar es que Lua es un lenguaje dinámicamente tipado, lo cual será explicado posteriormente en el apartado de Comparativa. Esto hace que escribir código en Lua sea relativamente más sencillo, también debido a que Lua no necesita que los programas tengan una función o método de cuerpo para que el programa funcione.

Hoy es día 5 en **C++**:

```
#include <iostream>
using namespace std;

int main() {

 int diahoy=5;

 cout << "Hoy es día " << diahoy << endl;

 return 0;
}
```

Hoy es día 5 en **Java**:

```
public class main {

 public static void main(String[] args) {

 int diahoy=5;

 System.out.println("Hoy es día "+diahoy);
 }

}
```

Hoy es día 5 en **Lua**:

```
diahoy=5

print("Hoy es día " .. diahoy)
```

Otro factor que hace que crear código en Lua sea más sencillo es que no nos tendremos que preocupar por la gestión de memoria, ya que la realiza automáticamente. Esto quiere decir que no debemos estar pendientes ni de asignar ni de reservar memoria para nuevos objetos ni de liberarla cuando los objetos ya no sean necesarios. Esto lo hace ejecutando un *garbage collector* de vez en cuando para eliminar objetos muertos.

El entorno de ejecución de Lua es autónomo, lo que significa que no depende de ningún sistema operativo específico y puede ser utilizado en muchas plataformas (multiplataforma).

Se procede ahora a desarrollar la API de Lua. Esta implementa una serie de módulos y funciones que permiten a los programadores interactuar con el intérprete de Lua para realizar tareas específicas. La API está escrita en C, esto facilita la integración de Lua con otros lenguajes de programación y sistemas, permitiendo también una mayor eficiencia.

Podríamos colocar a Lua en los lenguajes de *scripting* en "runtime", ya que permite personalizar funcionalidades y sistemas del motor, en este caso LÖVE. Otra característica de Lua es que es un lenguaje interpretado, por lo tanto, podemos considerar que es multiplataforma.

Para comenzar a implementar código en LÖVE solo necesitaremos crear un par de cosas. Lo primero, una carpeta en cualquier lugar, seguidamente siempre tendremos que crear un archivo "main.lua" en el cual gestionaremos todo el renderizado y *update* de nuestro juego.

Es ahora cuando entraremos en detalle en el estructuramiento de los *scripts* en LÖVE, empezando con las funciones de *callback* más comunes de LÖVE, que son las siguientes:

- love.load: esta función se llama al cargar un juego y se utiliza para inicializar los recursos y establecer la configuración inicial.
- love.update: esta función se llama en cada *frame* de juego y se utiliza para actualizar el estado del juego.
- love.draw: esta función se llama en cada *frame* de juego y se utiliza para dibujar los elementos en la pantalla.
- love.keypressed: esta función se llama cuando se presiona una tecla y se utiliza para detectar y controlar la entrada del usuario.
- love.mousepressed: esta función se llama cuando se presiona un botón del *mouse* y se utiliza para detectar y controlar la entrada del usuario.
- love.resize: esta función se llama cuando se cambia el tamaño de la ventana de juego y se utiliza para redimensionar los elementos en la pantalla.

- love.quit: esta función se llama cuando el usuario cierra el juego y se utiliza para liberar los recursos y salvar la configuración.

Procedemos ahora a mencionar algunos módulos que facilitan funcionalidades y mejoran la productividad del desarrollo:

- love.graphics: este módulo es una de las partes más importantes de LÖVE y permite a los desarrolladores crear y controlar gráficos en el juego, incluyendo la gestión de imágenes, texto y formas.
- love.audio: este módulo permite a los desarrolladores reproducir y controlar la música y los efectos de sonido en el juego.
- love.physics: este módulo permite a los desarrolladores crear y controlar la física en el juego, incluyendo la gestión de cuerpos, colisiones y la dinámica de los objetos.
- love.timer: este módulo permite a los desarrolladores crear y controlar temporizadores en el juego para realizar tareas específicas después de un período de tiempo.
- love.keyboard: este módulo permite a los desarrolladores controlar la entrada del teclado en el juego.

Ejemplos de Uso

Uno de los métodos más importantes dentro de la programación en videojuegos es el propio hecho de mover el *player* por el mapa. Para hacer esto, LÖVE utiliza el input del jugador, el cual recoge con el método -> love.keyboard.isDown(). Este método básicamente comprueba si hay cierta tecla presionada desde el input.

```
function Player:move(dt)
    if love.keyboard.isDown("d", "right") then
        if self.xVel < self.maxSpeed then
            if self.xVel + self.acceleration * dt < self.maxSpeed then
                self.xVel = self.xVel + self.acceleration * dt
            else
                self.xVel = self.maxSpeed
            end
        end
    elseif love.keyboard.isDown("a", "left") then
        if self.xVel > -self.maxSpeed then
            if self.xVel - self.acceleration * dt > -self.maxSpeed then
                self.xVel = self.xVel - self.acceleration * dt
            else
                self.xVel = -self.maxSpeed
            end
        end
    else
        self:applyFriction(dt)
    end
end
```

Aparte del movimiento terrestre, el salto es muy común dentro de los videojuegos. Para recrear esta acción, LÖVE lo hace como muchos otros lenguajes de programación dentro de los videojuegos. El código guarda unas variables para saber si el jugador ha tocado el suelo y si es así le permite saltar de nuevo, para hacer que el personaje vaya para arriba en el momento adecuado le cambia la velocidad de movimiento del eje correspondiente para que el jugador vaya para arriba.

```
function Player:jump(key)
    if (key == "w" or key == "up") then
        if self.grounded or self.graceTime > 0 then
            self.yVel = self.jumpAmount
            self.graceTime = 0
        elseif self.hasDoubleJump then
            self.hasDoubleJump = false
            self.yVel = self.jumpAmount * 0.8
        end
    end
```

Una de las particularidades de LÖVE es que tiene un script llamado Main que es donde van a parar todas las referencias de los otros scripts. En el script Main hay un método llamado love.load, el cual se encarga de cargar todos estos scripts y métodos, por ejemplo:

- El Player:load -> se encarga de cargar el script con nombre Player.
- El love.graphics.newImage() -> se encarga de cargar una imagen de tu repositorio dentro del propio juego.
- El love.physics.newWorld(0,0) -> es una función que sirve para crear un nuevo mundo con un tamaño concreto.

```
function love.load()
    Map = STI("map/1.lua", {"box2d"})
    World = love.physics.newWorld(0,0)
    World:setCallbacks(beginContact, endContact)
    Map:box2d_init(World)
    Map.layers.solid.visible = false
    background = love.graphics.newImage("assets/background.png")
    Player:load()
end
```

Aparte del script Main, LÖVE también necesita una clase llamada Conf, la cual es indispensable, ya que sin esta los aspectos gráficos de la pantalla no se pueden especificar. Se crea una variable específica (llamada "t" en este caso), la cual tiene algunas propiedades como: el nombre del juego, la versión, el tamaño y ancho de la ventana del juego y si tiene vsync o no.

```
function love.conf(t)
    t.title = "Platformer"
    t.version = "11.3"
    t.console = true
    t.window.width = 1280
    t.window.height = 720
    t.window.vsync = 0
end
```

Para animar en LÖVE se usa una *sprite sheet,* la cual tiene las divisiones correspondientes para cada *frame* de la animación, ya que en sí no tiene un reproductor de animaciones tan avanzado como otros motores Unity o Unreal. Una vez teniendo esto, estos frames se ponen en una lista que va avanzando en relación a un *timer*.

```
function Enemy:animate(dt)
    self.animation.timer = self.animation.timer + dt
    if self.animation.timer > self.animation.rate then
        self.animation.timer = 0
        self:setNewFrame()
    end
end

function Enemy:setNewFrame()
    local anim = self.animation[self.state]
    if anim.current < anim.total then
        anim.current = anim.current + 1
    else
        anim.current = 1
    end
    self.animation.draw = anim.img[anim.current]
end
```

Comparativa LÖVE vs Unity

En este apartado se procede a comparar el sistema de *scripting* de LÖVE con el de Unity.
En primer lugar, comparando los lenguajes de *scripting* de ambos motores podemos observar que el motor LÖVE utiliza el lenguaje de *scripting* conocido como Lua. En cambio, el motor de Unity emplea el lenguaje de *scripting* conocido como C#. Hay diferencias entre estos dos lenguajes de *scripting*. Una de ellas es que Lua es un lenguaje interpretado, mientras que C#

es ambos, se compila en un lenguaje ensamblador intermedio y luego es interpretado por el entorno local.
Los lenguajes interpretados (Lua) son aquellos lenguajes de programación donde el código fuente se ejecuta directamente, instrucción a instrucción. En cambio, los lenguajes compilados (C#) son aquellos que requieren de un compilador para funcionar.

Vamos a ver las ventajas y desventajas de ambos lenguajes, empezando por los lenguajes compilados. La primera ventaja de estos lenguajes es su eficiencia en la corrección de errores, es decir, cuando se hace el proceso de compilación, el compilador emite una lista de todos los errores que necesitan corrección. Otra ventaja que vemos en los lenguajes compilados es que su ejecución es mucho más rápida que en el caso de los lenguajes interpretados, ya que se ejecuta cuando se ha compilado todo el bloque de código y no instrucción a instrucción. Como desventajas podemos mencionar que los errores no se pueden corregir hasta que se complete la compilación y que, una vez que se han revisado y corregido los errores, es necesario volver a hacer el proceso de compilación desde cero.
Procedemos ahora a analizar los lenguajes de programación interpretados. Una de las ventajas que tienen estos lenguajes es que son multiplataforma, es decir, que el intérprete suele estar en varios sistemas operativos, así que no se tiene que adaptar el código a una plataforma en concreto. Otra ventaja es su portabilidad: el mismo programa puede ser llevado a diferentes plataformas. Las principales desventajas de los lenguajes interpretados son las siguientes: en primer lugar y la más importante es la velocidad de ejecución del programa, porque cuando compilamos lo transformamos a código máquina, que es en el que funciona el procesador. Otra desventaja es el hecho de ser multiplataforma, ya que es necesario que en la máquina dónde va a funcionar tenga el intérprete, ya sea como *framework* o como máquina virtual.
Finalmente, podemos destacar que la desventaja de los lenguajes compilados es la velocidad de ejecución, que a su vez es la ventaja de los lenguajes interpretados.
Procederemos ahora a ver las diferencias entre los tipados de ambos lenguajes. Como ha sido mencionado anteriormente, Lua es un lenguaje de tipado dinámico, en cambio C# es un lenguaje de tipado fuerte. Que un lenguaje sea de tipado dinámico significa que las variables no tienen tipos, solo tienen tipo los valores, no existiendo definiciones de tipo en el lenguaje. Todos los valores almacenan su propio tipo. En cambio, los lenguajes de programación de tipado fuerte son aquellos utilizados para que el código incluya el tipo de dato al declarar la variable. Solemos decir que esta forma de programar es más expresiva al saber qué sucederá

exactamente con las órdenes programadas. La ejecución de estos lenguajes es más veloz, ya que la inferencia de los tipos es anterior y el lenguaje necesita verificarlos a la hora de ejecutarlos.

Conclusión

Tras la realización del trabajo y el exhaustivo análisis de estos dos motores con todas sus características hemos llegado a las siguientes conclusiones:

El sistema de *scripting* de LÖVE es más fácil de utilizar ya que Lua es un lenguaje de *scripting* potente y sencillo gracias a sus características, como su tipado dinámico y que es un lenguaje de extensión. Es muy útil para introducirse y aprender y desarrollar proyectos cortos y flexibles.

En cambio Unity y su sistema de *scripting* y API y su estructura en general permite hacer proyectos con más alcance. Con más variedad de módulos se pueden hacer cosas que en LÖVE tardarían más. Y no solo es la variedad de módulos, sino que también la mayoría de ellos otorgan más funcionalidades que los que otorgan los de LÖVE. En definitiva, ambas opciones son válidas y útiles, pero la que debas escoger dependerá del enfoque que tenga el proyecto en cuestión que se realice.

Bibliografía

Jeepzor. (n.d.). Jeepzor/platformer-tutorial: Source code for each episode on my tutorial series on how to make a platformer game in Löve (love2d). Recuperado de https://github.com/Jeepzor/Platformer-tutorial

LÖVE. (n.d.). Recuperado de https://love2d.org/

Medina, E. (2016, 23 de mayo). Lua, un buen lenguaje para empezar a programar. Recuperado de https://www.muylinux.com/2016/05/23/lua-lenguaje-empezar-programar/