ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN INTERACTIVA LÚDICA MEDIANTE GINGA NCLUA PARA TELEVISIÓN DIGITAL TERRESTRE (ISDB-TB)

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

SERGIO DANIEL CHAMORRO GARCÍA
serginho9@live.com

DIRECTOR: ING. FLORES CIFUENTES WILLIAMS FERNANDO
fernando.flores@epn.edu.ec

Quito, Marzo 2016

DECLARACIÓN

Yo, Sergio Daniel Chamorro García, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Sergio Daniel Chamorro García

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Sergio Daniel Chamorro García, bajo mi supervisión.

Ing. Flores Cifuentes Williams Fernando
DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

Agradezco a todos quienes han caminado alado mío y me han apoyado a realizar mis sueños, un profundo agradecimiento a mis compañeros de estudio y maestros con quienes he compartido este viaje de conocimientos.

Sergio Daniel Chamorro García

DEDICATORIA

Dedico este trabajo a mi familia y a mi novia quienes siempre me han brindado su amor y me han apoyado incondicionalmente para seguir adelante, dedico también este trabajo a todos mis mentores quienes me han enseñado con paciencia y me han inculcado el amor por la ciencia.

Sergio Daniel Chamorro García

CONTENIDO

ÍNDICE DE FIGURAS

RESUMEN

En el presente proyecto se desarrollará una aplicación interactiva lúdica, mediante el middleware Ginga del estándar ISDBT-b, para esto se utilizará principalmente el lenguaje de programación Lua y una metodología de desarrollo aplicada a videojuegos, se probarán los resultados obtenidos en el emulador(máquina virtual Ginga) y un receptor Set-top box real.

El proyecto está dividido en cuatro capítulos:

En el capítulo I se estudiarán los módulos que integran el Middleware Ginga, Ginga, y las características en rendimiento, usabilidad y extensibilidad que presenta embeber el lenguaje Lua en el middleware Ginga enfocada principalmente en el área de los videojuegos.

En el capítulo II se realiza el desarrollo de la aplicación interactiva utilizando las recomendaciones dadas por la empresa CPqD de Brasil quienes han hecho investigación en el área de desarrollo de aplicaciones interactivas para el estándar ISDB-Tb y lo referente a ingreso y lectura de información en lo referente al aspecto visual, en este capítulo se presenta el análisis para incluir estilos, colores regiones de la pantalla y metodología de desarrollo de software.

En el capítulo III se realizarán las pruebas del aplicativo desarrollado en base a variaciones de resolución de pantalla, y su rendimiento en un decodificador de capacidad de procesamiento promedio para evaluar la fluidez con que se ejecuta el juego.

En el capítulo IV describe las conclusiones y recomendaciones obtenidas en el desarrollo y pruebas de la aplicación interactiva de televisión digital.

PRESENTACIÓN

Mediante el desarrollo de una aplicación lúdica (videojuego) se evaluarán las prestaciones que tiene el middleware Ginga en relación a la animación, interacción del usuario mediante el control remoto, uso del canal de retorno (comunicación del usuario- servidor mediante una red LAN), y en conjunto las posibilidades que tiene el middleware Ginga NCL Lua para la creación de videojuegos.

En este proyecto se implementará una aplicación interactiva que se probará en un receptor real de ISDB-Tb que integre el middleware Ginga en su versión NCL-Lua.

El presente proyecto es aplicable en nuestro entorno ya que el estándar ISDB-Tb se ha adoptado en el Ecuador y gran parte de Sudamérica, a medida que se vaya implementando la transición de televisión analógica a digital se desarrollarán muchas aplicaciones interactivas que abarcan varios temas tales como el T-Learning, T-Goverment, T-Healt y muchos otros más, entre ellas el desarrollo de pequeños juegos, animaciones y trivias para aplicaciones de carácter comercial.

CAPÍTULO 1

FUNDAMENTOS TEÓRICOS

1.1 BASES DEL MIDDLEWARE

A continuación se presenta una breve reseña sobre el lenguaje Lua que es la base sobre la cual se desarrolló el middleware Ginga para televisión digital terrestre en el estándar japonés con variaciones brasileñas (Estándar de televisión digital que está siendo implementado en el Ecuador).

El leguaje de script Lua fue desarrollado por Roberto Lerusalimschy, Luiz Henrique de Figueiredo y Waldemar Celes en los laboratorios del departamento de Ciencias de la Computación de la PUC-Rio Pontificia Universidad Católica de Río de Janeiro en Brasil, luego este proyecto pasó a manos del Grupo de Tecnología de Gráficos Computarizados, y ahora se lo desarrolla en LabLua.

Lua evolucionó desde un simple lenguaje de configuración hasta ser un lenguaje completo que combina una sintaxis procedural simple con una poderosa construcción de descripción de datos basada en arreglos asociativos y semántica extensible. Lua se escribe dinámicamente, corre en base a interpretar bytecode por medio de una máquina virtual. Tiene un manejo automático de memoria con colectores de basura, haciéndolo ideal para scripting y creación de prototipos de forma rápida.

Lua es un lenguaje de alto nivel de abstracción, esto significa que los programadores pueden escribir código en una forma similar a la que su mente concibe la solución a un determinado problema. El código generado en el lenguaje Lua es ejecutado en por el computador mediante una máquina virtual en este caso el Lua virtual machine (máquina virtual Lua), ya que el código necesita ser traducido a un bajo nivel de instrucciones las cuales son las operaciones más fundamentales que la CPU o la máquina virtual soporta; éste set de instrucciones

que puede ser leído y ejecutado en bajo nivel es lo que se conoce como bytecode.

Por el diseño y simplicidad a la hora de ejecución Lua es uno de los lenguajes de scripting más rápidos en lo que se refiere a rendimiento a tal punto que comúnmente existe un dicho “tan rápido como Lua”, varias pruebas (benchmarking ver capítulo 1.6) lo posicionan como uno de los lenguajes más rápidos de scripting interpretado; Lua ha sido muy utilizado en diversos sistemas tales como Switches Ethernet, procesadores de imágenes, robótica, desarrollo web, aplicaciones móviles y juegos en varias plataformas. Uno de los principales pilares de la rapidez de Lua es la efectividad con la que la máquina virtual Lua puede ejecutar el programa. Han habido muchos otros lenguajes que se escriben en script pero al inicio no han presentado rendimientos tan altos como Lua; un ejemplo de esto es el caso del lenguaje de JavaScript para el cual los diferentes browsers han hecho importantes avances para mejorar el motor que ejecuta JavaScript, tal como lo hizo Google con su versión V8 JavaScript Engine.

En estos últimos años, Lua ha sido uno de los lenguajes de scripting que ha ganado bastante popularidad (Figura 1.7), gran parte de esto se debe a que ha sido usado para motores de juegos, extensiones de juegos en base a scripting e integraciones en varios sistemas en base a un middleware como es el caso del middleware Ginga. Hay muchas plataformas de desarrollo de software que utilizan Lua, un ejemplo de esto es el SDK Corona que es un entorno de desarrollo que permite exportar aplicaciones móviles basadas en Lua.

Una buena práctica de diseño de componentes de hardware es dimensionarlo en base a los requerimientos de software que va a presentar de un determinado sistema, para esto se busca el balance entre el costo de los dispositivos de procesamiento, memoria, calidad de materiales, etc. El software debe ser desarrollado de tal forma que no exceda las capacidades del hardware, en última instancia si los recursos de hardware no son suficientes se debe optimizar el código del programa a tal punto que tenga una buena usabilidad. Toda esta

arquitectura comienza desde un pilar importante que es la selección del lenguaje de programación, esto se lo hace en base a las prestaciones (librerías, compatibilidad con aceleradores gráficos, etc.) que tiene para resolver una determinada tarea, de la misma manera se debe prever qué cantidad de recursos de hardware va a necesitar.

Lua se ha vuelto una alternativa altamente viable para solventar todas estas necesidades ya que combina un alto rendimiento con un consumo de memoria bajo. Su peso liviano al momento de ser embebido es tal que su última versión Lua 5.2.1 ocupa alrededor de 245KB por lo que resulta económico para ser embebido en una amplia gama de sistemas, Lua es una de las opciones más populares de extensibilidad de muchos otros lenguajes de programación tal como es el caso de C, C++, Java, C#, Smalltalk, Fortran, Ada, Erlang, Perl, Ruby entre otros.

Lua uno de los lenguajes que más rápido puede ser implementado con la menor footprint (tamaño de memoria necesario para su ejecución) dada esta ventaja puede ser embebido con otras aplicaciones, Lua tiene una fuerte integración con código escrito en otros lenguajes, Lua se puede extender mediante el uso de librerías de C pero últimamente también se han desarrollado librerías de extensibilidad para otros lenguajes y plataformas tal como es el caso de las aplicaciones móviles y juegos (IOS, Android), de la misma manera es fácil extender otros lenguajes mediante código escrito en Lua.

Lua se distribuye gratuitamente bajo licencia MIT, Lua puede ser usado para cualquier propósito incluyendo propósitos comerciales, no necesita de pago de royalties, permisos escritos ni permiso de copia además de ser de código abierto.

1.2 USOS DE LUA

Con respecto a la portabilidad Lua es distribuido en un pequeño paquete el mismo que puede ser ejecutado en cualquier sistema que tenga un compilador C estándar, por lo que puede ser instalado en Linux, Windows, dispositivos móviles

con diferentes sistemas operativos como Android, IOS, Symbian, etc. además podemos encontrarlo embebido en microprocesadores como ARM y Rabbit; Lua se implementa como Standalone no necesita librerías externas o de otros sistemas para funcionar.

Algunos ejemplos de aplicaciones que incluyen Lua:

- El Plugin 3DMLW utiliza scripts de Lua para animaciones 3D y manipular los efectos elementos gráficos.
- Adobe Photoshop Lightroom utiliza Lua como lenguaje de extensibilidad.
- EL popular servidor HTTP Apache utiliza Lua en el proceso de petición de las páginas web (esta funcionalidad está presente desde la versión 2.3, ya que en su núcleo integra el módulo llamado mod_lua).
- Artweaver que es un editor gráfico utiliza Lua para escribir el código de sus filtros.
- Awesome, que es un administrador de ventanas, fue escrito parcialmente en Lua, también utiliza Lua para manejar y manipular su estructura de archivos.
- El Canon Hack Development Kit (CHDK), que es un firmware de código abierto para las cámaras Canon, utiliza Lua como uno de sus dos lenguajes disponibles para realizar scripts.
- Celestia, un programa de educación en temas de astronomía, utiliza Lua como su lenguaje de script.
- Cisco utiliza Lua para implementar las Políticas de Acceso Dinámico (Dynamic Access Policies) dentro de su aparato de seguridad adaptiva, en dispositivos de interconexión.
- Dolphin Computer Access utiliza Lua para que mediante este lenguaje sus diferentes funciones sean accesibles para sus usuarios.
- Ginga, el middleware para el estándar de Televisión Digital (SBTVD o ISDB-T), utiliza Lua como un lenguaje de script para realizar aplicaciones de forma declarativa, Ginga-NCL.

- GrafX2, un editor de arte basado en píxeles, puede correr scripts Lua ya sea para procesar una simple imagen o para generar una compleja ilustración.
- El editor de dibujos llamado “Ipe” (principalmente utilizado por LaTeX) utiliza Lua para solventar sus funcionalidades además que es su lenguaje de extensibilidad.
- Lego Mindstorms NXT y NXT 2.0 pueden ser controlados mediante Lua utilizando controladores de terceros.
- VLC media player utiliza Lua para proveer el soporte de scripts.
- WeeChat IRC permite correr scripts escritos en Lua.
- Wireshark analizador de paquetes en la red permite escribir filtros de protocolos en Lua.

1.3 COMPILACIÓN DEL LENGUAJE LUA

Las construcciones y comandos de Lua son ejecutados en un ambiente global único, el mismo que guarda todas las variables globales y definiciones de funciones, al ser ejecutado Lua carga automáticamente estas definiciones y persisten mientras el programa esté corriendo.

El ambiente global de Lua puede ser manipulado por código escrito en Lua o por bibliotecas C que utilizan a su vez funciones de interfaz C-Lua. En Lua los códigos son compuestos por comandos globales y definiciones de funciones. Un módulo de Lua es un archivo con una cadena de caracteres; cuando los módulos de Lua se cargan son compilados por un lenguaje de máquina virtual interna de esta manera quedan listos para ser ejecutados. Los códigos globales que no están dentro de funciones son automáticamente ejecutados al final de la compilación, estos códigos de funciones son almacenados y ejecutados al ser llamados por las respectivas funciones, es por esto que se dice que Lua usa una estrategia híbrida de compilación e interpretación que evita un mayor costo en procesamiento y memoria que tendríamos en una compilación de interpretación textual directa, al mismo tiempo que mantiene la versatilidad de un ambiente interpretativo.

1.4 CONSIDERACIONES DEL LENGUAJE LUA Y SU EXTENSIBILIDAD

No existe un única forma de escribir código en Lua, por lo que podemos tener código ocupando una o varias líneas, los comandos pueden ser separados por (;) al estilo C y los comentarios se inician con (--) para concatenar cadenas de caracteres se utiliza (..).

A diferencia de otros lenguajes de programación, que promueven elementos específicos para desarrollar mediante una metodología orientada a objetos, Lua permite programar de la manera que queramos definiendo los comportamientos que nosotros deseemos, podemos hacer uso de funcionalidades como herencia, variables y métodos protegidos, polimorfismo y todo esto puede ser implementado en Lua o C. Lua dispone de tablas que son mapas asociativos los mismos que pueden ser indexados por números o por strings. Esto en la práctica se traduce a una estructura extremadamente útil, poderosa y fácil de utilizar.

Las tablas de Lua pueden contener cualquier tipo de objetos desde simples números hasta referencias a funciones, otro recurso poderoso que tiene Lua son los ‘tag methods’ o su sistema de ‘tags’ que permite que cualquier tabla pueda tener su propio conjunto de métodos, usando estos ‘tag methods’ podemos redefinir lo que sucede cuando se intenta por ejemplo indexar un elemento a la tabla o hacer operaciones con los elementos dentro de esta tabla.

Uno de los principales aspectos de Lua es su extensibilidad, Lua generalmente es ejecutado por medio de un programa anfitrión que suele ser C aunque pueden ser muchos otros, se han definido un conjunto de funciones que permite la conexión y comunicación entre Lua y el lenguaje anfitrión permitiendo a este ejecutar comandos arbitrarios de Lua.

Cualquier tipo de dato de Lua puede ser pasado como parámetro o valor de retorno para una función C y viceversa, lo que incluye números, tablas, funciones Lua y hasta funciones en C. En la práctica usando el tipo de dato ‘userdata’

cualquier dato de C puede ser pasado a Lua inclusive punteros lo que le da un alcance total a Lua dentro de C. Este tipo ‘userdata’ es un aliado del ‘tag methods’ que es un mecanismo muy flexible, se puede construir un objeto en C++ y pasarlo a Lua indicando su clase a través de un ‘tag’ así cuando este sea indexado el valor retornado será determinado por la función en C.

Extensión mediante librerías: se pueden cargar módulos independientes en mediante archivos de extensión (.lua), al final todos estos módulos son cargados en un solo ambiente global por lo tanto es importante el orden en que se cargan todos estos archivos. Bibliotecas enteras pueden ser definidas en C/C++ y exportadas a programas Lua, estas inclusive pueden estar encapsuladas en tablas de tal forma que al momento de exportarlas no interfieran con las variables globales de Lua.

Los principales tipos de datos que se definen en Lua son los siguientes:

1. ‘number’ el mismo que es implementado como flotante.
2. ‘strings’ cadenas de caracteres.
3. ‘tablas asociativas’.
4. ‘userdata’ tipo de dato que permite intercambio de datos entre Lua y el lenguaje anfitrión el mismo que tiene la capacidad de almacenar un puntero de C.
5. ‘function’ funciones en lenguaje Lua.
6. ‘C function’ funciones en lenguaje C.
7. ‘nil’ nulo o falso.

Nota: Los tipos de datos ‘userdata’ y ‘C function’ fueron diseñados para permitir la interacción entre el programa anfitrión C y Lua, Lua puede hacer uso de funciones escritas en C.

Lua fue proyectada para ser un lenguaje interpretado a partir de ‘bytecode’ generado a partir de una pre-compilación. Para mantener la simplicidad se usaron herramientas ya existentes de compilación como “lex” que divide al código fuente en ‘tokens’ y “yacc” que busca la estructura jerárquica.

Por el avance de la tecnología y necesidad de rendimiento gráfico fue necesario que Lua pueda leer meta-archivos gráficos de gran tamaño, para esto Lua necesitó acelerar el proceso de compilación, por lo que se sustituyó ‘lex’ por un analizador léxico dedicado lo que dobló la velocidad del compilador.

Lua es proyectado como un lenguaje de extensión, Lua es muy utilizado a la hora de personalización de programas, es decir es una extensión para controlar y cambiar algunos parámetros dentro del programa principal. Debido a su sintaxis simple y su meta-mecanismo se utilizaba mucho a Lua como un lenguaje de extensión en lugar de un lenguaje principal.

1.5 LUA Y EL MIDDLEWARE GINGA

Internacionalmente Lua es muy popular, se conocen muy bien sus beneficios a tal punto que muchos juegos ya incluyen scripts Lua, empresas como Microsoft, LucasArts, BioWare, Absolute Studios eMonkeystone, Ubisoft y otros utilizan Lua en algunos de sus juegos.

El middleware Ginga del estándar de televisión digital terrestre ISDB-Tb tiene dos opciones a la hora de presentar aplicaciones interactivas, una es por medio de Java con sus aplicaciones Xlet y la otra por Ginga NCL. Una aplicación NCL son documentos en formato XML los mismos que se interpretan y ejecutan mediante Lua. Se pueden desarrollar aplicaciones solo con NCL o aplicaciones híbridas con Lua y NCL llamadas “NCLua”; al momento de combinar NCL y Java se llaman aplicaciones NCLet.

NCL es un lenguaje ‘declarativo’ ya que permite definir condiciones indicando que es lo que se desea presentar, especificando cómo, cuándo y dónde, esto se realiza mediante mecanismos internos de control sin especificar exactamente el código ya que para esto se usan funciones pre definidas. Lua en cambio es un lenguaje ‘imperativo’ es decir se definen paso a paso las instrucciones, algoritmos, procesos necesarios para solucionar un problema o realizar alguna acción. Al dejar libre la opción de combinar estos dos lenguajes NCL y Lua se

combina la simpleza y rapidez de utilizar un lenguaje procedural con el poder y flexibilidad que provee el lenguaje imperativo Lua.

NCL se define como una aplicación XML basada en NCM (Nested Context Model, Modelo de Contexto Anidado) que es un modelo para la especificación de documentos de hipermedia (métodos para diseño o composición de contenidos como texto, imágenes, audio y video) con una sincronización temporal y espacial entre estos objetos de media. Se especifica el comportamiento espacial y temporal de estos objetos, a manera de cómo, cuándo y dónde se los presentan al momento de desencadenarse una acción como es el caso de presionar un botón acceder a un menú, o al terminar un temporizador.

Se pueden incluir scripts NCLua en documentos NCL, ya que NCL necesita de la ayuda de Lua para realizar tareas específicas o más complicadas tales como realizar operaciones matemáticas, uso del canal de interactividad, control más complejo del teclado, control de colisiones de elementos gráficos, animaciones, inteligencia artificial entre otros.

1.6 LUA EMBEBIDO, PRINCIPALES APLICACIONES Y RENDIMIENTO

Lua presenta varias ventajas técnicas por lo que en muchos casos la mejor opción a la hora de elegir un lenguaje de programación para extender o controlar un sistema como se puede observar en la Figura 1.1.

El middleware Ginga provee la compatibilidad entre Ginga JAVA y Ginga NCL mediante el bridge. El Common Core o Núcleo Común del middleware está desarrollado en C y éste es el programa anfitrión para poder embeber Lua que a la vez es el interpretador de las aplicaciones escritas en NCL.

Las partes importantes son: el motor de decodificación de contenido declarativo llamado NCL Fromatter, el módulo XHTML que incluye estilos CSS y un interpretador ECMAScript, y por último la máquina virtual Lua que permite correr los scripts de Lua.

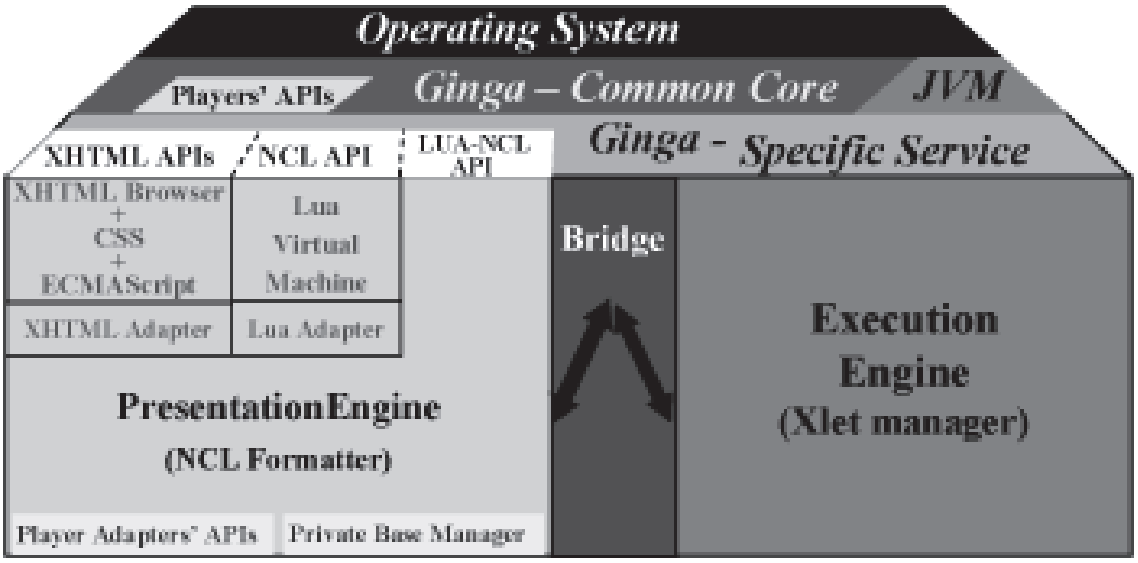


Figura 1.1 Arquitectura del Middleware Ginga

Escribir un motor de lógica o programa principal en Lua ofrece ventajas muy interesantes ya que por un lado los recursos de memoria y procesamiento que se utilizarán serán drásticamente menores que con otros lenguajes, además de que Lua emplea estos recursos de forma eficiente.

Estructura del Ginga Common Core:

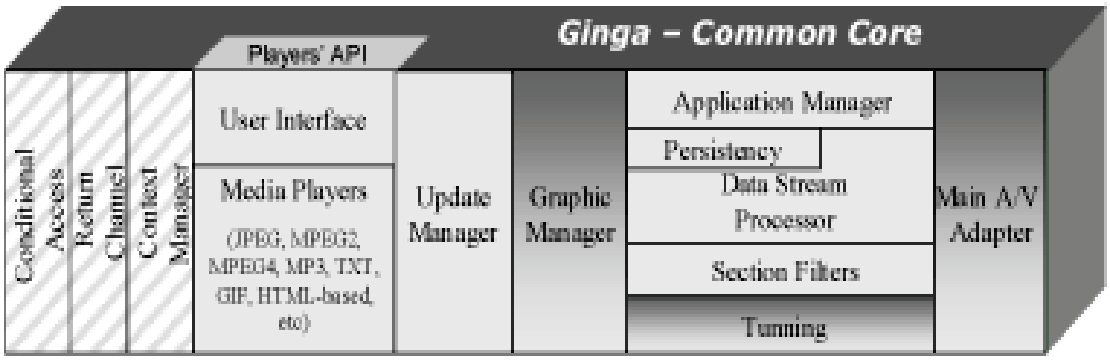


Figura 1.2 Ginga Common Core

Ginga-NCL (incluye Lua) y Ginga-J (Aplicaciones Ginga que utilizan Java) están construidas sobre el módulo Ginga Common Core, el Common Core (Núcleo común) resuelve las necesidades de decodificación y presentación de tipos de contenidos tales como PNG, JPEG, MPEG, etc. a ambos tipos de aplicaciones

'procedurales' y 'declarativas'. El Common Core está compuesto por decodificadores y procedimientos para obtener contenidos.

Lua al ser distribuida mediante licencia MIT es completamente libre para ser usado en todo sentido, incluyendo uso comercial, además es de código abierto lo que permite hacer adiciones y cambios en su código fuente.

Se muestra a continuación datos tomados del blog 'onlyjob' sobre consumo de memoria de Lua comparado con otros lenguajes de programación y ver con qué eficiencia se realiza la recolección de basura. Los datos se los pueden apreciar en las figuras 1.4 y 1.5.

OnlyJob realizó pruebas del rendimiento de memoria de Lua mediante hacer crecer una cadena de caracteres añadiendo cíclicamente más caracteres hasta tener un total de 4Mb, cada iteración substituye texto mediante 'string.gsub', se captura el consumo de memoria cuando el string crece en intervalos de 256 KiB, para la captura del uso de memoria se usa el comando 'memstat' para cada línea impresa en pantalla. El código Lua utilizado por 'onlyJob' se puede encontrar en la figura 1.5.

En la figura 1.3 y 1.4 se puede apreciar la captura de memoria utilizada en pasos de 256Kb que realiza el código de la figura 1.5.

Line size Kb	C (gcc)	C++ (G++)	Perl5	Python	Python3	Ruby	Lua	tcl	PHP	Javascript (sm)	Javascript (V8)	Java (gcj)	Java (OpenJDK)	Java (Sun)
0	1,668	2,932	4,776	5,352	10,328	11,040	2,416	1,236	36,752	7,720	39,272	49,156	72,4832	658,560
256	1,928	3,444	5,052	6,384	13,404	9,620	3,960	13,696	38,040	50,664	47,236	68,320	725,852	661,056
512	2,184	3,956	5,308	5,876	16,476	11,672	5,404	14,720	39,064	29,672	47,636	76,200	725,852	661,056
768	2,440	3,956	5,564	7,676	19,548	7,328	6,428	18,052	40,088	16,872	49,404	84,392	725,852	661,056
1024	2,696	4,980	5,820	6,388	14,420	12,704	7,820	14,716	41,112	53,224	46,540	92,584	725,852	661,056
1280	2,952	4,980	6,076	9,212	15,444	8,604	6,104	15,228	42,136	44,520	47,044	110,072	725,852	661,056
1536	3,208	4,980	6,332	6,900	16,468	11,164	10,572	18,816	43,160	21,480	50,124	118,264	725,852	662,080
1792	3,464	4,980	6,588	7,156	17,492	8,856	11,812	16,252	44,184	38,376	51,916	126,976	725,852	662,080
2048	3,720	7,028	6,844	11,516	18,516	13,724	10,908	16,764	45,208	51,176	47,540	126,976	725,852	662,080
2304	3,976	7,028	7,100	7,668	19,540	12,700	6,644	17,276	46,232	38,376	46,252	161,824	725,852	662,080
2560	4,232	7,028	7,356	7,924	20,564	11,160	15,592	22,912	41,876	41,960	44,452	161,824	725,852	662,080
2816	4,488	7,028	7,612	8,180	21,588	14,748	16,848	18,300	42,388	79,336	50,612	161,824	725,852	662,080
3072	4,744	7,028	7,868	8,436	22,612	15,772	15,716	18,812	49,304	73,704	51,636	161,824	725,852	662,080
3328	5,000	7,028	8,124	8,692	23,636	16,796	19,492	19,324	50,328	39,400	55,996	170,536	725,852	662,080
3584	5,256	7,028	8,380	12,536	24,660	17,820	17,072	19,840	43,924	27,624	46,500	170,536	725,852	662,080
3840	5,512	7,028	8,636	9,204	25,684	18,844	23,276	20,348	44,436	29,160	58,556	170,536	725,852	662,080
4096	5,768	11,124	8,892	9,460	26,708	15,768	20,200	20,860	44,948	96,232	59,836	170,536	725,852	662,080

Figura 1.3 Comparación Uso de memoria tabla de valores

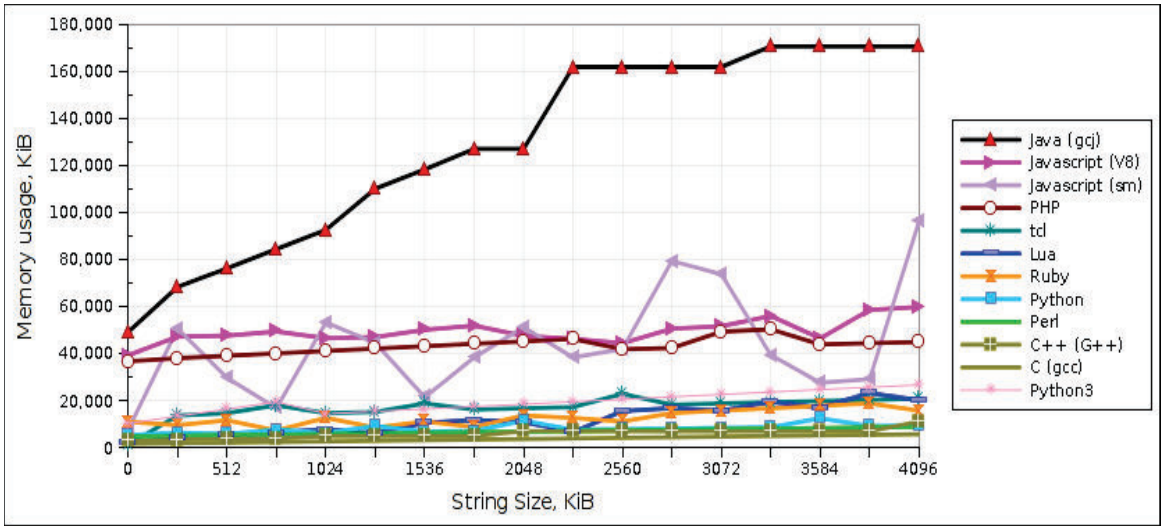


Figura 1.4 Comparación de uso de memoria entre varios lenguajes de programación

Resumen del consumo de memoria de los lenguajes de la figura 1.4:

Muy alto: Java OpenJDK, Java Sun

Alto: Java GCJ

Medio: Javascript V8, Javascript sm., PHP

Bajo: TCL, Lua, Ruby

Muy bajo: Python, Perl5, C++, C

El código utilizado en el caso de Lua fue el de la figura 1.5:

```
#!/usr/bin/lua
io.stdout:setvbuf "no";          --  io.flush();
str='abcdefgh'..'efghefgh';
imax=1024/string.len(str)*1024*4;        -- 4mb
starttime=os.time();
print "exec.tm.sec\tstr.length";
gstr='';
i=0;
while i < imax+1000 do
        i=i+1;
        gstr=gstr..str;
        gstr=string.gsub(gstr,"efgh","____");
        lngth=string.len(str)*i;
        if(math.mod(lngth,1024*256)==0) then
                print(os.time()-starttime.."sec\t\t"..(lngth/1024).."kb");
        end
end
```

Figura 1.5 Código utilizado para pruebas de Consumo de Memoria

Lua ha tenido gran acogida en los últimos tiempos para ser incluido en diversos proyectos y la creciente actividad de proyectos relacionados con el lenguaje Lua en el repositorio GitHub y StackOverflow, estos indicadores se basan en el trabajo, investigación y oferta de empleo para programadores que tienen experiencia con Lua, ver figura 1.6.

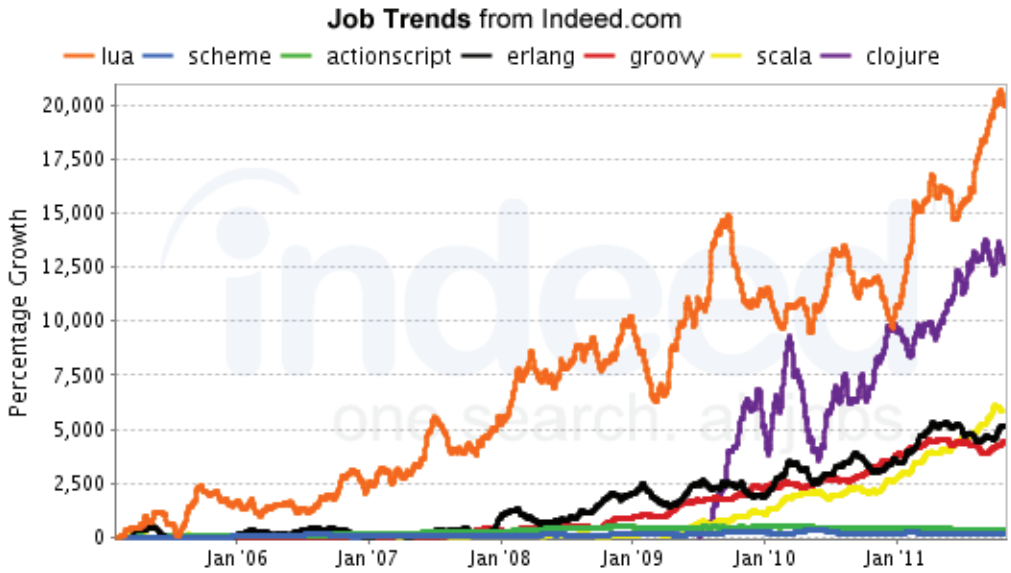


Figura 1.6 Ofertas de empleo los diferentes lenguajes de programación

Con el constante crecimiento que ha tenido Lua lo podemos encontrar entre los más populares del 2015 como se puede ver en la figura 1.7 con un porcentaje del 0.3%.

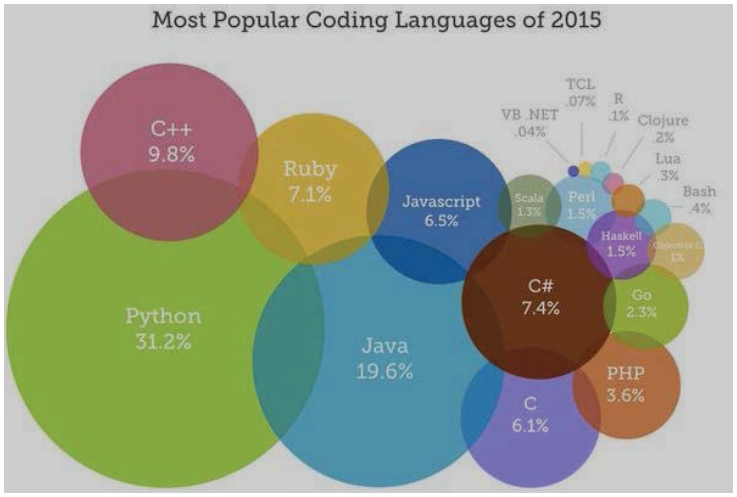


Figura 1.7 Los lenguajes de programación más populares del 2015

1.7 COMUNICACIÓN ENTRE NCL Y LUA

Para que NCL y Lua se puedan comunicar Ginga establece cuatro módulos:

Módulo ‘canvas’: ofrece la posibilidad de controlar gráficos mediante scripts Lua.

Módulo ‘settings’: exporta una tabla con variables definidas por el autor del documento NCL y variables de ambiente reservadas en un nodo “application/x-ginga-settings”.

Modulo ‘persistent’: exporta una tabla con variables persistentes que están disponibles para manipularlas por medio de objetos procedurales.

Módulo ‘event’: permite que las aplicaciones NCLua se comuniquen con el middleware a través de eventos por ejemplo al presionar un botón del control remoto.

Para la comunicación se define mediante una tabla, la misma que se enviará mediante post para comunicar el lenguaje Lua con NCL como se ve en el siguiente ejemplo:

Ejemplo con el tipo ‘presentation’, ver figura 1.8.

```
evt = {
        class = "ncl",
        type = "presentation",
        action = "start"
}
```

Figura 1.8 Evento inicio de presentación

Ejemplo con el tipo ‘attribution’, ver figura 1.9

```
evt = {

        class = "ncl",
        event = "attribution",
        action = "start",
        property = "nombre_de_la_propiedad",
        value = "valor_atribuido"
}
```

Figura 1.9 Evento inicio de 'attribution'

Esta tabla llamada ‘evt’ tiene tres campos ‘class’, ’type’ y ‘action’, la acción que se va a desencadenar es un ‘start’ en la presentación en pantalla.

Los elementos de la tabla tienen las siguientes definiciones:

1. **Clase ‘class’:** las principales clases que se utilizan son ‘key’, ‘ncl’, ‘user’, ‘sms’ y ‘tcp’.
2. **Tipo ‘type’:** puede tener los valores de ‘presentation’ y ‘atribution’.
   - Presentación ‘presentation’: los eventos de presentación controla la exhibición de los nodos de Lua, los mismos que están ligados a las definiciones de comportamiento definidos en NCL.
   - Atribución ‘attribution’: estos eventos de atribución controlan las propiedades de un nudo NCLua.
     - Propiedad ‘property’: nombre de la propiedad asociada al evento.
     - Valor ‘value’: [string] es el nuevo valor que se le va a atribuir a determinada propiedad.
3. **Acción ‘action’:** puede asumir los valores de ‘start’, ‘stop’, ‘abort’, ‘pause’ y ‘resume’.

Los eventos de usuario, teclado y los propios generados por la comunicación entre NCL y Lua, etc. se los debe agrupar mediante una función handler o de manejo, esta función recibe como parámetro el evento.

La función manejo debe ser registrada para poder recibir notificaciones de los de esta manera los eventos pueden ser controlados, como se puede ver en la figura 1.10.

```
function manejo (evt)

        --codigo de la función manejo
end

event.register(manejo)
```

Figura 1.10 Función de manejo de eventos

La comunicación de Lua a NCL se realiza mediante el evento ‘post’ ver figura 1.11.

```
function manejo (evt)
        event.post{
                        class = "ncl",
                        type = "presentation",
                        action = "stop"
      }
end
event.register(manejo)
```

Figura 1.11 Función de comunicación mediante post

La comunicación entre NCL y Lua es muy importante, es necesario capturar eventos desde Lua como por ejemplo reconocer cuándo se presionó un botón o cuándo se desea terminar la aplicación Lua y volver a la aplicación NCL, ver figura 1.12.

```
evt = {

        class = "key",
        type=[‘press’ o ‘release’],
        key="botón"
}
```

Figura 1.12 Evento botones del control remoto

Los valores que son aceptados por el atributo ‘key’ son los siguientes:
“0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”, “A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”, “I”, “J”, “K”, “L”, “M”, “N”, “O”, “P”, “Q”, “R”, “S”, “T”, “U”, “V”, “W”, “X”, “Y”, “Z”, “*”, “#”, “MENU”, “INFO”, “GUIDE”, “CURSOR_DOWN”, “CURSOR_LEFT”, “CURSOR_RIGHT”, “CURSOR_UP”, “CHANNEL_DOWN”, “CHANNEL_UP”, “VOLUME_DOWN”, “VOLUME_UP”, “”, “ENTER”, “RED”, “GREEN”, “YELLOW”, “BLUE”, “BACK”, “EXIT”, “POWER”, “REWIND”, “STOP”, “EJECT”, “PLAY”, “RECORD”, “PAUSE”.
Fuente: Tomado de las Normas ABNT.

CAPÍTULO 2

DISEÑO DE LA APLICACIÓN INTERACTIVA

2.1 ANÁLISIS DE ASPECTOS DE DISEÑO QUE INTERVIENEN EN EL DESARROLLO DE VIDEOJUEGOS

Se seleccionará una metodología diseño de videojuegos apropiada, esto nos servirá para guiarnos dentro de las etapas de desarrollo, nos ayudará a planificar y dividir las tareas para avanzar ordenadamente en las tareas del proyecto y nos proporcionará métricas para poder visualizar el avance del mismo. De la misma manera se eligirá un formato estándar para documentar el código del proyecto.

Dentro del middleware Ginga tenemos dos opciones de lenguajes de programación, una declarativa (NCL) y una impertiva (Lua). Se hará uso de los dos lenguajes de modo que para las partes básicas y sobre todo estáticas de la aplicación se utilizará NCL y para diseñar los algoritmos de movimiento, lógica y las demás partes que componen el motor del juego de video se utilizará el lenguaje Lua.

Se abordarán aspectos escenciales en el diseño gráfico para la creación de los personajes del juego y escenarios, se seguirán las recomendaciones de la empresa CPqD para el desarrollo de aplicaciones interactivas para el estándar ISDB-Tb, y a la par se tomarán en cuenta los diferentes detalles que se puedan apreciar en los juegos de las consolas de donde se están tomando las ideas para desarrollar la dinámica de los tres diferentes niveles para el videojuego del presente proyecto.

La principal diferencia que se encuentran en el desarrollo de software tradicional y el desarrollo de videojuegos es la cantidad de ramas de conocimiento que intervienen en el desarrollo de un videojuego, para lograr una experiencia de juego o *gameplay*.

2.2 SELECCIÓN DE LA METODOLOGÍA DE DESARROLLO Y DISEÑO DEL VIDEOJUEGO

Para el desarrollo de software en este proyecto se tiene como enfoque principal los resultados, se dividirá el proyecto en pequeñas partes secuenciales para solventar todas las tareas de forma incremental y finalmente probar los resultados obtenidos en cada etapa para dar una retroalimentación y tener un buen nivel de calidad en el proyecto.

Para el desarrollo de videojuegos generalmente se utiliza alguna metodología ágil de desarrollo de software, y de entre ellas se ha elegido a SCRUM, el término Scrum proviene de una formación del juego rugby llamada *scrummage* ; se la ha seleccionado ya que permite enfocarse en los resultados (objetivos y directrices del plan del proyecto de titulación) y dividir el proyecto en pequeños pasos que facilitan centrar los esfuerzos en una determinada tarea, solventarla y pasar a la siguiente de forma incremental.

2.2.1 LISTA DE DESEOS, PRODUCT BACKLOG Y SPRINTS

En la metodología Scrum se empieza por crear una lista de deseos que será una lista tentativa de las partes, cualidades y funcionalidades que se desea que tenga el producto final, esta lista puede ser muy extensa y a veces no se apega a las posibilidades reales del producto o las necesidades reales del usuario final; es por esto que en primera instancia la lista de deseos se la analiza y reduce para obtener el llamado Product Backlog.

Una vez obtenido el product backlog en el cual cada ítem se lo llama historias de usuario se lo analiza de forma que se crea un plan de liberación, este plan de liberación contiene las historias de usuario finales que se compromete a cumplir y entregar en el producto final, este backlog final se lo llama backlog de liberación. A continuación se muestra el backlog de liberación del presente proyecto de desarrollo del videojuego.

- Se desarrollará una aplicación interactiva para el estándar ISDB-Tb que utilizará Ginga NCL y Lua, que consistirá en un videojuego.
- El juego constará de tres diferentes niveles en los cuales se hará relación a juegos de diferentes generaciones antiguas de consolas de videojuegos.
- En juego permitirá ingresar un nombre corto del jugador para mostrarle las instrucciones y puntaje en cada nivel del juego.
- El juego hará uso del canal de retorno Ethernet del estándar ISDB-Tb para enviar la puntuación obtenida por el jugador y a la vez traer información de los tres puntajes más altos alcanzados en el juego.
- Se harán pruebas de funcionamiento en receptores reales de televisión digital compatibles con el estándar ISDB-Tb en varias resoluciones.

Una vez que contamos con el backlog de liberación procedemos a descomponer estas historias de usuario en pequeñas tareas técnicas y de esta manera formar los sprints, los sprint tienen la función de descomponer el proyecto en partes que se puedan probar por separado de esta manera se obtiene a la vez un mayor nivel de calidad.

A continuación se presenta el gráfico donde se puede observar los diferentes sprints del código a desarrollar para el videojuego, se ha tenido cuidado de que cada bloque pueda tener un resultado o producto que se lo pueda probar por separado. Al final también se harán pruebas de la aplicación consolidada en la máquina virtual de ginga y en el receptor de tv digital set-top box a diferentes resoluciones.

La prueba de las partes del proyecto por separado ayuda para que se pueda se tenga control un de calidad en todo el proceso y se pueda llevar el seguimiento de las tareas del sprint, es por esto que se puede dar un buen aproximado de la fecha de terminación del proyecto.

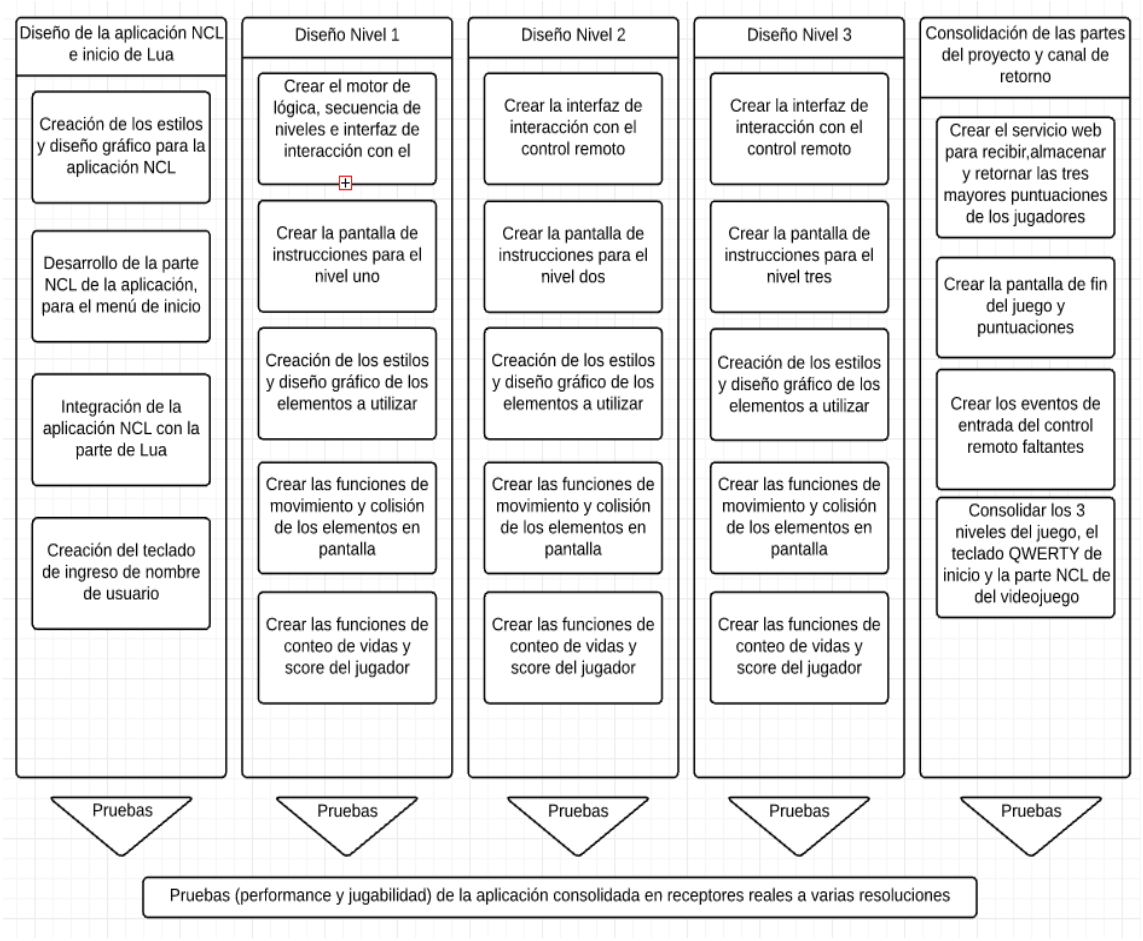


Figura 2.1 Diagrama de Sprints de la metodología SCRUM del proyecto

2.3 EXPERIENCIA DE JUEGO 'GAMEPLAY'

El principal objetivo de un videojuego es captar la atención del jugador, el desarrollador del juego busca crear una experiencia de reto basado en la destreza y la suerte. Hay muchas formas de evaluar el 'gameplay' de un determinado videojuego entre las principales encontramos las siguientes:

- Calidad de los gráficos, la calidad de los gráficos ha evolucionado con la tecnología desde los gráficos basados en vectores, imágenes de 8bits, monocromáticos y a todo color, texturas y modelos 3d, en conjunto son el arte gráfico que se muestra en un juego.
- Jugabilidad, esto se refiere a que tan atractivo es un juego en lo referente al uso de un control y cómo el personaje se desenvuelve en

pantalla, para esto hay que buscar un equilibrio entre dificultad pero sin hacer que el jugador pierda el interés en el juego.

- Trama del juego o también llamada Historia, es el punto principal porque algunos juegos realmente marcan hitos, la trama del juego es lo que lleva al jugador a pasar semanas incluso meses con un mismo juego si capta totalmente su interés. Existen juegos que por su trama en combinación con una buena jugabilidad se han convertido en obras muy apreciadas y no necesariamente han tenido los mejores gráficos.

2.4 DISEÑO VISUAL DE LA APLICACIÓN INTERACTIVA

Diseño visual áreas en la pantalla: En base a las recomendaciones de diseño de las interfaces de usuario es conocido que los usuarios tienen más facilidad reconocer información en las siguientes áreas de la pantalla ver figura 2.2:

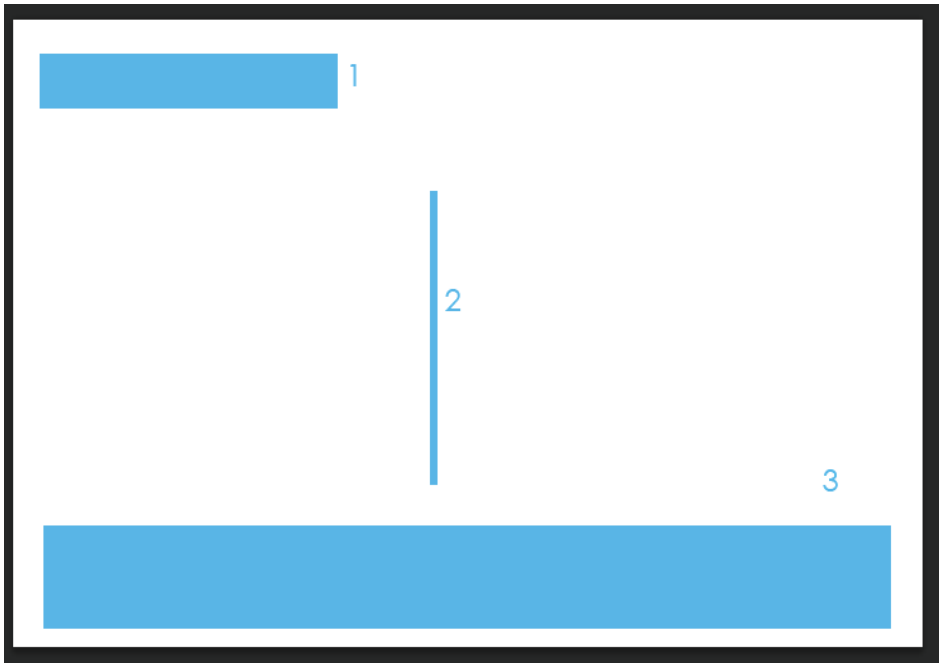


Figura 2.2 Áreas con mayor visibilidad según recomendaciones de CPqD

La aplicación interactiva utilizará para el menú y mostrar información el área 1 y 3.

Ya dentro del juego la aplicación utilizará las siguientes áreas ver figura 2.3:

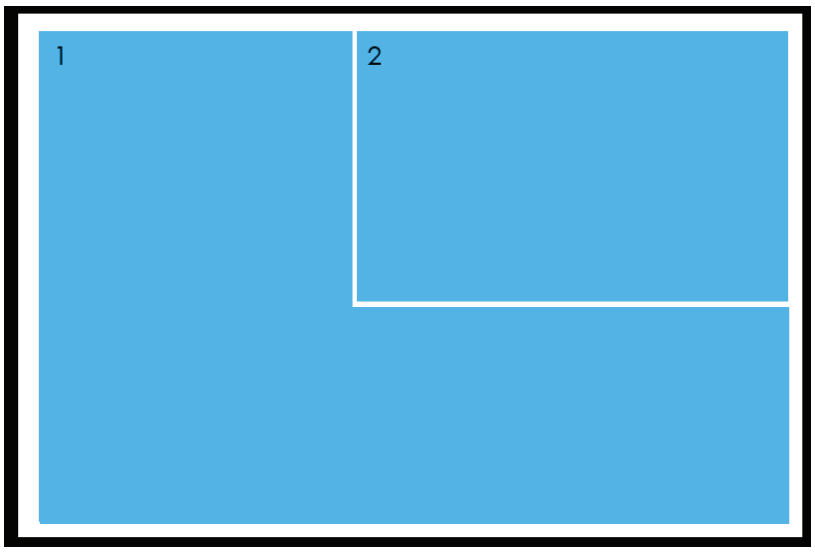


Figura 2.3 Áreas utilizadas en el Juego

El área 2 está destinada a la señal de video del canal de tv digital, el video no tiene relación alguna con el juego a desarrollar. El área 1 está destinada al videojuego, se lo realiza de esta manera para no interrumpir la señal de video de la televisión, tal como lo hacen muchas otras aplicaciones interactivas de otros estándares, la aplicación a desarrollar se mostrará algo similar a la figura 2.4.

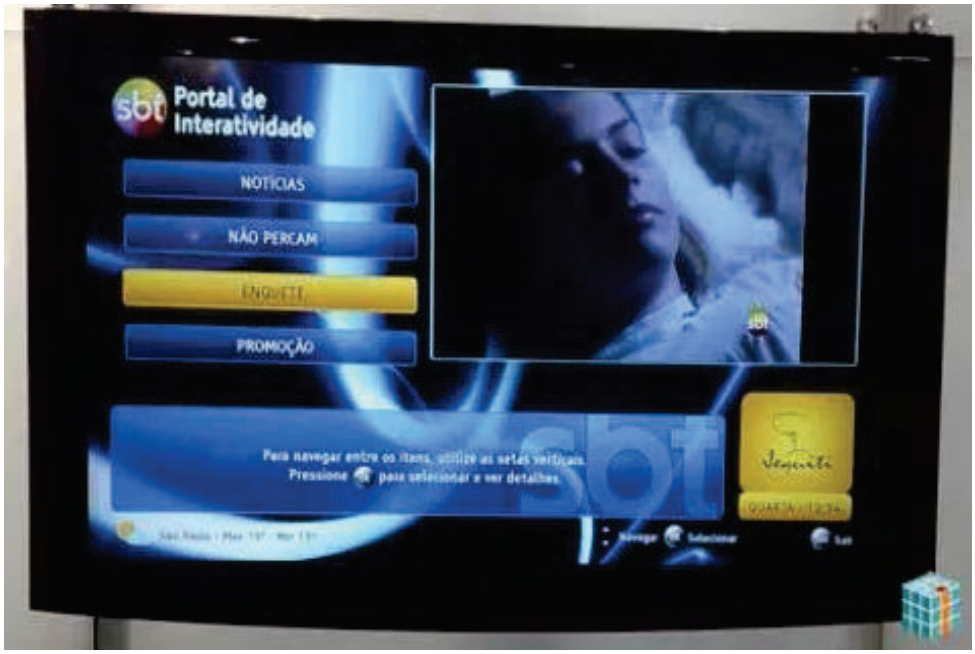


Figura 2.4 Ejemplo de aplicación interactiva ISDBT-b Java

El área 1 se dividirá en dos partes de la siguiente manera, ver figura 2.5:

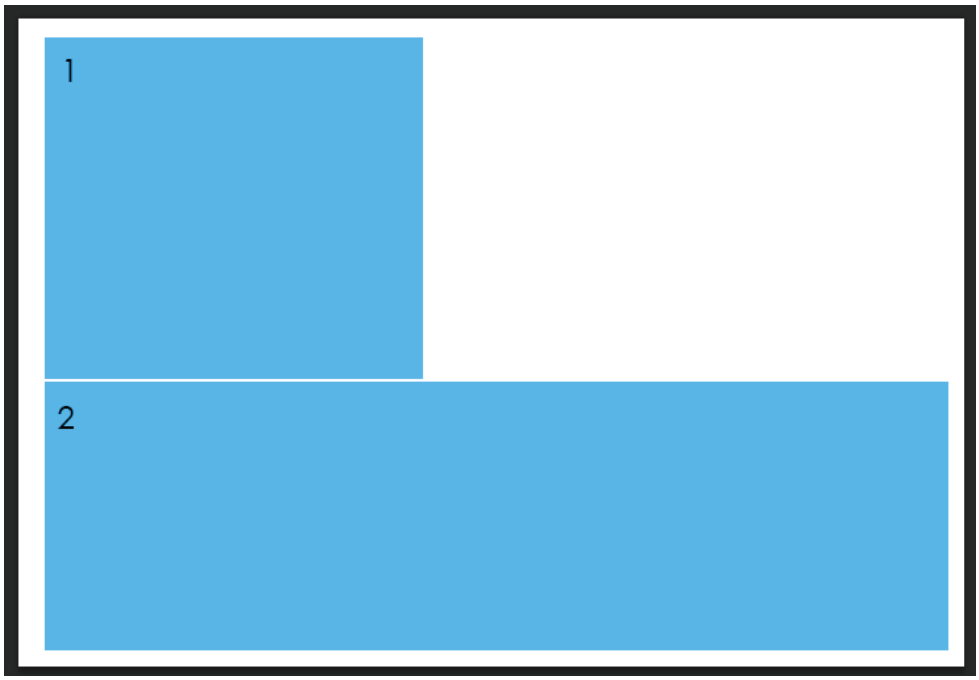


Figura 2.5 Áreas destinadas a la información de Juego

El área 1 en la parte superior de la figura se utilizará para mostrar el nombre del juego y un fondo relacionado al juego, en el área 2 en la parte inferior se desarrollará el juego, dejando un área apropiada para no ocultar o perder el video de la transmisión del canal de televisión digital.

Se toma en cuenta también que se debe dejar un margen en los bordes de la pantalla (5% izquierda y 5% derecha) ya que en las aplicaciones reales estos bordes se pierden.

2.5 MÉTODO DE ENTRADA (TECLADO QWERTY EN PANTALLA)

CPqD realizó pruebas de entrada de información es decir texto en aplicaciones de televisión digital, como resultado de estas pruebas se determinó que para largas cadenas de caracteres de texto era conveniente utilizar un teclado parecido a los que se encuentran en los celulares que tienen solo teclado numérico tal como se muestra en el siguiente gráfico, ver figura 2.6.

Figura 2.6 Teclado numérico Celular

En el estudio realizado los usuarios familiarizados con los teclados celulares, podía ingresar texto de forma rápida utilizando los controles remotos estándar de un receptor ISDB-Tb.

CPqD también determinó que un teclado QWERTY en pantalla también es una opción muy válida aún más si el texto que se va a ingresar es corto, siguiendo el modelo que se ha utilizado en los juegos de video de las primeras consolas, se tenían un método de entrada de texto basado en un teclado QWERTY en pantalla.

En la siguiente figura se puede observar la entrada de texto en un juego de una antigua consola de videojuegos como se puede ver en la figura 2.7.

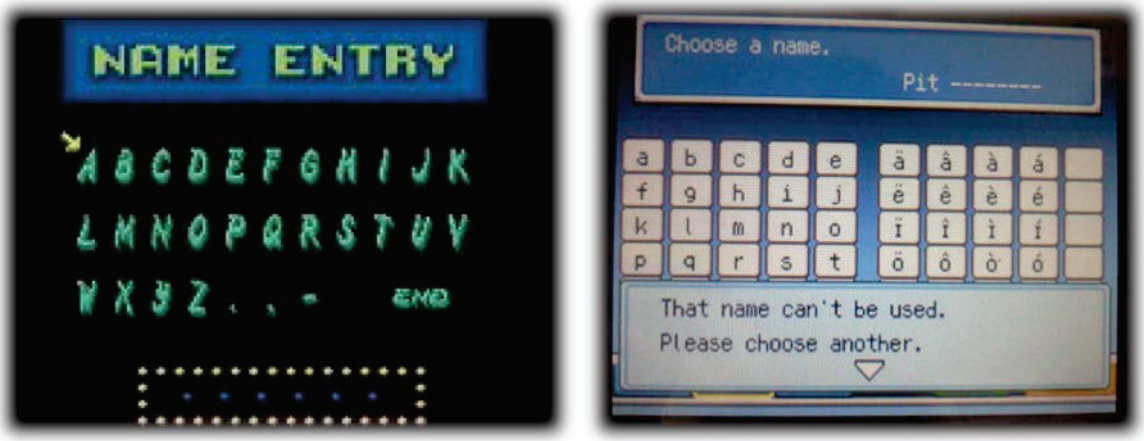


Figura 2.7 Ejemplo de entrada de texto juegos antiguos

Es importante conservar un mismo estilo para navegar dentro de la aplicación es decir los menús deben ser simples estar en lugares visibles dentro de la aplicación para que el usuario no se pierda haciendo que sea sencilla de utilizar dentro de las limitaciones para interactuar mediante un control remoto.

Zonas de navegación:

- **Zona 1** Inicio/final y retroceder (Atrás).
- **Zona 2** Avance o siguiente (esta sección puede ser omitida si el jugador visualiza los mensajes o instrucciones).

CPqD recomienda el uso de las zonas específicas para cada acción, como se ha constatado, hay veces que el usuario también conoce los significados de los colores tal como el verde para seguir o iniciar y el rojo para parar se puede también hacer referencia a establecidas para utilizarlas en el diseño y colores de la aplicación para el middleware Ginga como se puede ver en la figura 2.8.

Figura 2.8 Áreas recomendadas por la CPqD para navegación

2.6 DISEÑO DEL CÓDIGO DE LA APLICACIÓN INTERACTIVA UML

Para documentar el software desarrollado en el presente proyecto se ha seleccionado el Lenguaje de Modelado de Software LUM (Lenguaje Unificado de Modelado) o UML por sus siglas en inglés, debido a que permite visualizar el

cómo los componentes de la aplicación interactúan entre sí para controlar el estado del juego y los elementos que se muestran en pantalla. Dependiendo del número de vidas que tenga el jugador podrá ir accediendo a los tres diferentes niveles que tiene el juego mientras acumula puntos. El código se diseñara de tal forma que se tenga control del estado del juego es decir en qué nivel se encuentra, y también el control del número de vidas y la puntuación lograda por el jugador.

El juego iniciará una vez que el usuario haya revisado las instrucciones del juego, estas instrucciones aparecerán al inicio de la aplicación y también al inicio de cada nivel, tienen la finalidad de mostrar al usuario de que se trata la aplicación, cómo completar los objetivos de cada uno de los niveles y cuáles son los botones del control remoto que tiene que utilizar en cada nivel.

2.6.1 DIAGRAMA DE FLUJO

A continuación se muestra un diagrama de flujo del funcionamiento básico del juego, ver en las figuras 2.9 y 2.10:

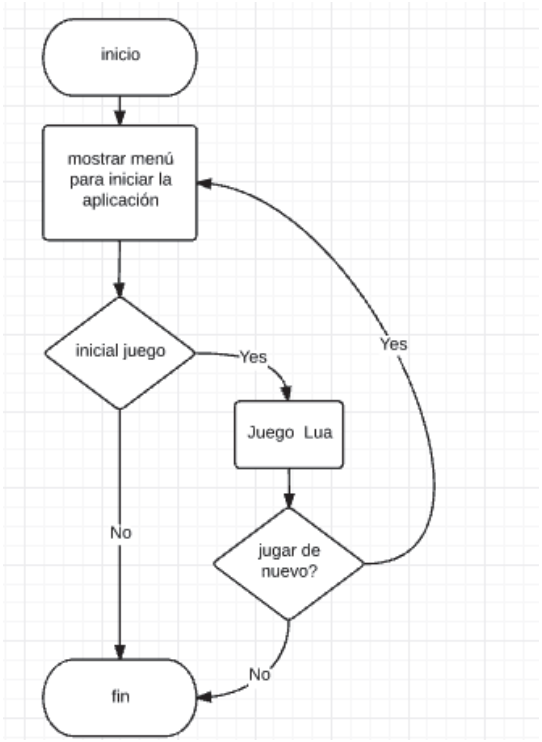


Figura 2.9 Diagrama de flujo de la aplicación interactiva parte NCL

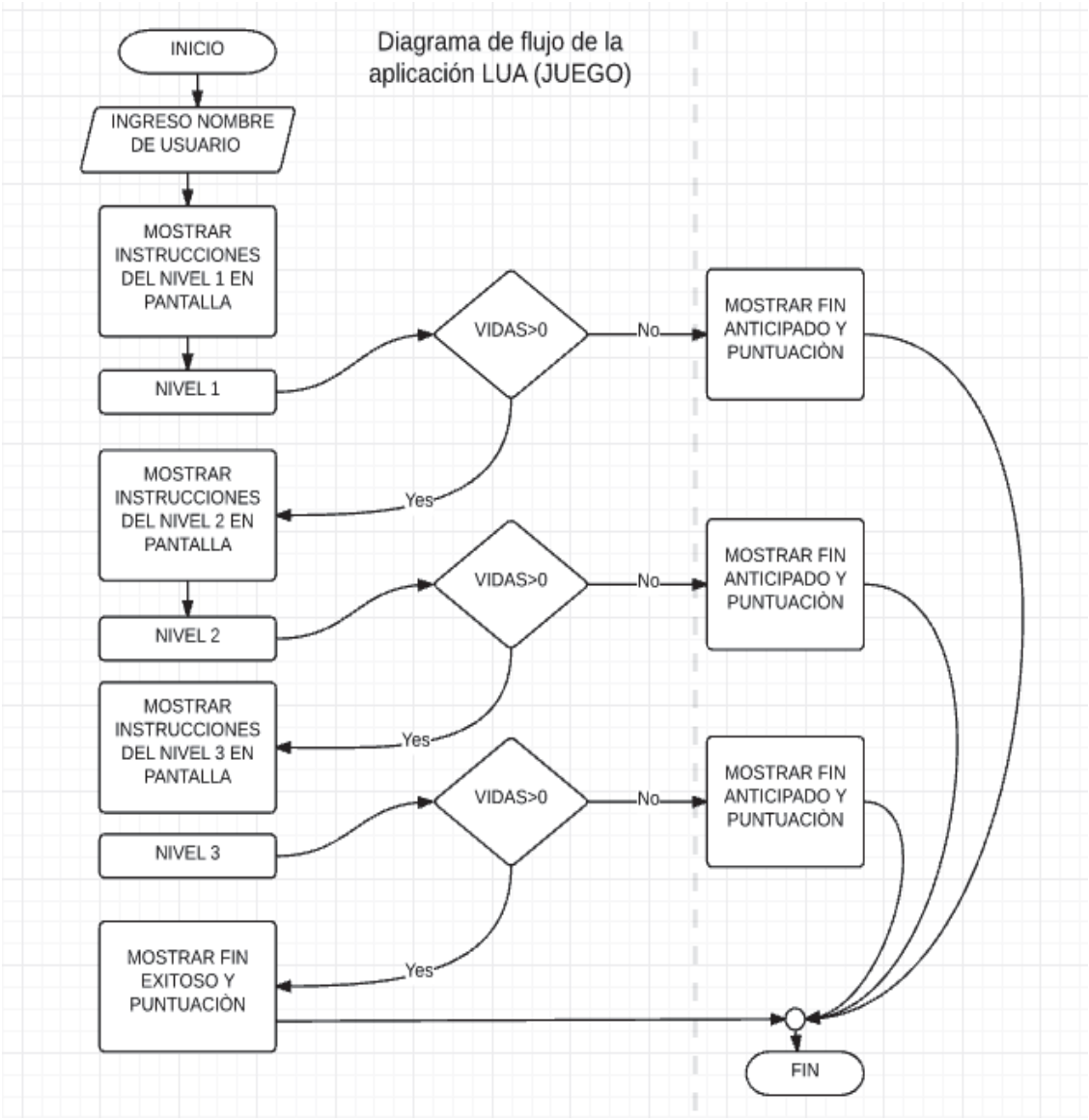


Figura 2.10 Diagrama de flujo de la aplicación interactiva parte LUA

Como se puede observar en la figura 21 el juego está dentro del flujo NCL y este se lo muestra como un elemento Lua, NCL tiene la capacidad de iniciar nuevamente el juego o salir de la aplicación. En la figura 2.10 podemos ver el diagrama de flujo del juego, en el que si disponemos de vidas podemos pasar los 3 niveles, el juego tiene dos posibles finales, uno anticipado si el jugador pierde todas sus vidas o uno exitoso si el jugador completa todos los tres niveles, Lua tiene la capacidad de comunicarse con NCL de esta manera se puede terminar el juego Lua y volver al menú NCL.

2.6.2 DIAGRAMA DE CASOS DE USO

En este diagrama de casos de uso se evidencian las funciones que ejecuta el jugador cuando desarrolla el juego, ver figura 2.11.

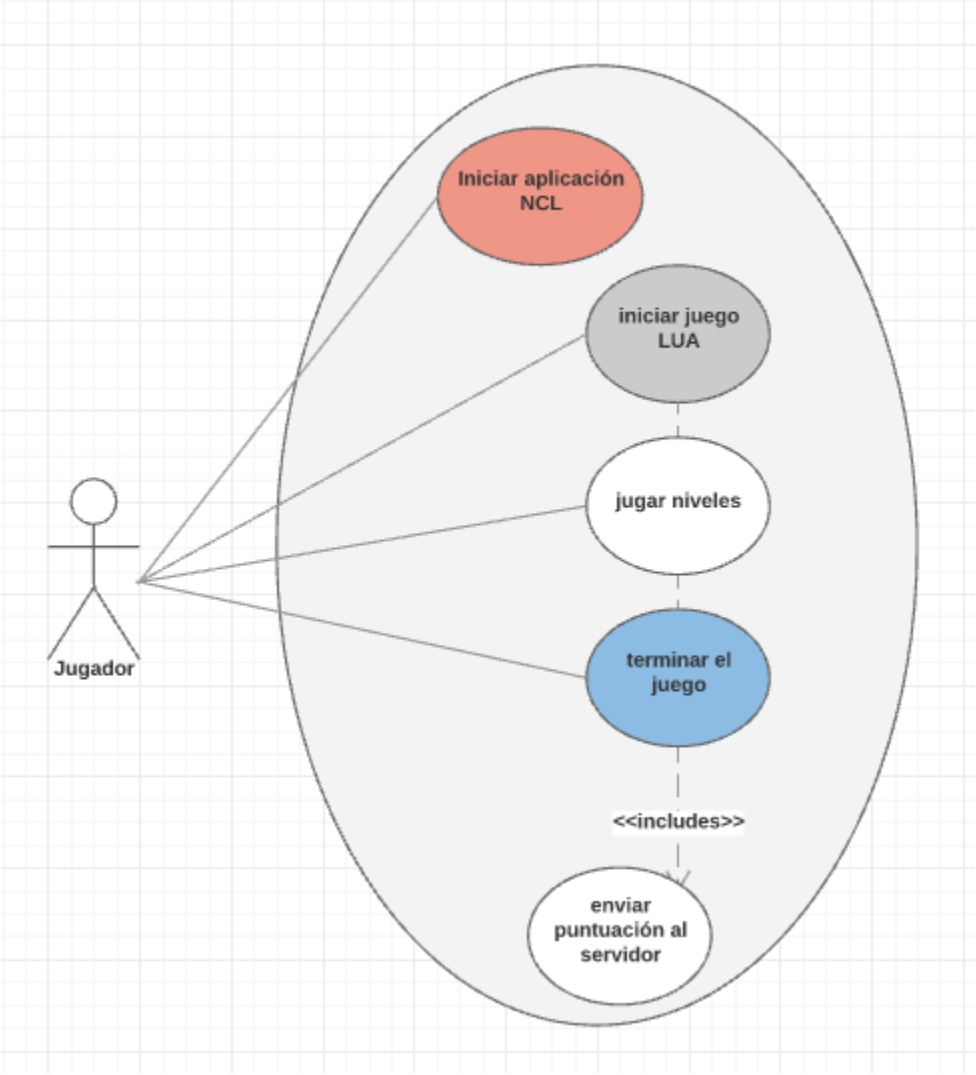


Figura 2.11 Diagrama de flujo de la aplicación interactiva del presente proyecto

2.6.3 DIAGRAMA DE SECUENCIA DE LA APLICACIÓN

En este caso se utilizan las siguientes funciones, ver figura 2.12:

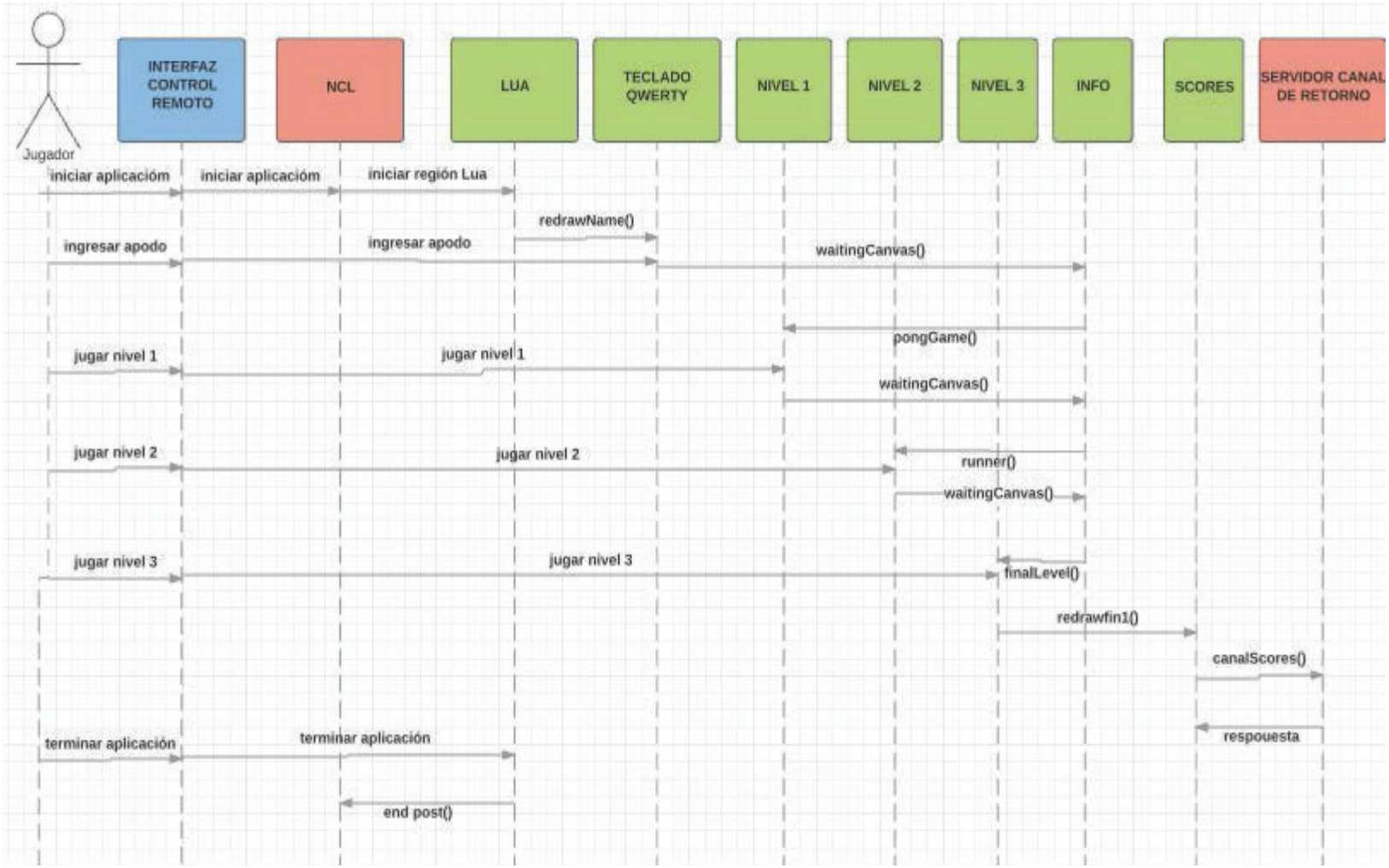


Figura 2.12 Diagrama de secuencia de la aplicación interactiva del presente proyecto

2.7 DISEÑO GRÁFICO

Es importante tomar en cuenta que las imágenes son fundamentales en el diseño gráfico, estas se muestran con un mayor nivel de brillo y saturación en las pantallas de televisión. Para el diseño de las aplicaciones de televisión digital se debe tomar en cuenta que las imágenes diseñadas por computador se deben depurar al máximo es decir en el caso de las imágenes con transparencia no se debe dejar píxeles sueltos en la zona de transparencia ni tampoco bordes no continuos y limpios como se ve en la siguiente imagen, ver figura 2.13:

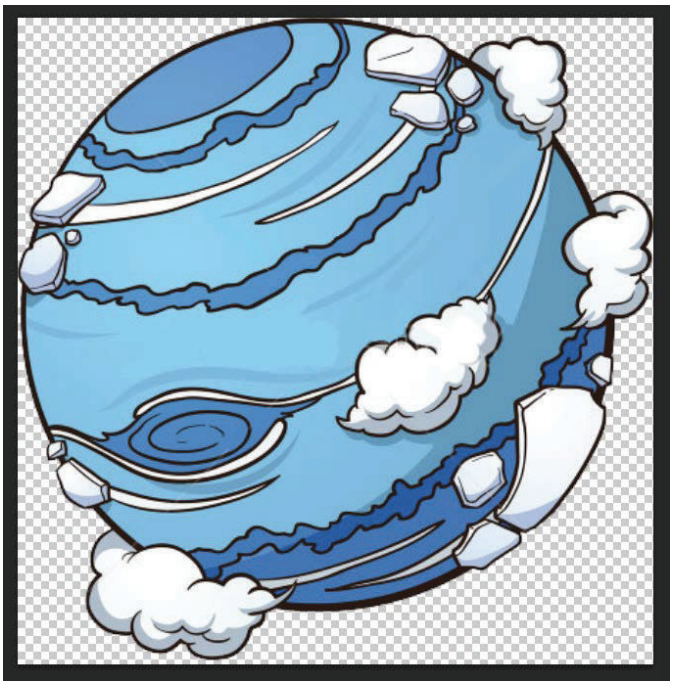


Figura 2.13 Imagen básica para editar en formato JPG o PNG

No se debe trabajar con la imagen cuando hay píxeles que tienen contacto con los bordes del área de trabajo ya que impide que se encuentren las imperfecciones en estas zonas, la imagen puede parecer que está lista para exportarla y ubicarla en la aplicación pero requiere todavía de procesamiento.

Al aplicar el efecto stroke (se traza un contorno de un color específico en toda agrupación de píxeles) en Photoshop vemos que aparecen grumos alrededor de la imagen, ver figura 2.14:

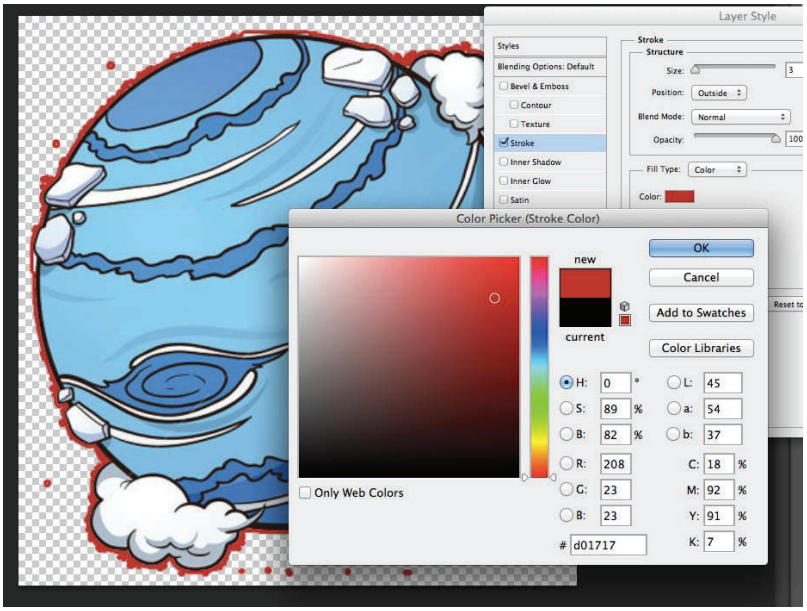


Figura 2.14 Marcar imperfecciones mediante stroke

Como podemos notar la figura redondeada alrededor del planeta debe ser lisa, con la herramienta del stroke y el borrador se puede fácilmente limpiar estas zonas. Fuera de la imagen se tienen píxeles sueltos que no se han borrado, en el monitor de un computador esto es difícil de notar pero cuando esta imagen se muestra en una tv real sobre un video se nota muy claramente.

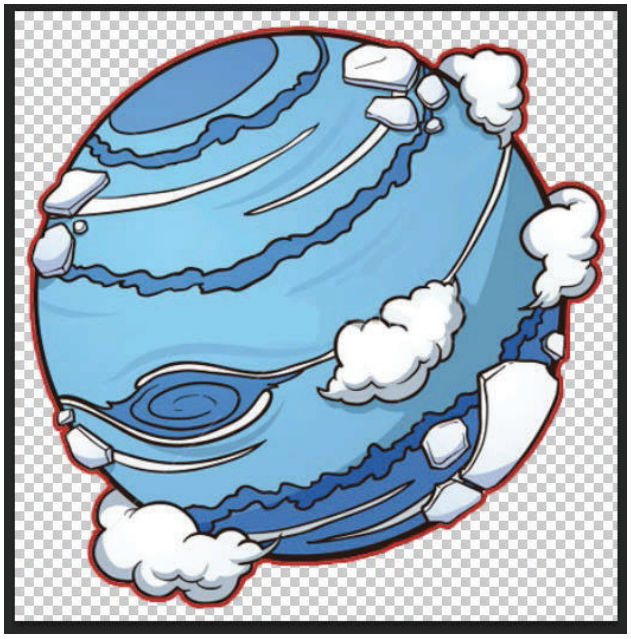


Figura 2.15 Borrado de Imperfecciones

En la figura 2.15 se puede ver cómo queda la imagen luego de retirar todas las imperfecciones del contorno, a continuación se deben hacer pruebas de visualización comprobando que la imagen se distinga correctamente sobre varios fondos dado que pueden aparecer otras imperfecciones tales como píxeles blancos alrededor de exterior negro de la imagen o que se pierda la imagen al fundirse con colores de la señal de video de la televisión digital. Es probable además que por accidente se haya borrado dentro de la imagen en este caso la imagen tendría un hueco por donde se verían colores cambiando pertenecientes a la señal de video de televisión digital, en la figura 2.16 se puede ver la imagen final sobre varios fondos.

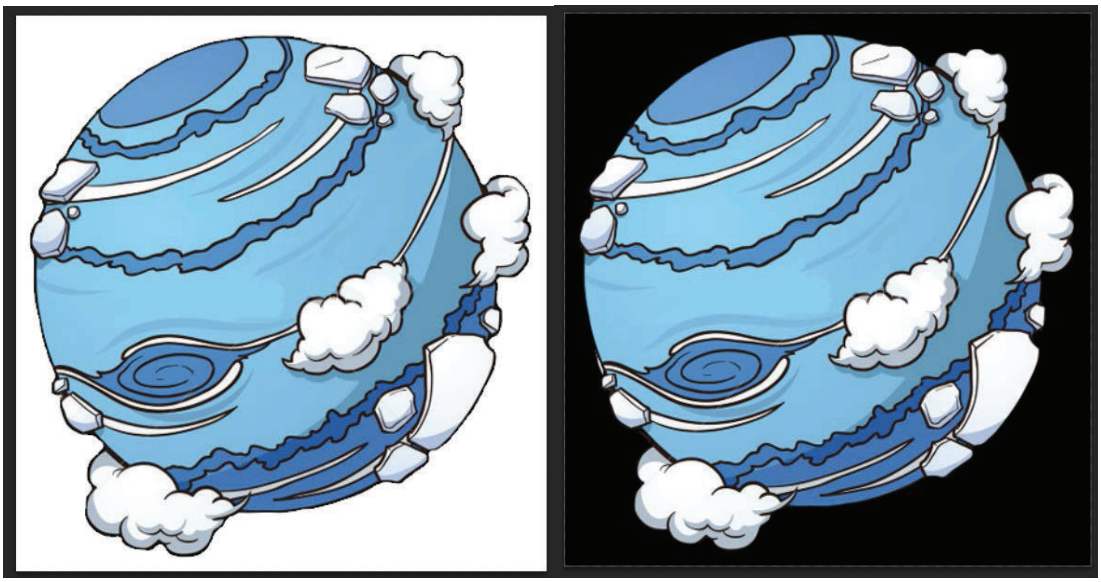


Figura 2.16 Prueba de diseño en diferentes fondos

Una vez que logremos una imagen de buena calidad se le puede agregar texto o situarla en alguna región específica de la pantalla.

La edición anteriormente descrita es muy común al utilizar imágenes rasterizadas (imágenes en formato png, jpg, gif, etc) por el contrario si utilizamos vectores tendremos otro tipo de inconvenientes que tienen que ver como por ejemplo con transparencias, poner bordes continuos a toda la imagen, trabajar con varios grupos de formas geométricas, escalamientos y por último conservar las relaciones de alto y ancho. Al final del trabajo se exportará el vector a un formato “png” con transparencia.

NOTA: Se han hecho pruebas de aplicar una animación de esta imagen directamente en NCL la misma que consta de pasar continuamente tres imágenes simples de forma que se cambie la imagen cada tercio de segundo dando la perspectiva de movimiento de las nubes alrededor del planeta pero se ha comprobado que hay mucha lentitud en cambiar las imágenes, estas imágenes desaparecen en pantalla. Es por esta razón que se comprueba que generalmente NCL en los receptores no aprovecha de manera óptima los recursos de memoria y procesamiento al contrario de LUA en el que tenemos mejor control de las imágenes.

Es muy común en el desarrollo de juegos utilizar funciones de colisión de imágenes que toman en cuenta los píxeles y no el tamaño de la imagen, para hacer estas funciones de colisión mediante píxeles es necesario manipular la imagen a través de código la saturación de todos los píxeles a un solo color, y comprobar la colisión mediante conjuntos de píxeles, estas funciones que son muy livianas en una PC o consola de juegos dedicadas pero no se implementarán en el presente proyecto dado que no se tienen las librerías LUA de manipulación de imágenes para la saturación y obtención de píxeles, pero se ha implementado una función colisionador simple que puede reconocer colisiones mediante un método simple que se detalla a continuación:

```
function colisionador(B, A)

    local ax1, ay1 = A.x, A.y
    local ax2, ay2 = ax1 + A.dx, ay1 + A.dy
    local bx1, by1 = B.x, B.y
    local bx2, by2 = bx1 + B.dx, by1 + B.dy

    if ax2 < bx1  or bx2 < ax1 then
        return false
    elseif ay2 < by1 or by2 < ay1 then
        return false
    end

    --cambios de imagenes
    return true
end
```

Figura 2.17 Código simple para detectar colisiones en Lua

Como se puede ver en la figura 2.17 se muestra el código implementado para detectar colisiones, la función ***colisionador*** utiliza dos parámetros en este caso A y B, A y B son tablas cada una contiene la posición de la imagen en coordenadas (x, y) y también tienen la información de cuántos píxeles mide cada imagen, en la figura 2.18 se puede ver gráficamente como se toman las coordenadas y cómo se comprueba la colisión.

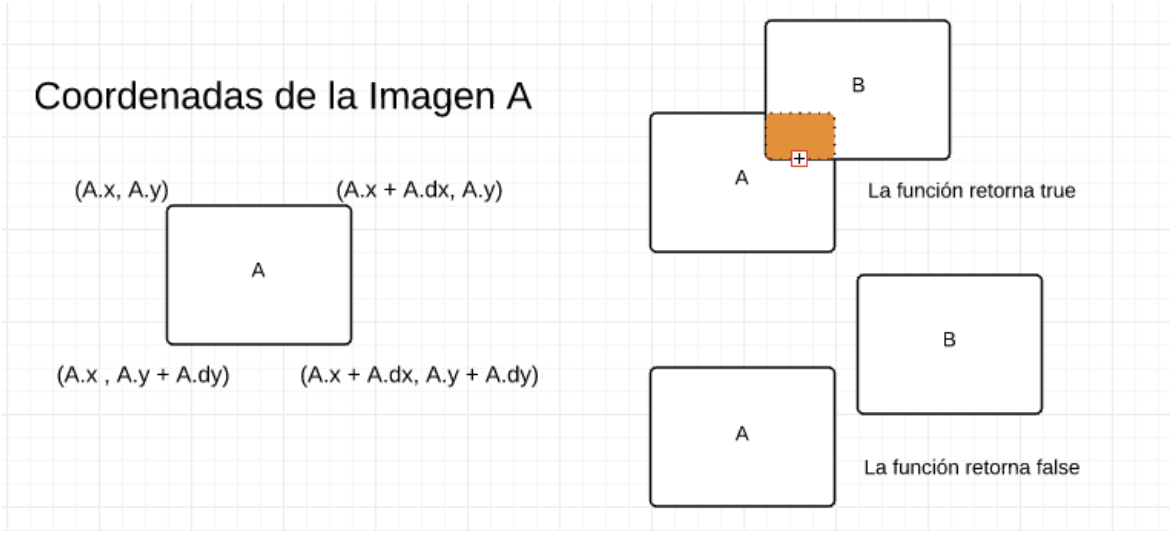


Figura 2.18 Coordenadas de la imagen dentro de la función colisionador

2.8 SPRITES

Para los sprites se ha dibujado la nave que es el personaje principal en varias posiciones que a lo largo del juego van a ser lo que permitan crear la ilusión de que la nave se mueve e interactúa con el escenario del juego, a continuación se puede ver la nave en estado de descenso, ascenso y movimiento horizontal, ver figura 2.19.

Figura 2.19 Sprites animación de la nave en diferentes posiciones

La nave también tiene la opción de cambiar de color al obtener energía o vidas para lo cual se han cambiado algunas líneas y colores como se indica a continuación, cabe señalar que el diseño se lo realiza a nivel de píxeles como se puede observar en la figura 2.20.

Figura 2.20 Diseño de la nave con energía

2.9 INICIO DEL JUEGO

Para iniciar el juego presionaremos el botón azul como se indica en pantalla con el ícono de interactividad ver figura 2.21.

Figura 2.21 Visualización del inicio de la aplicación del presente proyecto

Al iniciar la interactividad se muestra cómo iniciar o salir del juego, ver figura 2.22.

Figura 2.22 Menú inicio del Juego

2.10 DISEÑO DEL MOTOR DE JUEGO

Un motor de juego es el que permite controlar la lógica y la presentación de un determinado videojuego, además de dar ciertas facilidades en lo referente a funciones, librerías y organizar el proyecto, un motor de juego básico se puede componer de las siguientes partes:

- **Renderizador:** es el encargado de mostrar los gráficos en el juego, el renderizador usado en el presente juego es muy básico. Al no disponer de un sistema de GPU dentro del decodificador y dados los escasos recursos de memoria y procesamiento, se toman todos los recursos disponibles para mostrar las animaciones lo más rápido posible, se hace un cálculo muy simple para según la resolución de la pantalla hacer recorrer más píxeles a los personajes y dar la una sensación de movimiento acorde al tamaño de la pantalla.

  En el renderizado es muy importante el orden para mostrar cada elemento gráfico.

- **Assets:** Son los diversos elementos que se utilizarán dentro del juego tales como archivos multimedia, archivos de scripts, librerías, texturas, vectores, elementos 3D, etc.

- **Librerías:** son muy variadas y por lo general son las facilidades reutilizables para el desarrollador tales como: motores de física, reproductores de archivos (imágenes, video, sonido, etc.), detector de colisiones (2D, 3D), proveer interfaces a la GPU, etc. En el presente proyecto se ha tratado de dar ciertas funciones que permiten un desarrollo más rápido tales como, funciones para comunicaciones TCP, detector de colisiones, etc.

- **Storyboard:** es la parte donde se evidencia la secuencia de los niveles del juego o historia, de la misma manera esta puede cambiar dependiendo de los parámetros que cumpla el jugador o la inteligencia artificial, también puede ser lineal como en la mayoría de los juegos antiguos.

- **Eventos de Usuario:** es la parte encargada de obtener y procesar la interacción del usuario que proviene de los controles, es muy importante que sea aislado de todo lo demás porque permite que el juego sea más modular y pueda ser exportado a otras plataformas.

2.10.1 PARTES DEL MOTOR DE JUEGO DENTRO DEL CÓDIGO

- **Renderizador:** en este caso están contenidas dentro da cada una de las funciones correspondientes a graficar cada nivel del juego tales como:

  - **redrawfin1**() Fin exitoso del Juego
  - **redrawName**() Mostrar el teclado en pantalla
  - **waitingCanvas**() Instrucciones del juego
  - **pongGame**() Juego Nivel 1
  - **runner**() Juego nivel 2
  - **finalLevel**() Juego Nivel 3

  - **redrawEndLose**() Fin anticipado del juego

- **Assets:** son las llamadas a los diferentes elementos que se van a mostrar en pantalla en este caso solo imágenes.

  - Ejemplo : local nave = canvas:new('mini2.png')

- **Librerías:** Utilidades tales como *function collide (A, B)* que permite saber si han colisionado dos imágenes o *require "tcp"* que permite transmitir datos por la red mediante TCP.
- **Storyboard**: permite controlar la secuencia de los niveles dentro del juego como se puede ver en la siguiente figura:

```
if estado == 'intro' then
    redrawName()

elseif estado == 'game0' then
    --NIVEL 1
    pongGame()
    sigEstado = 'runner'
elseif estado == 'runner' then
    --NIVEL 2
     runner()
elseif estado == 'finalLevel' then
    --NIVEL 3
    finalLevel()
elseif estado == 'endg1' then
    --FIN EXITOSO
    redrawfin1()
elseif estado == 'endg1mal' then
    --FIN FALLIDO
    redrawEndLose()
elseif estado == 'waiting' then
    --MENSAJES E INSTRUCCIONES ENTRE NIVELES
    waitingCanvas()
end
```

Figura 2.23 Código simple para manejar el Storyboard del Juego

Eventos de Usuario: son las funciones que se disparan al momento de que hay un evento en este caso de presionar un botón del control remoto y en este caso se lo captura mediante el siguiente código:

```
If(evt.class == ‘key’ and evt.type == ’press’) then
        --*Código de uso del teclado
end
```

Figura 2.24 Código para capturar eventos de botones del control remoto

2.11 DISEÑO DE LAS INSTRUCCIONES DE INICIO EN NCL

Basado en el diagrama de flujo de la figura 2.10 se ha planificado los mensajes que aparecerán en pantalla, se ha seleccionado una paleta de colores base para el diseño de los elementos de la interfaz usuario, se han establecido estilos de botones, fuentes (tipos de letra), todo esto con el fin de tener homogeneidad a través de toda la aplicación.

Para ubicar la posición de los elementos en pantalla realizar se utilizó los programas Composer Ginga de PUC-Río y la herramienta de autoría de la CpqD para la generación de aplicaciones interactivas Ginga NCLua, las herramientas antes mencionadas tienen la ventaja de que nos permiten visualizar el lugar y el tamaño de la imagen similar a la que veremos en la televisión real. CPqD recomienda utilizar estilos de letras legibles, evitar combinarlos con colores de fondo que puedan dificultar su lectura, evitar el uso de letras muy delgadas, en cursiva o con un excesivo grueso ya que no se van a poder leer desde la distancia normal en la que se mira la televisión.

2.12 DISEÑO DEL TECLADO EN PANTALLA QWERTY

Para almacenar todos los caracteres que se van a utilizar mediante el teclado QWERTY se utilizará una matriz de dos dimensiones (4x11) de esta manera va a ser más fácil que se puedan seleccionar los caracteres del teclado en pantalla ya que se tendrá una correspondencia directa con el teclado en pantalla de la figura 2.25.

Figura 2.25 Imagen del teclado desarrollado para el presente proyecto

Nótese que en el teclado se tiene un doble espacio para ENTER y el resto de espacio se ha llenado con caracteres especiales, también se dispone del botón 'DEL' para borrar si se ha ingresado mal los caracteres para formar el nombre o apodo del jugador.

Para seleccionar un determinado carácter de la pantalla se utilizará un cursor (también llamado pointer) que tiene un color rojo llamativo para ubicarlo debajo del carácter deseado, el jugador puede mover el cursor a través de la pantalla mediante los botones de flechas del control remoto (UP, DOWN, LEFT, RIGHT) y cuando esté

sobre el carácter debe presionar el botón ‘OK’ del control remoto para seleccionarlo e ir ingresando el nombre del jugador para mostrar las instrucciones del juego de video.

En el siguiente código se puede observar cómo se agregan al array (4x11) todos los caracteres del teclado QWERTY, estos caracteres se los agregan uno por uno mediante lazos for, ver figura 2.26:

```
local img = canvas:new('keyboard.png')
local dx, dy = img:attrSize()
local keyn = { img=img, x= 130 , y=50, dx=dx, dy=dy }
local img = canvas:new('pointer.png')
local dx, dy = img:attrSize()
local pointer = { img=img, x=130 , y=60, dx=dx, dy=dy }
local ix, jx = 0
local keycodes={}
local linekey={'0','1','2','3','4','5','6','7','8','9','|'}

   for jx=0,10 do
        keycodes[jx] = {}
      keycodes[jx][ix] = linekey[jx + 1]
   end
   ix = ix + 1
   linekey={'Q','W','E','R','T','Y','U','I','O','P','Ñ'}

   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
        ix = ix + 1
    linekey={'A','S','D','F','G','H','J','K','L','!','!'}

   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
        ix = ix + 1
   linekey={'Z','X','C','V','B','N','M','@','.','_','-'}
   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
```

Figura 2.26 Creación del Array del teclado en Pantalla

En el código de la figura 2.27 se muestra cómo se desplaza el cursor sobre la imagen del teclado QWERTY de la figura 2.25, en el código se puede apreciar cómo

el cursor se desplaza a través de la matriz “keycodes” (donde están almacenados los caracteres) mediante los índices pk1 y pk2:

```
if (evt.class == 'key' and evt.type == 'press') then
        if estado == 'intro' then
                if evt.key == 'CURSOR_UP' then
                        pk1 = pk1 - 1
                        if pk1 < 0 then
                        pk1 = pk1 + 1
                        else
                        pointer.y = pointer.y - 40
                        end
                elseif evt.key == 'CURSOR_DOWN' then
                        pk1 = pk1 + 1
                        if pk1 > 3 then
                                pk1 = pk1 - 1
                        else
                                pointer.y = pointer.y + 40
                        end
                elseif evt.key == 'CURSOR_LEFT' then
                        pk2 = pk2 - 1
                        if pk2 < 0 then
                        pk2 = pk2 + 1
                        else
                        pointer.x = pointer.x - 38.18
                        end
                elseif evt.key == 'CURSOR_RIGHT' then
                        pk2 = pk2 + 1
                        if pk2 > 10 then
                        pk2 = pk2 - 1
                        else
                        pointer.x = pointer.x + 38.18
                        end
                elseif evt.key == 'ENTER' then
                        if keycodes[pk2][pk1] == '|' then
                        name = string.sub(name, 0, string.len(name) - 1)
                        elseif keycodes[pk2][pk1] == '!' then
                          estado = 'game1'
                          event.post('in', { class='user', time=now })
                        else
                                if string.len(name) == 15 then
                                else
                                name = name..keycodes[pk2][pk1]
                                end
                end
end
```

Figura 2.27 Manejo del puntero sobre el teclado

En la figura 2.27 se puede ver el código mediante el cual se desplaza el cursor sobre el teclado QWERTY, como podemos notar a cada evento de las teclas (UP, DOWN, LEFT, RIGHT) se resta o se suma una constante 38.18 en la posición del eje X o Y según corresponda, de la misma manera se recorre la matriz (4x11) pero en este caso se aumenta o se disminuya una posición dentro del arreglo bidimensional.

También se puede observar que cuando se presiona el botón 'ENTER' se reconoce si se elige el carácter '|' para borrar el último carácter almacenado en la variable 'name'. Si se elige el carácter '!' se indica que ya se terminó de ingresar el nombre o apodo del usuario, una vez hecho esto el juego pasa al estado 'game1' comenzando el juego en el primer nivel.

Si no se han cumplido las dos primeras condiciones para los caracteres "|" o "!" simplemente se agrega el carácter seleccionado en la matriz (4x11) al final del string en la variable 'name'.

2.13 DISEÑO DE LOS NIVELES DEL JUEGO

2.13.1 NIVEL 1

Para este nivel se ha tomado como base el primer video juego creado de consumo masivo que es similar al que hoy se conoce como Pong, este juego es muy antiguo y su primera implementación se la realizó utilizando la pantalla monocromática de un osciloscopio en estos tiempos estos juegos se realizaban como experimentos o pasatiempo por entusiastas de la electrónica y la computación.

Pong se trata de un juego similar al Tenis desarrollado por Allan Alcorn y comercializado en 1972 por Atari, en este juego se utilizan dos elementos gráficos (paletas) en los extremos de la pantalla sobre los cuales rebota una pelota, si la pelota no es interceptada en su recorrido se obtiene una penalización y su adversario obtiene un punto a favor. El juego pong fue muy difundido y tuvo una consola dedicada totalmente a este juego, los controles de este juego eran muy simples ya que podían ser dos botones, una palanca o una perilla como se puede ver en la figura 2.28:

Figura 2.28 Consola del juego antiguo conocido como Pong

En el siguiente gráfico se observan las instrucciones para el juego (nivel 1):

Figura 2.29 Instrucciones de juego para el nivel 1

En este juego en vez de tener una pelota se tiene un proyectil de fuego, el jugador debe utilizar un rectángulo de color verde para controlar el rebote de este proyectil y hacer que el monstruo contrario se queme con el proyectil enviado hacia su

dirección, el jugador obtendrá puntos mientras que el monstruo contrario se esté quemando, ver figura 2.30.

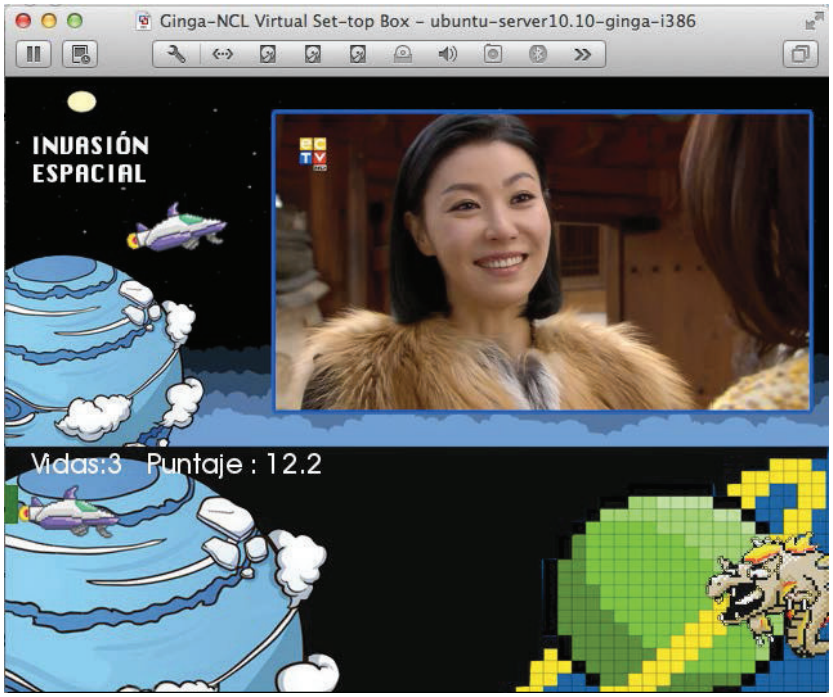


Figura 2.30 Se observa el adversario siendo quemado por el proyectil

El código implementado para este nivel consiste en mover el proyectil con una determinada velocidad en el eje X y Y, cuando el proyectil choca contra un borde de la región Lua se cambia la dirección del proyectil correspondiente a dicho eje.

En el siguiente extracto de código (Fig. X) se evidencia cómo se cambia la dirección del proyectil en el eje Y:

```
if (balls.y + bsy < totaly and balls.y + bsy > 0  ) then
    bsy = bsy + vsy
else
    vsy = - vsy
    bsy = bsy + vsy
end
```

Figura 2.31 Código para cambio de dirección del proyectil

La variable totaly tiene el valor de la longitud total de la región Lua en el eje Y, el proyectil se mantiene aumentando su posición en intervalos constantes dada por la variable vsy, si el proyectil topa con un borde (superior o inferior) se multiplica el valor

de vsy por -1 de esta manera el proyectil cambia de sentido en dicho eje. Se tiene un código similar para el eje X.

Si el proyectil choca con el borde izquierdo significa que la nave no hizo rebotar el proyectil con la paleta verde y pierde una vida. En cambio si la nave hace rebotar el proyectil con la paleta verde esta rebota antes de topar con el borde izquierdo para comprobar esto utiliza una función similar al de la figura 2.31, de la misma manera se lo hace para comprobar si el proyectil colisiona con el monstruo ubicado en la parte derecha de la pantalla.

El movimiento de la imagen se controla por separado en el eje X y en el eje Y como podemos notar la esquina superior izquierda corresponde a las coordenadas (0,0). En la siguiente figura se puede ver el código utilizado para controlar el movimiento de la imagen en el eje X, ver figura 2.32:

```
if (balls.x + bsx < totalx and balls.x + bsx > 0  ) then
    bsx = bsx + vsx
else
    vsx = - vsx
    bsx = bsx + vsx
    if(vsx > 0)then  -- si la pelota choca contra la pantalla izquierda
        vidas = vidas - 1
        if vidas < 1 then
            estado = 'endg1mal' -- si se terminan las vidas finaliza el juego
        else
        end
    else
        -- si el proyectil rebota contra la pared derecha
        colisionesMonster = colisionesMonster + 1
        if (colisionesMonster > 3 ) then
            estado = 'waiting'
        end
        --puntaje1 = puntaje1 + 5
    end
end
```

Figura 2.32 Código utilizado para controlar la dirección en el eje X

Como se puede observar en el código anterior se tiene una funcionalidad similar al eje Y con la diferencia que en este eje se controlan las vidas de la nave y del monstruo. Cada vez que el proyectil no se lo haga rebotar se perderá una vida, al

perder todas las vidas el juego pasa a estado ‘endg1mal’ y se procede a mostrar la pantalla de fin de juego anticipado como se muestra en la figura 2.32.

2.11.2 NIVEL 2

Para este Nivel se ha tomado como base el juego “CIRCUS CHARLIE” lanzado el año 1984, este juego recreaba un espectáculo de circo en el cual el personaje hacía malabares y sorteaba obstáculos, era uno de los juegos más conocidos de la plataforma NES y Arcade, se puede apreciar una de sus escenas cuando el personaje (pequeño payaso) salta sobre los obstáculos que en este caso son monos que vienen caminando en dirección contraria. Si el payaso no salta en el momento adecuado y llega a colisionar con el mono, el mono cae de la soga en la cual se está equilibrando como se puede ver en la figura 2.33:

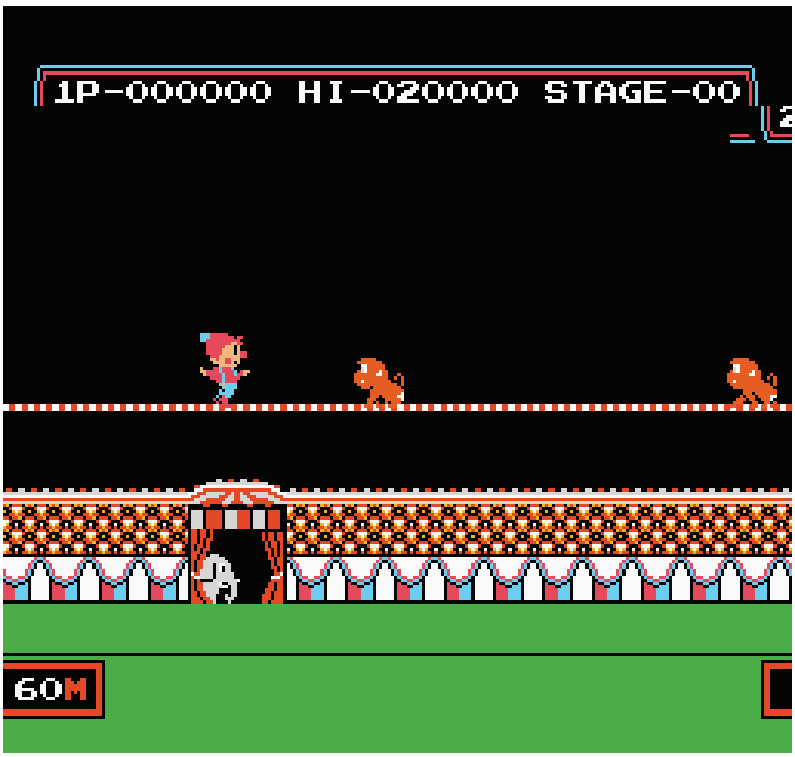


Figura 2.33 Escena del juego “Circus Charlie” nivel 00 salto de obstáculos

En este proyecto, se ha diseñado un juego similar a “Circus Charlie” pero en este caso la nave debe sortear tanques de guerra, los tanques aparecen de forma aleatoria, hay ocasiones en que la nave debe mantenerse en vuelo para no chocarse

con los tanques, la dificultad consiste en que la nave debe reabastecerse de energía y para eso debe tener contacto con el piso, la nave debe reabastecerse cuando tenga la oportunidad es decir cuando los tanques se encuentren separados unos de otros una distancia adecuada para que la nave no se choque con ellos.

A continuación se muestra las instrucciones para el nivel 2:

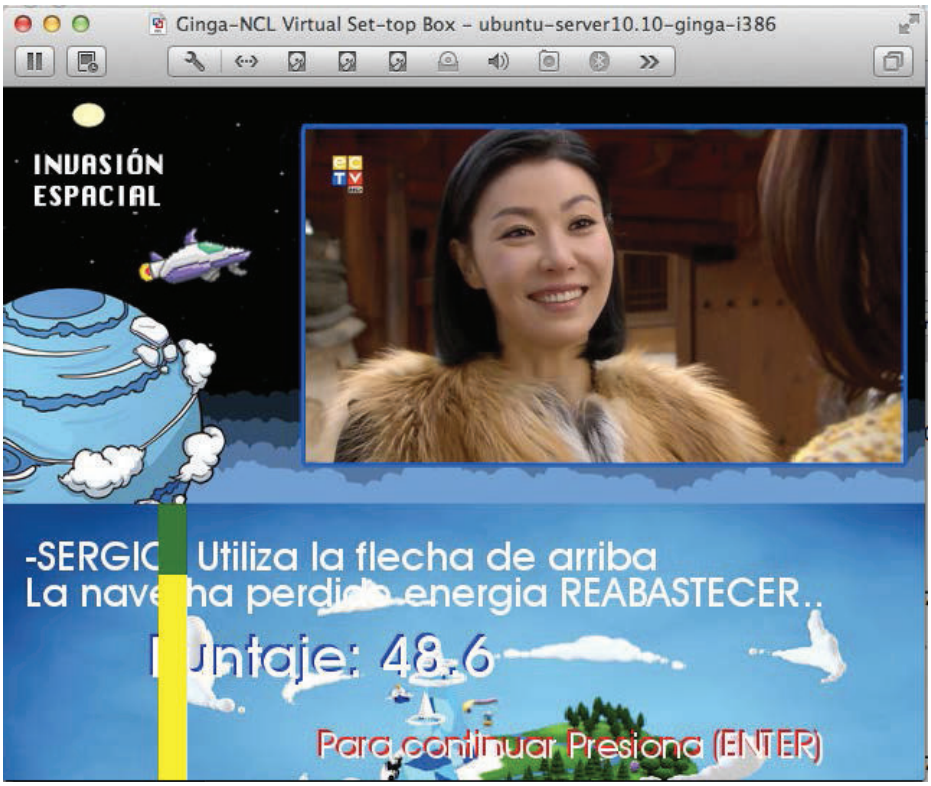


Figura 2.34 Instrucciones de juego para el nivel 2

Como se muestra en la figura 2.34 los tanques se acercan mientras que la nave se encuentra en la estación abasteciéndose de energía, si los tanques se acercan demasiado la nave debe alzar vuelo (presionar botón UP) y esperar que los tanques dejen un espacio para que la nave pueda aterrizar de nuevo (dejar de presionar el botón UP), Los sprites que se utilizan dan la idea de que la nave aterriza, se carga, alza vuelo y desciende de nuevo, además se tiene un contador de la energía el cual debe llegar al 100% una vez que se ha reabastecido hasta el 100% la nave automáticamente alza vuelo, desaparece de la escena y se continúa con las instrucciones iniciales del nivel 3.

Figura 2.35 Escena del nivel 2 donde la nave debe saltar los tanques y pasar el laberinto para abastecerse de energía

Al terminar este nivel como recompensa a los logros alcanzados se premia al jugador con cuatro vidas extras que serán necesarias en el siguiente nivel ya que tiene un mayor grado de dificultad. En el siguiente código se muestra cómo se comprueba la colisión de la nave (variable run) con los tanques (variables rock1, rock2, rock3) en cuyo caso se resta una vida a la nave y la imagen recorre de nuevo desde el borde derecho de la pantalla, las imágenes rock1 y rock2 aleatoriamente se cambian a las figuras en forma de L (rockra, rockrb) superior e inferior para dar la impresión movernos en un laberinto.

```
-- Comprobar choque con tanques
if collidel(rock1 , run) then
    vidas = vidas - 1
    rock1.x = totalx
    local nn = math.random(3)
    if(nn==1)then
        rock1.img = rock
        rock1.y = 120
    end
    if(nn==2)then
        rock1.img = rockra
        rock1.y = 0
    end
    if(nn==3)then
        rock1.img = rockrb
        rock1.y = 120
    end
end
if collidel(rock2 , run) then
    vidas = vidas - 1
    rock2.x = totalx
    local nn = math.random(3)
    if(nn==1)then
        rock2.img = rock
        rock2.y = 120
    end
    if(nn==2)then
        rock2.img = rockra
        rock2.y = 0
    end
    if(nn==3)then
        rock2.img = rockrb
        rock2.y = 120
    end
end
if collidel(rock3 , run) then
    vidas = vidas - 1
    rock3.x = totalx
end
```

Figura 2.36 Código para detectar colisiones con los tanques o laberinto

En el siguiente código se puede apreciar el uso de los diferentes sprites de la nave en carga (run.img), normal (shipNormal.img) y en descenso (shipDown.img). Al mostrar estas imágenes en pantalla se da la apariencia de un movimiento más realista.

```
-- posición más baja
if(run.y < 110) then
    run.y = run.y +1
    canvas:compose( run.x , run.y  , shipDown.img)
else
    -- Animación de la nave en tierra
    if(normalPower<16) then
        canvas:compose( run.x , run.y  , run.img)
        normalPower = normalPower + 1
    else
        canvas:compose( run.x , run.y  , shipNormal.img)
        normalPower = normalPower + 1
        if(normalPower==32)then
            normalPower = 0
            energiaRunner = energiaRunner + 4
            puntaje1 = puntaje1 + 4

        end
    end
end
```

Figura 2.37 Código para controlar la carga de energía y gráficos de la nave

Como se puede notar en el código anterior se hacen rotar dos imágenes en la carga de la nave una mientras un contador es menor a 16 y la otra si el contador está entre 16 y 32.

2.11.3 NIVEL 3

Para este nivel se ha tomado como referencia el juego “Total Carnage” (o conocido en español como Matanza Total) de la consola SNES y Atari lanzado el año 1994, en este juego el personaje principal se enfrenta a un gran número enemigos que lo persiguen, para escapar el personaje principal dispara y pone trampas a sus enemigosque lo persiguen, tratando de que el número de enemigos disminuya como se puede ver en la figura 2.38:

Figura 2.38 Escena del juego “Total Carnage” de SNES y Atari

Se ha tomado la idea básica de Total Carnage pero se la ha modificado a la temática de Invasión Espacial, de manera que la nave no puede disparar y su único recurso para disminuir el número de sus oponentes sea el de ponerles trampas, la trampa en este caso consiste en atraerlos hacia un proyectil de fuego que deja suspendido en el aire.

Un aspecto importante de “Total Carnage” y que lo hace diferenciar de los demás juegos es el algoritmo que permite que los enemigos mantengan una especia de formación, ya que si todos se unen en una región muy pequeña pueden morir una mayor parte de ellos en una sola trampa, en el presente proyecto se ha diseñado el nivel de manera que los enemigos en este caso ‘invasores espaciales’ mantengan el mayor tiempo posible su formación original mientras que persiguen a la nave.

Como se puede observar en la figura 2.39, se ha creado un array que contiene las imágenes de los enemigos de la nave.

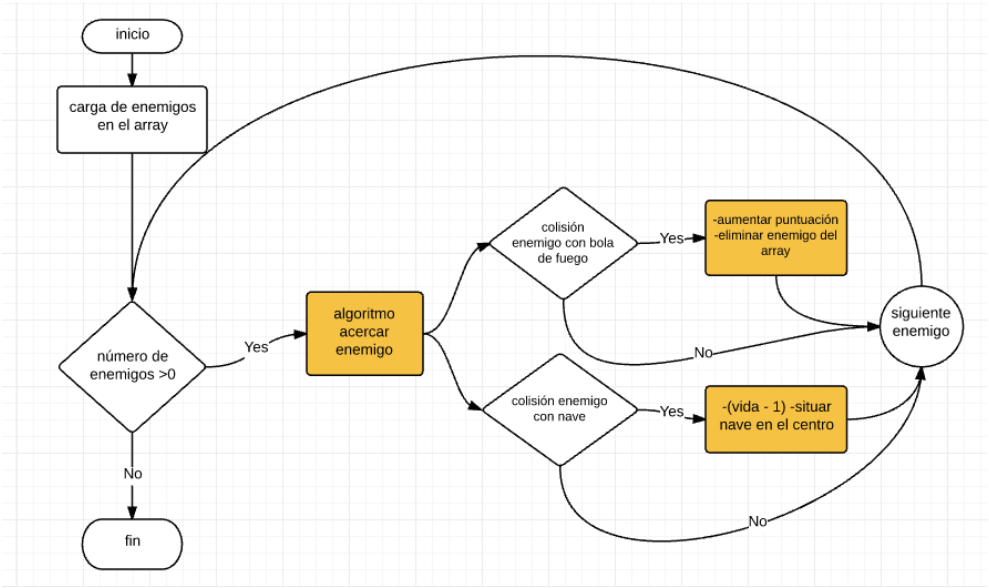


Figura 2.39 Diagrama de flujo del nivel 3

El algoritmo de acercamiento descrito en la figura 2.40 nos permite comparar la posición de cada enemigo con respecto a la nave y hacerla acercarse, dado que el juego puede ejecutarse a varias resoluciones, la cantidad de píxeles que se mueve el enemigo para acercarse puede variar con respecto a la resolución. Para el cálculo de lo que podemos llamar velocidad de la nave y enemigo se toma en cuenta la resolución de la pantalla y se calcula una constante que se multiplicará a todos los elementos móviles dentro del juego.

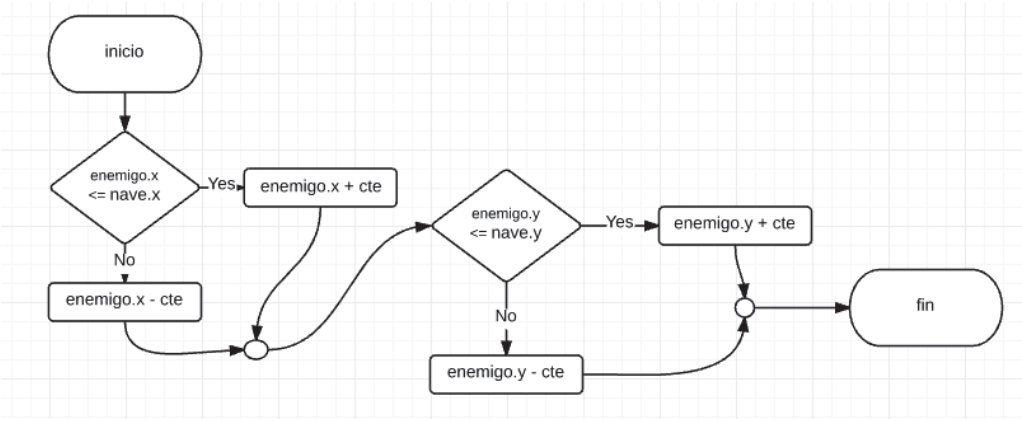


Figura 2.40 Algoritmo para acercamiento de los enemigos a la nave

A continuación se muestra en la figura 2.41 las instrucciones para el nivel 3:

Figura 2.41 Instrucciones para el nivel 3

En este nivel se observa que todos los 'invasores espaciales' son diferentes y al inicio del juego aparecen randómicamente en la pantalla, la nave debe dejar proyectiles de fuego a su paso, cuando un invasor espacial choca con el proyectil dejado por la nave este desaparece, si la nave choca con un invasor espacial los dos desaparecen, en este caso la nave pierde una vida y reaparece de nuevo en el centro de la pantalla pero tiene un tiempo corto de inmunidad a las colisiones hasta que pueda ir a un sitio despejado y atacar de nuevo a las naves enemigas, ver figura 2.42:

Figura 2.42 Escena del inicio del juego en el Nivel 3

En el siguiente código se aprecia cómo se comprueba si el proyectil choca contra alguno de los enemigos:

```
if (colisionador(Value, shoot)) then
    naveShoot = 0
    table.remove(enemies,Index)
    -- se quita de la pantalla las coordenadas del shoot para que no siga eliminando enemigos
    shoot.x = -100
    shoot.y = -100

    puntaje1 = puntaje1 + math.random(15,20)

end
```

Figura 2.43 Código de comprobación de colisión entre la nave y enemigos

Como se observa en la figura 2.43 si el enemigo (Value) tiene una colisión con la bola de fuego (shoot) se retira dicho enemigo del array (table.remove) que contiene todos los enemigos activos, al mismo tiempo se hace desaparecer la bola de fuego de la región Lua visible ya que el jugador debe volverla a ubicar una en la pantalla y continuar eliminando enemigos.

Si el jugador pierde todas sus vidas en el transcurso de cualquiera de los tres niveles aparece esta escena indicando el puntaje obtenido, ver figura 2.44:

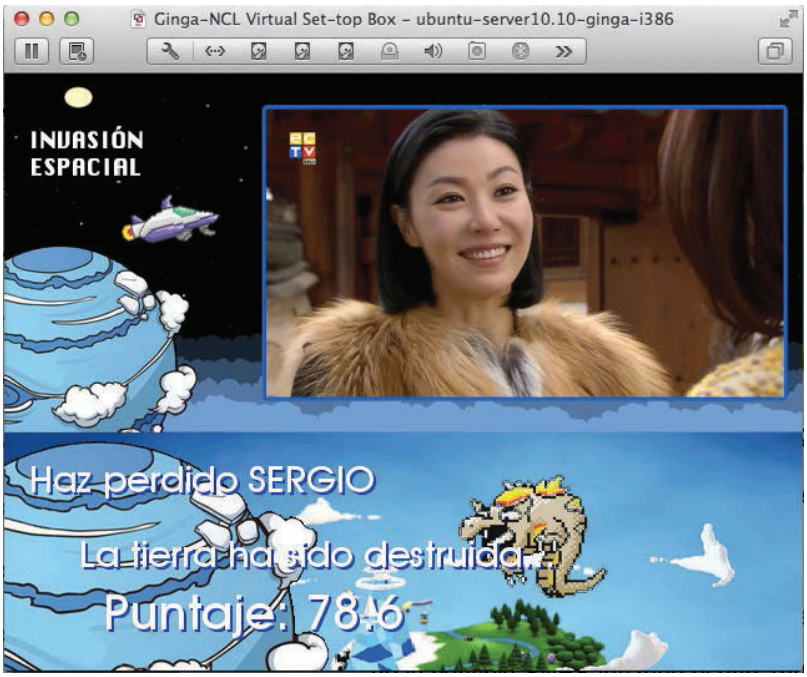


Figura 2.44 Escena final del juego causado por la pérdida de todas las vidas

2.11.4 CANAL DE RETORNO

Se ha desarrollado un servicio WEB en .NET y SQL el mismo que permite almacenar la puntuación del jugador y en ese mismo instante leer las tres puntuaciones más altas logradas en el juego.

El servicio WEB es del tipo REST es decir se envía la información en la misma URL como se muestra a continuación con las variables 'jugador' y 'score':

- http://130.206.85.9/wsGinga.asmx?userName=Sergio&score=304

Respuesta las tres mayores puntuaciones en orden descendente:

- Jugador1:345| Jugador2:325| Jugador3:315

Como se puede notar en la respuesta del servicio WEB, se obtiene los nombres y puntuaciones en texto plano sin ningún tipo de formato ya sea JSON o XML, dado que en el presente proyecto solo se contempla enviar y recibir información simbólica sin ahondar en temas de seguridad, autenticación, autenticidad, encriptación, etc. que serían muy complejos desarrollar sobre el lenguaje Lua y están fuera del alcance del presente proyecto.

Para el almacenamiento de las puntuaciones de los jugadores de ha creado una base de datos simple en SQL que almacena el nombre del usuario y su puntación, la base de datos se denomina gingaScores y contiene la tabla scores:

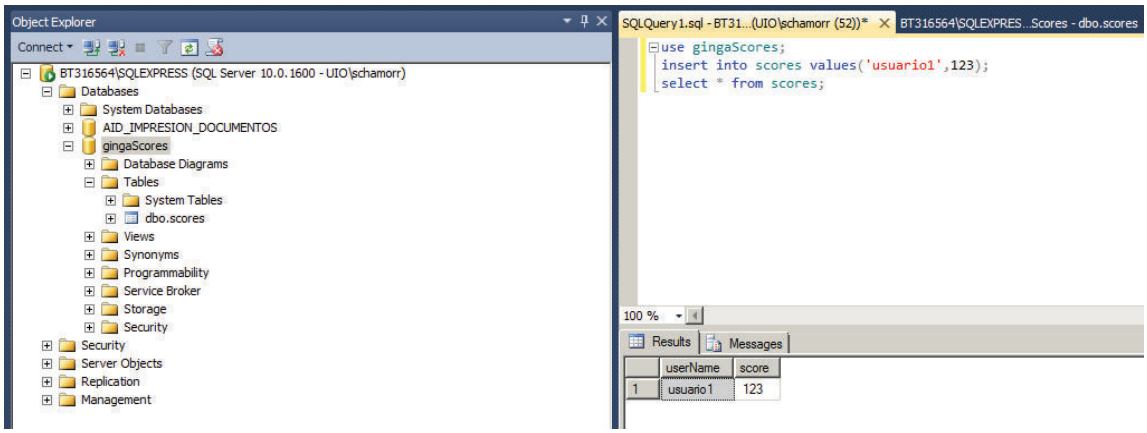


Figura 2.45 Base de datos para almacenar los puntajes de los jugadores

Para leer las puntuaciones que envía el Set-top-box se ha creado un servicio WEB tipo REST en .NET, el programa Lua envía el nombre de usuario y su puntuación mediante variables en la URL (ver figura 2.46) y el servicio web responde con las 3 mejores puntuaciones que encuentre en la base de datos, estas puntuaciones están en orden descendente:

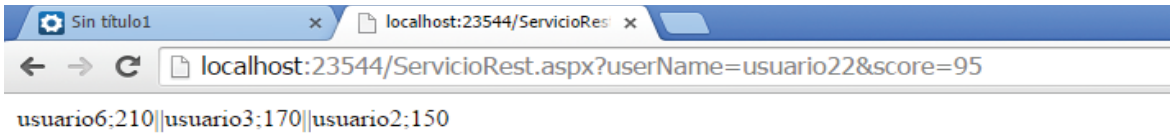


Figura 2.46 Consumo del servicio WEB REST de puntuaciones

El Servicio WEB en .Net toma las variables de la URL y almacena los datos del jugador en la Base de datos, lee las tres mejores puntuaciones y crea la respuesta del servicio REST en base al código de la figura 2.47:

```
        Try
            myConn = New SqlConnection("Data Source=BT316564\SQLEXPRESS;Initial
Catalog=gingaScores;Integrated Security=True")

            myCmd = myConn.CreateCommand
            myCmd.CommandText = "insert into scores values('" + userName + "'," + score + ")"
            myConn.Open()
            myReader = myCmd.ExecuteReader()
            myConn.Close()

            myCmd.CommandText = "select top 3 * from scores order by score desc"
            myConn.Open()
            myReader = myCmd.ExecuteReader()
            Do While myReader.Read()
                results = results + (myReader.GetValue(0)).ToString + ";"
                results = results + (myReader.GetValue(1)).ToString + "||"
            Loop

              'se elimina separadores no necesarios
              results = results.Substring(0, results.Length - 2)

        Catch ex As Exception
            'error en la conexión
            results = ex.Message
        End Try
```

Figura 2.47 Código utilizado para crear el servicio WEB REST en .NET

Esta respuesta del servicio WEB la recibe el código Lua, le da un formato de texto diferente y la muestra en pantalla.

CAPÍTULO 3

PRUEBAS DE LA APLICACIÓN INTERACTIVA

En este capítulo se realizarán las pruebas de funcionamiento de la aplicación desarrollada en el capítulo 2, esta aplicación inicialmente se la probará en un simulador (máquina virtual Ginga). Al terminar las primeras pruebas se utilizará un decodificador real que implemente el middleware Ginga NCL-Lua. Se analizarán los resultados obtenidos de las variaciones resolución.

3.1 RECEPTOR MÁQUINA VIRTUAL

Selección de la máquina virtual en resolución por defecto, ver figura 3.1.

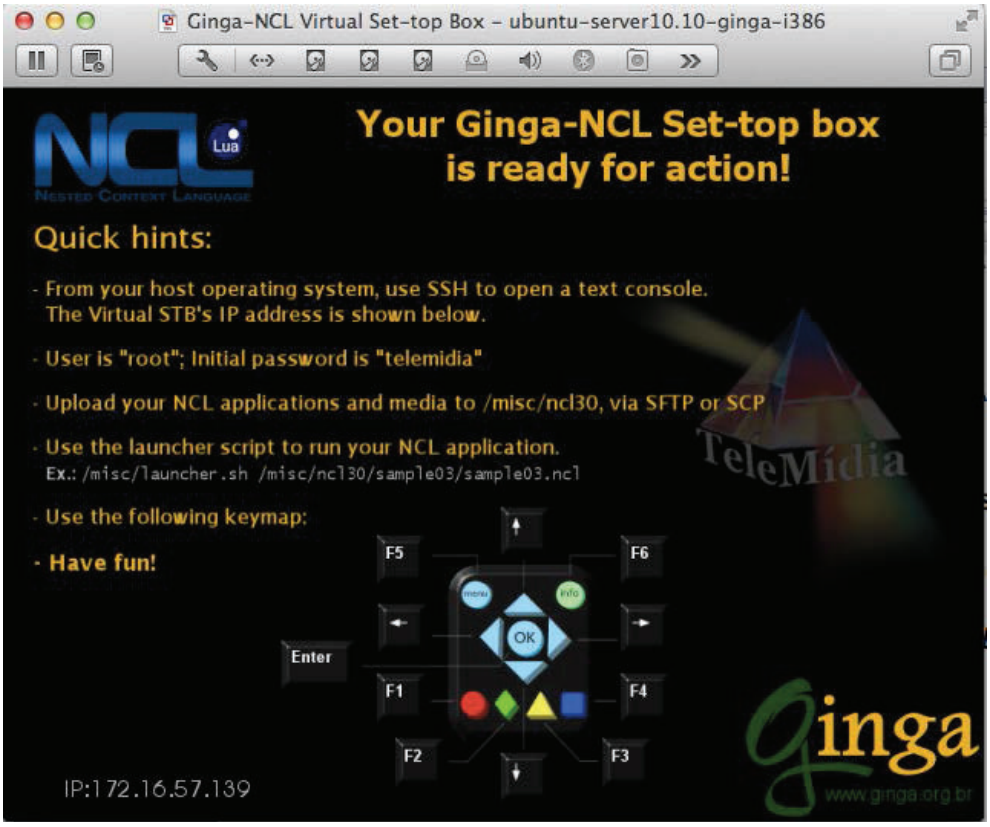


Figura 3.1 Máquina virtual Ginga con resolución por defecto

Comando ejecutado para correr la aplicación interactiva:

- /misc/launcher.sh /misc/AppNCILua/main.ncl

3.2 VISUALIZACIÓN DEL TECLADO EN PANTALLA QWERTY

Visualización del teclado en pantalla QWERTY, ver figura 3.2.

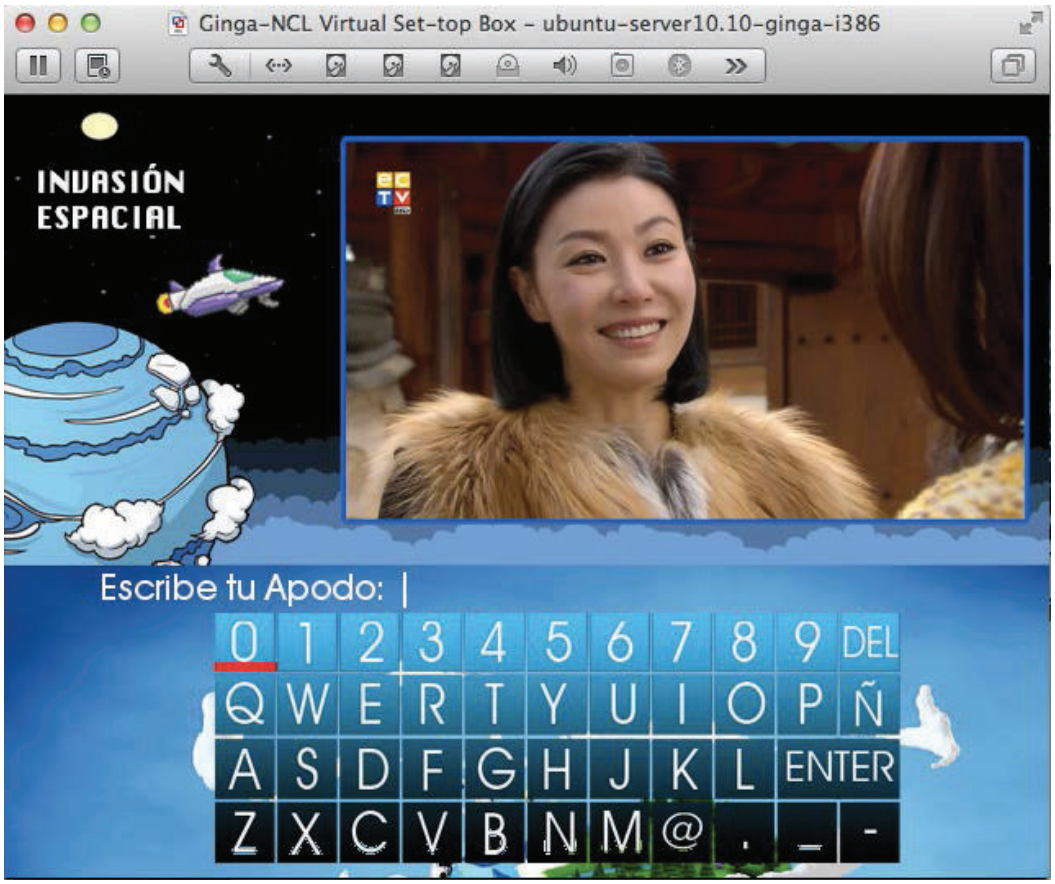


Figura 3.2 Visualización del teclado QWERTY

Como se puede observar en sobre el teclado QWERTY aparece un cursor de color rojo que se posiciona inicialmente en el número cero de la parte superior izquerda, este cursor se lo controla y posiciona mediante las flechas del contro remoto.

3.2.1 RESULTADOS INGRESO DE NOMBRE

Se ha ingresado el nombre del jugador sin ningún problema, no hubo problemas tampoco en desplazarse por la pantalla y elegir las letras o en borrar el nombre e ingresarlo de nuevo. Se comprueba que se tiene un máximo de 15 caracteres para el nombre.

Ingreso del nombre del jugador “apodo” mediante Teclado QWERTY, ver figura 3.3.

Figura 3.3 Ingreso del nombre del jugador mediante teclado QWERTY

3.3 FLUIDEZ EN EL JUEGO

En la resolución por defecto de la máquina virtual y con la capacidad de memoria que tiene la máquina virtual se procede a probar la jugabilidad. Cada nivel tiene sus propios elementos gráficos unos pueden ser más pesados que otros pero se ha diseñado el juego de manera que no se note una diferencia de rendimiento en los diferentes niveles o haya problemas al correr la aplicación en los set-top box que tienen limitados recursos de memoria y procesamiento, ver figura 3.4.

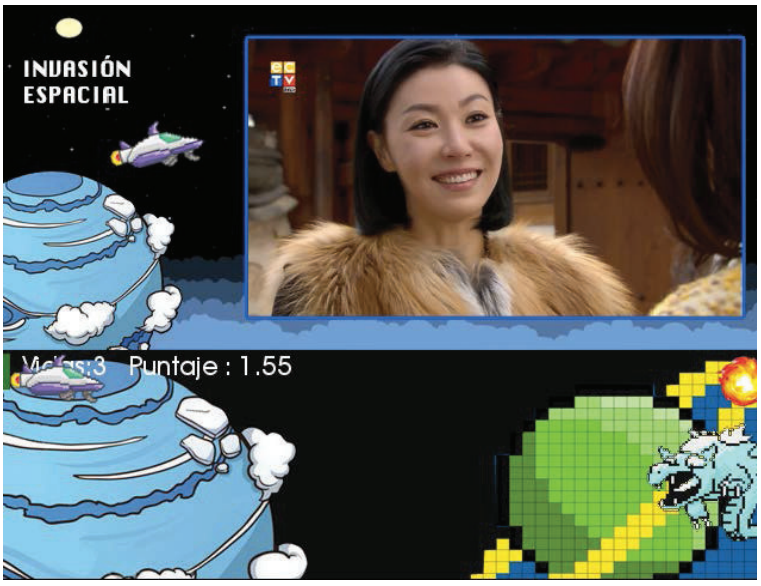


Figura 3.4 Desarrollo normal del juego

3.3.1 RESULTADOS FLUIDEZ DEL JUEGO

El juego se ejecuta con una gran fluidez, no se evidenció ningún tipo de retardo o parpadeo en los gráficos, el uso del teclado del computador permite una gran maniobrabilidad dentro del juego por lo que da la impresión de que se lo puede terminar fácilmente.

3.4 ENVÍO DE DATOS PUNTUACIÓN AL SERVIDOR

Envío de datos puntuación al servidor ver figura 63. Se procede a probar el envío de la puntuación del jugado y la recepción de las tres mejores puntuaciones registradas en la base de datos del juego, para esto se habilita una interfaz de red en la máquina de virtual la misma que tiene conectividad con el servidor donde está el servicio REST.

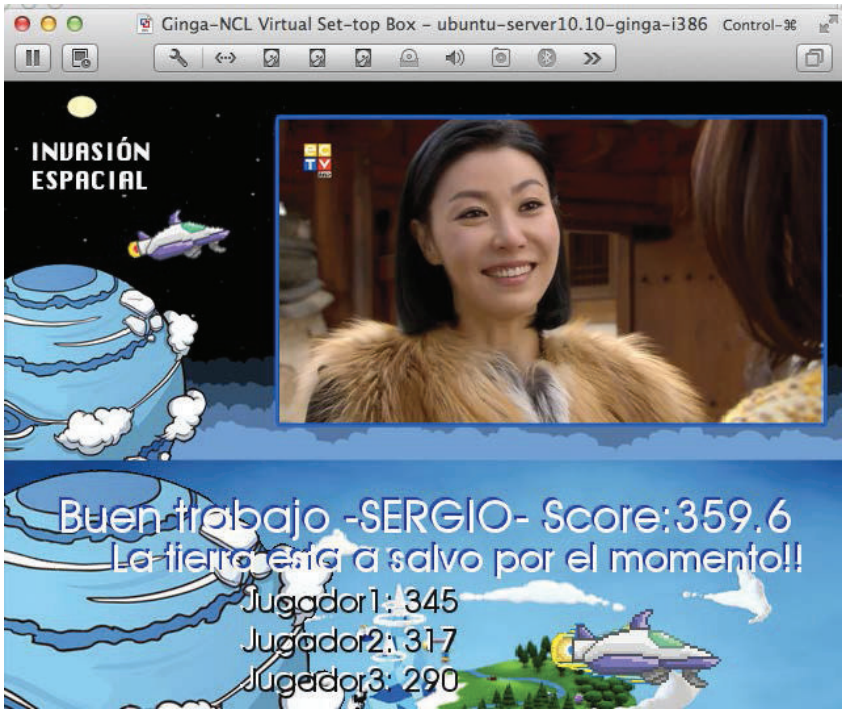


Figura 3.5 Visualización de la puntuación lograda por el jugador

3.4.1 RESULTADOS DEL ENVÍO DE DATOS PUNTUACIÓN AL SERVIDOR

No se observaron problemas al mostrar la puntuación en pantalla o enviar la puntuación al servidor. Se hizo pruebas enviando todos los caracteres del teclado

QUERTY donde se ingresa el nombre del jugador y no hubo problemas, se probó esto ya que los datos se pasan a través de una URL se almacenan en una base de datos y se los vuelve a mostrar en la pantalla.

3.5 RECEPTOR SET-TOP BOX ISDB-TB CON MIDDLEWARE GINGA

-Televisor HD 1080 x 1920 píxeles.

-Decodificador Set-top Box Proview XPS-1000.

-Antena para señal de TDT.

Para el canal de interactividad se ha utilizado la interfaz Ethernet del Set-top box y se ha cargado la aplicación mediante una memoria flash USB.

Figura 3.6 Elementos utilizados en la prueba de la aplicación desarrollada

Para realizar la prueba de esta aplicación en un escenario real se seleccionó el decodificador XPS-1000 de Proview este decodificador tiene la posibilidad de probar las aplicaciones desde su interfaz USB en tres resoluciones como son 720x480, 1280x720, 1920x1080:

Figura 3.7 Resoluciones en las que trabaja el Set-top Box Proview XPS-1000

Como podemos observar en la figura 3.7 se muestra el receptor Proview XPS-1000 en la parte del menú correspondiente a las aplicaciones Ginga que se cargan a través del puerto USB. La pantalla muestra una lista de todas las carpetas que contienen un archivo ".ncl", mediante el control remoto podemos navegar en esta lista y seleccionar la aplicación que queremos ejecutar, mediante los botones de colores del control remoto podemos elegir la resolución Rojo (720x480), Verde (1290x720) y Amarillo (1920x1080).

3.6 RECEPTOR SET-TOP BOX A RESOLUCIÓN DE 720X480

Se debe tomar en cuenta que 720x480 es la resolución para la cual se diseñó el juego, es por esto que la aplicación NCL va a presentarse similar a como se la vio en el simulador. La diferencia del simulador a un ambiente real es que se pierde un poco la visualización de los bordes como si el área de la aplicación fuera un poco más grande que la señal de video. Comparado con los resultado obtenidos al simularlo en el ambiente virtual vemos que hay un poco más de lentitud debido a que el Set-top Box real Proview XPS-1000 no es un receptor de alta gama si no que tiene unas prestaciones de memoria y rendimiento promedio.

Figura 3.8 Selección de la resolución 720x480 y visualización de una pantalla de instrucciones

La aplicación se ejecutó con relativa fluidez, se notaban algunos parpadeos en las letras mostradas por código en la pantalla, se ha tratado que las letras tengan un color diferente al del fondo para que no se pierdan y sean legibles. En el código del juego se muestran los textos al final para que se vean sobre los otros elementos gráficos. Para mitigar la ralentización del juego en resoluciones mayores, se ha implementado código Lua que permite calcular una "**constante"** para centrar las imágenes en pantalla y un "**factor"** que depende de la resolución con la que se ejecuta la aplicación, este factor se multiplicará por las velocidades de los elementos en pantalla de manera que en una resolución mayor los elementos gráficos se desplazarán una mayor cantidad de píxeles.

La variable llamada "constante" se calcula en base a dx que es lo que mide la imagen de fondo en píxeles, total es el ancho de la región Lua en píxeles y el 65 es un margen izquierdo para conservar la misma apariencia en las diferentes resoluciones. La variable llamada "factor" almacena un número que irá cambiando conforme la resolución de la región Lua, para una mayor resolución "factor" será mayor dado que dx es constante. En la figura 3.9 semuestra el código implementado para manejar cambios de resolución en la región Lua:

```
local constante = ((total/2)-dx/2)) + 65
local constante = total/(dx/2)
```

Figura 3.9 Código para mejo de cambios de resolución

Figura 3.10 Prueba de la aplicación en resolución 720x480

En esta resolución el juego se ejecutó exitosamente, se desplegaron todos los elementos visuales sin excepción del tamaño y emplazados correctamente en la pantalla tal como se esperaba. Se pudo interactuar durante toda la aplicación si fallas o bugs del programa, no se observó en ningún momento que la aplicación se detenga termine inesperadamente; se notó un poco de ralentización cuando se mostraban muchos elementos gráficos como es el caso de las pantallas de instrucciones y en el nivel 3; en general considerando que el receptor XPS-1000 es de gama media, dado que las funciones se ejecutaron correctamente se espera que funcione de mejor manera en receptores con mayor capacidad de memoria y procesamiento como es el caso del Set-top de la empresa Eitv.

En el código Lua no se implementaron funciones de re-escalamiento de imágenes por lo que para compensar esto se centraron las imágenes para que queden visibles aunque ahora no se ocupe toda la pantalla.

3.7 RECEPTOR SET-TOP BOX A RESOLUCIÓN DE 1280X720

En esta resolución se espera de antemano que toda la parte NCL funcione igual que en la resolución de 720x480. A diferencia de la prueba anterior se espera que los elementos gráficos en la parte Lua tengan un tamaño menor. Dado que las regiones que se manejan por porcentajes en NCL se mantienen con respecto al tamaño de la pantalla, esto no sucede con las imágenes mostradas en Lua por que la región Lua se maneja en píxeles.

En general para esta resolución se observó que los elementos Lua se emplazaron en el sitio que les corresponde, pero dado que ahora se tiene más espacio en la pantalla se nota visualiza todo diferente como es el caso del fondo de los niveles en el juego que ahora no ocupa toda la pantalla, las letras se ven más pequeñas.

El juego se percibe un poco más ralentizado que en la resolución anterior, para compensar esto en base a varias pruebas de percepción se ha establecido un método de cálculo de las variables “constante” y “factor” en base a la resolución en la que ejecuta el set-top box la aplicación interactiva.

Figura 3.11 Pruebas de la aplicación a resolución de 1280x720

3.8 RECEPTOR SET-TOP BOX A RESOLUCIÓN DE 1920X1080

Para esta resolución lo que se observó es que básicamente NCL funciona igual sin problemas, pero Lua en este caso exige demasiados recursos es por esto que ya la primera pantalla de ingreso del nombre se visualiza muy pequeño en la pantalla. Se puede decir que se observa las mismas partes de juego pero mas alejados entre sí y más pequeños que en las otras resoluciones.

Figura 3.12 Pruebas de la aplicación a resolución de 1920x1080

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Los juegos que se pueden desarrollar con el middleware Ginga tienen limitaciones en lo relacionado a recursos gráficos y procesamiento, no se pueden comparar a los juegos de las consolas actuales. Reproducir mini juegos antiguos con gráficos similares a los de 8 bits es totalmente posible. Aunque se puedan desarrollar juegos interesantes encontramos que el dispositivo (Control remoto del televisor) con el que el jugador va a interactuar es muy limitado, la mayoría de controles de los receptores set-top box son de muy baja calidad por lo que el desarrollador debe diseñar los juegos de manera que no haga un uso extensivo del control remoto.

Lua es un lenguaje muy atractivo para ser embebido en una gran diversidad de sistemas, dado que utiliza muy poca memoria y procesamiento para correr los scripts en Lua, razón por la cual ha sido embebido en muchos sistemas tales como dispositivos de conectividad de redes de datos, cámaras, receptores de televisión, celulares, y muchos otros más.

En lo relacionado al desarrollo de software Lua al ser un lenguaje no fuertemente tipado facilita el rápido desarrollo y aprendizaje del mismo. Durante el desarrollo se utilizaron editores de texto con plugins que permitan visualizar de mejor manera la sintaxis del lenguaje Lua (TextWrangler) pero al no disponer de un editor de código más completo y dedicado para Lua el proceso de desarrollo del juego en este lenguaje fue bastante lento. Se presentaron muchos inconvenientes a la hora de depurar y solventar errores de código, fueron muchas veces que el log de error que presentaba el terminal de la máquina virtual mostraba información insuficiente para solucionar dicho error, algo que en otros lenguajes por ejemplo mediante el IDE se tiene pistas muy claras de errores aún antes de ejecutar el código.

Lua no dispone de grandes recursos de consulta de funcionalidades, librerías, comunidades de discusión de temas puntuales (Blogs, Wikis), proyectos con código

de ejemplo y otras ayudas que permiten a los nuevos desarrolladores aprender de mejor manera este lenguaje de programación. Hay muchos otros lenguajes de script que tienen una gran cantidad de librerías desarrolladas por terceros lo que permite que se los utilicen en varios proyectos. Lua tiene pocas librerías y de la misma manera no son muy conocidas.

El desarrollo de juegos necesita habilidades especiales, dado que no necesariamente un desarrollador de software convencional tendrá la facilidad de desarrollar videojuegos. El manejo de elementos gráficos requiere de creatividad, antes de escribir el código es importante imaginar cómo se verán los elementos de la aplicación y cómo van a interactuar entre ellos, es necesario visualizar cómo se ordenará el código de manera que sea fácil de consolidar dentro del motor de juego y mantenga la homogeneidad y flexibilidad del código de todo el proyecto. El software debe tener una arquitectura flexible para integrar cambios profundos de forma fácil y ordenada, de la misma manera se debe tratar de optimizar el manejo de los recursos de memoria y procesamiento.

El motor de juego y la arquitectura que se desarrolló permitió que se pueda dividir y estructurar todo el proyecto de forma ordenada, por un lado las funciones gráficas permitían controlar y cambiar fácilmente las imágenes y el orden en que se renderizaban; las funciones de interfaz con el teclado permiten añadir o cambiar las funcionalidades de los botones utilizados en el juego sin afectar a la funcionalidad del control remoto en los otros niveles; un manejo de variables globales permitió poder terminar el juego en cualquier nivel si se terminaron las vidas y de la misma manera controlar la puntuación del jugador en cada nivel.

Al realizar las pruebas de funcionamiento en el receptor real fue necesario idear una forma de cambiar la velocidad con que se movían los elementos de la pantalla en resoluciones más altas esto fue posible implementar dado que el código estaba bien estructurado fue fácil agregar una variable global para centrar todas las imágenes en las diferentes resoluciones y un factor de multiplicación para las velocidades de los elementos móviles del juego.

4.2 RECOMENDACIONES

Para que el videojuego resulte atractivo es importante pulir la jugabilidad ya que por ejemplo si en una determinada aplicación el jugador interactúa intensamente con el control remoto no va a ser posible que el juego se desarrolle con fluidez, los controles remotos de televisores proveen una limitada capacidad de interacción en lo relacionado a videojuegos.

Al desarrollar el videojuego se buscó un equilibrio entre presentación de gráficos y el rendimiento, dados los limitados recursos por parte del receptor de televisión en algunos casos se tuvo que reescribir el código varias veces para hacerlo más simple optimizando en lo posible el consumo de procesamiento y tratando de evitar que exista mucho parpadeo de las imágenes.

A lo largo de toda la aplicación se debe tener un diseño gráfico uniforme es decir se deben utilizar los mismos colores, menús con el mismo diseño, emplazar los botones de funcionalidades en lugares de buena visibilidad, todo esto para dar una buena experiencia al jugador. El videojuego debe ser fácil de entender dado que muchos jugadores o usuarios no leerán las instrucciones pero lo relacionarán con otros juegos de manera que deducirán que deben hacer en cada nivel.

Se debe prever que generar juegos utilizando Ginga NCLua tendrá un tiempo de desarrollo más largo debido a que necesitan pulir bastante su jugabilidad, otro factor que impide su rápido desarrollo es que a menudo es difícil encontrar y solucionar los errores que se presentan en el código Lua ya que no se utilizó un IDE para realizar el debugging.

Es muy importante tener una buena arquitectura inicial del juego ya que es muy común en la práctica que se cambie la trama del juego o se hagan otros cambios de todo tipo, si no tenemos una arquitectura inicial flexible y ordenada cualquier cambio que se nos solicite en el transcurso de las pruebas o desarrollo va a requerir mucho trabajo y tiempo. Fue muy útil utilizar la metodología de desarrollo ágil Scrum ya que se pudo fragmentar el problema global y poner todo el esfuerzo en resolver

pequeñas partes del proyecto hasta poder consolidar una parte que se la puede probar y así continuar con las siguientes tareas dentro de la planificación.

Es primordial tener versiones ordenadas y calendarizadas de nuestro código ya que como no se cuenta con un editor o IDE completo para el desarrollo de este tipo de aplicaciones se puede fácilmente perder los archivos que estemos utilizando. Una fuente de error a menudo es el paso de los archivos al simulador, donde se pueden perder versiones, sobrescribir versiones nuevas con versiones antiguas.

Un aspecto muy importante que se debe cuidar es que las letras y las imágenes puedan ser legibles a una distancia normal del televisor y sobre cualquier fondo porque se pueden perder en la señal de video.

REFERENCIAS BIBLIOGRÁFICAS

[1] John Shedletsky Blog. *Definiciones de bytecode*. [Online]. Disponible en: http://blog.roblox.com/2012/08/bye-bye-bytecode/

[2] Roberto Ierusalimschy, Waldemar Celes y Luiz Henrique de Figueiredo. *Definiciones de Lua* [Online]. Disponible en: http://www.lua.org/about.html

[3] Roberto Ierusalimschy. *Lua Performance Tips*. [Online]. Disponible en: http://www.lua.org/gems/sample.pdf

[4] Lua.org, PUC-Rio. *Licencia Lua*. [Online]. Disponible en: http://www.lua.org/license.html

[5] Grupo Ginga Goias. *Ejemplos de integración de Lua y NCL*. [Online]. Disponible en: http://grupogingagoias.com.br/blog/wpcontent/uploads/downloads/2010/12/Integracao-NCL-LUA.pdf

[6] Eitv Brasil. *Especificaciones del Middleware Ginga* [Online]. Disponible en: http://www.eitv.com.br/middleware_es.php

[7] Roberto Ierusalimschy, Waldemar Celes y Luiz Henrique de Figueiredo. Lua Team, *The Evolution of Lua* [Online]. Disponible en: http://www.lua.org/doc/hopl.pdf

[8] PongGame.org. *Historia y evolución del juego Pong*. [Online]. Disponible en: http://www.ponggame.org/

[9] The International Arcade Museum. *Museo de Juegos antiguos y consolas*. [Online]. Disponible en: http://www.arcade-museum.com/game_detail.php?game_id=10167

[10] The International Arcade Museum. *Circus Charlie*. [Online]. Disponible en: http://www.arcade-museum.com/game_detail.php?letter=&game_id=7341

[11] IBM. *Embed Lua for scriptable apps*. [Online]. Disponible en: http://www.ibm.com/developerworks/library/l-embed-lua/

[12] Luiz Fernando Gomes Soares, Rogério Ferreira Rodrigues, Márcio Ferreira Moreno. *Ginga-NCL: the declarative environment of the Brazilian digital TV system.* [Online]. Disponible en: http://www.scielo.br/scielo.php?pid=S0104-65002007000100005&script=sci_arttext

[13] ABNT. *Normas TV Digital ISDB-Tb*. Pág. 57 [Online]. Disponible en: http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Vc_2008.pdf

[14] Plugin 3WDL. *Ejemplo de integración con Lua*. [Online]. Disponible en: https://sourceforge.net/projects/dmlw/

[15] Adobe. *Integración de Lightroom con Lua*. [Online]. Disponible en:https://www.adobe.com/content/dam/Adobe/en/devnet/photoshoplightroom/pdfs/lr4/lightroom-sdk-guide.pdf.

[16] OnlyJob Blog*. Comparación Benchmarking de varios lenguajes incluyendo Lua*. [Online]. Disponible en: http://onlyjob.blogspot.com/2011/03/perl5-python-ruby-php-c-c-lua-tcl.html

[17] Dzone. *What Happened To The 9 Programming Languages To Watch in 2011*. [Online]. Disponible en:http://java.dzone.com/articles/what-happened-9-programming

[18] Codeeval. *Most Popular Coding Languages of 2015*. [Online]. Disponible en: http://blog.codeeval.com/codeevalblog/2015?utm_content=buffer27664&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer#.VOImiC44Q5w=

[19] IBM. *Metodología de desarrollo SCRUM*. [Online]. Disponible en:https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wik

i/Rational+Team+Concert+for+Scrum+Projects/page/SCRUM+como+metodolog%C3%ADa

[20] Game Fabrique. *Captura de Imagen del juego Total Carnage* [Online]. Disponible en: http://www.gamefabrique.com/storage/screenshots/snes/total-carnage-05.png

[21] Universidad la Laguna. *La lista de referencias (Normas IEEE)*. [Online]. España, 2016 Disponible en: http://www.ull.es/view/institucional/bbtk/Referencias_Normas_IEEE/es

ANEXOS

ANEXO 1 CÓDIGO DE LA APLICACIÓN LUA

```
--*********************************************************************
--*********************************************************************
--*********************************************************************
--*********************************************************************
--******* PROYECTO DE TITULACION              ********************
--******* Desarrollado por Sergio Chamorro     ********************
--*********************************************************************
--*********************************************************************

-- Juego desarrollado en lenguaje Lua del Middleware Ginga
-- Canal de Interactividad Television Digital
-- Estandar ISDTB-Tb el mismo que esta vigente en el Ecuador
-- ESCUELA POLITECNICA NACIONAL DEL ECUADOR
-- QUITO 2014 - 2015
--*********************************************************************




require "tcp"
dofile("xml.lua")
dofile("handler.lua")
dofile("Entities2AccentedChars.lua")


function writeText(text)
  canvas:attrColor(255,255,255,0)
   canvas:clear();

   local cw, ch = canvas:attrSize()

   canvas:attrFont("vera", 8)
   local tw, th = canvas:measureText("A")
   canvas:drawText(5, ch-th, text)
   canvas:flush()
end

function escape (s)
  s = string.gsub(s, "([&=+%c])", function (c)
       return string.format("%%%02X", string.byte(c))
     end)
  s = string.gsub(s, " ", "+")
  return s
end




local totalx, totaly = canvas:attrSize()
local img = canvas:new('ooo.png')
local dx, dy = img:attrSize()
```

```
local constante = ((totalx/2)-(dx/2))+65
local factor = totalx/(dx/2)
local ooo = { img=img, x=(totalx/2)-(dx/2) , y=0, dx=dx, dy=dy }


--Textos y mensajes del juego.
local estado = 'intro' -- estado inicial del juego
local sigEstado = 'game0' -- nivel inicial
local l1 = 'Bienvenido jugador introduce tu Nombre...' -- texto de bienvenida
local name = ''
local mgame1

-- variables del GLOBALES
local hinst
local newh = true
local hup = true
local hmed
local hdown
local img = canvas:new('heart.png')
local heart = { img=img, x=totalx , y=5, dx=dx, dy=dy }
local vidas = 3
local puntaje1 = 0
local countg1 = 0

--variables de graficos waiting (graficos de animaciones)
local xr = 0
local yr = 0
local rm = 0


--variables canal retorno
local canalRetorno = 0
local topScores = "Obteniendo Datos ..."

-- Fin exitoso del juego
local tierraFin = canvas:new('earth.png')
local tierraFin = { img=tierraFin, x=0 , y=0, dx=130, dy=108 }
local naveFin = canvas:new('navet.png')
local naveFin = { img=naveFin, x= totalx * 0.65 , y= totaly * 0.55, dx=130, dy=108 }
function redrawfin1 ()
canvas:attrColor('black')
        canvas:drawRect('fill', 0,0, canvas:attrSize())
        canvas:compose(ooo.x, ooo.y - 20, ooo.img)
        canvas:compose(tierraFin.x, tierraFin.y , tierraFin.img)
        canvas:compose(naveFin.x, naveFin.y , naveFin.img)

        canvas:attrColor('blue')
        canvas:attrFont('vera', 35)
        canvas:drawText(40 + constante,20, 'Buen trabajo -'.. name .. '- Score:'..puntaje1  )
        canvas:attrFont('vera', 30)
        canvas:drawText(80 + constante,55, 'La tierra esta a salvo por el momento!!' )

        -- se repiten todos los textos para que con el fondo azul para su facil lectura
        canvas:attrColor('white')
```

```
        canvas:attrFont('vera', 35)
        canvas:drawText(42 + constante,22, 'Buen trabajo -'.. name .. '- Score:'..puntaje1  )
        canvas:attrFont('vera', 30)
        canvas:drawText(82 + constante,57, 'La tierra esta a salvo por el momento!!' )

        getScore()

        -- se libera la memoria
        canvas:flush()
end

function getScore ()

   if (canalRetorno==0)then
                    puntaje1 = roundMyScore(puntaje1,0)
                    canalScores()
                    canalRetorno = 1
        end

        -- funcion overwrite en archivo lua separado
        canvas:attrColor('white')
        canvas:attrFont('vera', 20)
        canvas:drawText(80 + constante,127,"Mejores puntuaciones: ".. topScores )

        canvas:attrColor('black')
        canvas:attrFont('vera', 20)
        canvas:drawText(80 + constante,125,"Mejores Puntuaciones: ".. topScores )

end




--*****************************************************************
--*****************************************************************
--*************** CODIGO TECLADO      ****************************
--*************** REUTILIZABLE        ****************************
--*****************************************************************
-- Desarrollado por: Sergio Chamorro    ***************************
-- No se necesitan librerias extra      ***************************
-- imagenes utilizadas  'keyboard.png'  ***************************
-- uso ver la seccion correspondiente en eventos del control remoto*
--*****************************************************************
--*****************************************************************
--*****************************************************************
-- Codigo eventos de teclado reutilizable    ***********************
--*****************************************************************
--*****************************************************************
--** para este ejemplo se almacena en la variable 'name'   *********
--*********************************************************************

local img = canvas:new('keyboard.png')
local dx, dy = img:attrSize()
```

```
local keyn = { img=img, x= 130 , y=50, dx=dx, dy=dy }
local img = canvas:new('pointer.png')
local dx, dy = img:attrSize()
local pointer = { img=img, x=130 , y=60, dx=dx, dy=dy }
local ix, jx = 0
-- se crea un array multidimensional para almacenar cada caracter
local keycodes={}
local linekey={'0','1','2','3','4','5','6','7','8','9','|'}
   for jx=0,10 do
         keycodes[jx] = {}
      keycodes[jx][ix] = linekey[jx + 1]
   end
   ix = ix + 1
linekey={'Q','W','E','R','T','Y','U','I','O','P','Ñ'}
   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
         ix = ix + 1
linekey={'A','S','D','F','G','H','J','K','L','!','!'}
   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
         ix = ix + 1
linekey={'Z','X','C','V','B','N','M','@','.','_','-'}
   for jx=0,10 do
      keycodes[jx][ix] = linekey[jx + 1]
   end
         ix = ix + 1

local pk1 = 0
local pk2 = 0

         -- se grafica el teclado en pantalla
function redrawName ()
         canvas:attrColor('black')
         canvas:drawRect('fill', 0,0, canvas:attrSize())
         canvas:compose(ooo.x, ooo.y - 20, ooo.img)
         canvas:attrColor('white')
         canvas:attrFont('vera', 22)
         canvas:drawText(60 + constante,0, 'Escribe tu Apodo: '..name..'|')
         canvas:compose(keyn.x + constante, keyn.y - 20, keyn.img)
         canvas:compose(pointer.x + constante, pointer.y, pointer.img)
         canvas:flush()
end

--*****************************************************************
--*****************************************************************
--****** Fin codigo teclado reutilizable **************************
--*****************************************************************
--*****************************************************************
```

```
-- variables necesarias para RUNNER NIVEL 2

local ra = canvas:new('navet.png')
local ra2 = canvas:new('navetl.png')
local rb = canvas:new('naveup.png')
local rc = canvas:new('navedown.png')
local runbg = canvas:new('run-bg.png')
local rock = canvas:new('rock.png')
local rockra = canvas:new('rocka.png')
local rockrb = canvas:new('rockb.png')
run = { img=ra, x=50 , y=110, dx=dx, dy=dy }
runup = { img=rb, x=50 , y=110, dx=dx, dy=dy }
shipDown = { img=rc, x=50 , y=110, dx=dx, dy=dy }
shipNormal = { img=ra2,x=50, y=110, dx=dx, dy=dy }
runback = {img=runbg, x=0 , y=0, dx=dx, dy=dy}
rock1 = {img=rock, x=600, y=120, dx=dx, dy=dy}
rock2 = {img=rock, x=670, y=120, dx=dx, dy=dy}
rock3 = {img=rock, x=750, y=120, dx=dx, dy=dy}
local runupvar = 0
local normalPower = 0
local complete1 = 0
local energiaRunner = 0

-- comprobar colisiones
function collidel (B, A)

        local ax1, ay1 = A.x, A.y
        local ax2, ay2 = ax1+20, ay1+10
        local bx1, by1 = B.x, B.y
        local bx2, by2 = bx1+20, by1+10

        if ax2 < bx1  or bx2 < ax1 then
                return false
        elseif ay2 < by1 or by2 < ay1 then
                return false
        end

        --cambios de imagenes
        return true
end

-- se grafica el nivel 2
function runner ()

        -- fondo
        canvas:attrColor('black')
        canvas:drawRect('fill', 0,0, canvas:attrSize())
        canvas:compose(runback.x, runback.y, runback.img)

        -- en este caso los tanques se vuelven un impedimento para que la nave cargue energia
        canvas:compose(rock1.x, rock1.y, rock1.img)
        canvas:compose(rock2.x, rock2.y, rock2.img)
        canvas:compose(rock3.x, rock3.y, rock3.img)
        -- velocidad de los tanques
```

```
rock1.x = rock1.x - 1.5
rock2.x = rock2.x - 1.5
rock3.x = rock3.x - 1.5

if (rock1.x < -50) then
		rock1.x = totalx
		local nn = math.random(3)
		if(nn==1)then
			rock1.img = rock
			rock1.y = 120
		end
		if(nn==2)then
			rock1.img = rockra
			rock1.y = 0
		end
		if(nn==3)then
			rock1.img = rockrb
			rock1.y = 120
		end
end

if (rock2.x < -50) then
		rock2.x = totalx
		local nn = math.random(3)
		if(nn==1)then
			rock2.img = rock
			rock2.y = 120
		end
		if(nn==2)then
			rock2.img = rockra
			rock2.y = 0
		end
		if(nn==3)then
			rock2.img = rockrb
			rock2.y = 120
		end
end

if (rock3.x < -50) then
		rock3.x = totalx
end


-- Comprobar choque con tanques
if collidel(rock1 , run) then
		vidas = vidas - 1
		rock1.x = totalx
		local nn = math.random(3)
		if(nn==1)then
			rock1.img = rock
			rock1.y = 120
		end
		if(nn==2)then
			rock1.img = rockra
```

```
                    rock1.y = 0
                end
                if(nn==3)then
                    rock1.img = rockrb
                    rock1.y = 120
                end
        end
        if collidel(rock2 , run) then
                vidas = vidas - 1
                rock2.x = totalx
                local nn = math.random(3)
                if(nn==1)then
                    rock2.img = rock
                    rock2.y = 120
                end
                if(nn==2)then
                    rock2.img = rockra
                    rock2.y = 0
                end
                if(nn==3)then
                    rock2.img = rockrb
                    rock2.y = 120
                end
        end
        if collidel(rock3 , run) then
                vidas = vidas - 1
                rock3.x = totalx
        end

        if(runupvar == 1 and complete1==0) then
                canvas:compose( run.x , run.y  , run.img)


                        -- posici—n mas alta
                        if(run.y > 5) then
                                run.y = run.y - 1
                                        if(run.y<=5 and complete1==0) then
                                                runupvar = 0

                                    end
                        else
                                if(run.y<=10) then
                                        runupvar = 0
                                end
                        end

        elseif(complete1 == 1) then
                        --La nave despega NIVEL 2 COMPLETO
                                        run.y = run.y - 1.1
                                        run.x = run.x + 5
                                        canvas:compose( run.x , run.y  , run.img)
                                                if (run.y < -50) then

                        -- al terminar este nivel se dibuja la pantalla de informacion:
```

```
                                        estado = "waiting"
                -- luego de esto se grafica el NIVEL 3
                sigEstado = "finalLevel"
                end


        elseif(runupvar == 0 and complete1==0)then



                -- posici—n m‡s baja
                if(run.y < 110) then
                        run.y = run.y +1
                        canvas:compose( run.x , run.y  , shipDown.img)
                else
                        -- Animaci—n de la nave en tierra
                        if(normalPower<16) then
                                canvas:compose( run.x , run.y  , run.img)
                                normalPower = normalPower + 1
                        else
                                canvas:compose( run.x , run.y  , shipNormal.img)
                                normalPower = normalPower + 1
                                if(normalPower==32)then
                                        normalPower = 0
                                        energiaRunner = energiaRunner + 4
                                        puntaje1 = puntaje1 + 4

                                end


                        end
                end
        end


        if vidas <= 0 then
                estado="endg1mal"
        end

        if energiaRunner >= 100 then
                complete1 = 1
                --vidas = vidas + 1
        end

        -- texto en la pantalla del NIVEL 2
        canvas:attrColor('white')
        canvas:attrFont('vera', 22)
        canvas:drawText(140,9, 'ENERGIA'..'              Vidas ' .. vidas )
        canvas:attrFont('vera', 50)
        canvas:drawText(228,0, energiaRunner .. '%')

        canvas:flush()
end
```

```
--variables necesarias para Pong
local img = canvas:new('ball.png')
local dx, dy = img:attrSize()
local balls = { img=img, x=totalx/2 , y=totaly/2, dx=dx, dy=dy }
local bsx = 0  --sumando del eje x
local bsy = 0  --sumando del eje y
local vsx = 1*factor    -- velocidad constante del eje X
local vsy = 1*factor   -- velocidad constante del eje Y
local psum = 0
local vpaleta = 0.8
local vsp = vpaleta*factor   -- velocidad de la paleta

local imgp = canvas:new('heart.png')
local heartp = { img=imgp, x=43 , y=0, dx=dx, dy=dy }
local imgp = canvas:new('planet.png')
local dxp, dyp = imgp:attrSize()
local planet = { img=imgp, x=totalx-dxp+10 , y=0, dx=dxp, dy=dyp }
local imgp = canvas:new('monster.png')
local monsterp = { img=imgp, x=totalx-100 , y=0, dx=130, dy=108 }
local imgp2 = canvas:new('monster2.png')
local monsterp2 = { img=imgp2, x=totalx-100 , y=0, dx=130, dy=108 }
local imgp1 = canvas:new('monster1.png')
local monsterp1 = { img=imgp1, x=totalx-100 , y=0, dx=130, dy=108 }

local tierra = canvas:new('earth.png')
local tierra = { img=tierra, x=0 , y=0, dx=130, dy=108 }

local mini = canvas:new('mini.png')
local mini = { img=mini, x=0 , y=0, dx=130, dy=108 }

local monsterUp = 0
local counterMonster = 0

local colisionesMonster = 0


function pongGame ()


        if (balls.x + bsx < totalx and balls.x + bsx > 0  ) then
                bsx = bsx + vsx
        else
                vsx = - vsx
                bsx = bsx + vsx
                if(vsx > 0)then  -- si la pelota choca contra la pantalla izquierda
                        vidas = vidas - 1
                        if vidas < 1 then

estado = 'endg1mal' -- si se terminan las vidas finaliza el juego
                        else
                        end
```

```
                else
                        -- si el proyectil rebota contra la pared derecha
                        colisionesMonster = colisionesMonster + 1
                        if (colisionesMonster > 3 ) then
                                estado = 'waiting'
                        end
                        --puntaje1 = puntaje1 + 5
                end
        end

        if (balls.y + bsy < totaly and balls.y + bsy > 0  ) then
        bsy = bsy + vsy
        else
        vsy = - vsy
        bsy = bsy + vsy
        end

        if (totaly/2 + psum < totaly - 30 and totaly/2 + psum > 0) then
        psum = psum + vsp
        else
        vsp = - vsp
        psum = psum + vsp
        end

        if((balls.x+bsx)<10 and ((balls.y+bsy)>(totaly/2 + psum) - 17 and(balls.y+bsy)<(totaly/2 +
psum + 30 - 5 ))) then
                -- si la pelota choca en la raqueta verde
                vsx = - vsx
                bsx = bsx + vsx
                -- puntaje1 = puntaje1 + 5
        else
        end


        if(monsterp.y < totaly - 107 and monsterUp == 0) then
                monsterp.y = monsterp.y + 0.2
                if (monsterp.y >= totaly - 107) then
                        monsterUp = 1
                end
        end

        if(monsterp.y > 0 and monsterUp == 1) then
                monsterp.y = monsterp.y - 0.2
                if (monsterp.y <= 0) then
                        monsterUp = 0
                end
        end




        canvas:attrColor('black')
        canvas:drawRect('fill', 0,0, canvas:attrSize())
```

```
	canvas:compose( tierra.x , tierra.y  , tierra.img)
	-- canvas:compose( heartp.x , heartp.y  , heartp.img)
	canvas:compose( planet.x , planet.y  , planet.img)


	canvas:compose( balls.x + bsx, balls.y + bsy , balls.img)


	if (balls.x + bsx > totalx - 130) then
		if (balls.y + bsy > monsterp.y  and balls.y + bsy < monsterp.y + 108  ) then
			canvas:compose( monsterp.x , monsterp.y  , monsterp2.img)
			puntaje1 = puntaje1 + 0.05
			puntaje1 = round(puntaje1,2)
		else
			canvas:compose( monsterp.x , monsterp.y  , monsterp.img)
		end
	else
		if (counterMonster < 60) then
			canvas:compose( monsterp.x , monsterp.y  , monsterp.img)
			counterMonster = counterMonster + 1
		else
			counterMonster = counterMonster + 1
			canvas:compose( monsterp.x , monsterp.y  , monsterp1.img)
			if (counterMonster == 120) then
				counterMonster = 0
			end

		end
	end


	canvas:attrColor('white')
	canvas:attrFont('vera', 22)
	canvas:drawText(20,0, 'Vidas:'..vidas)
	canvas:drawText(80,0, '     Puntaje : '..puntaje1 .. ' '  )
	canvas:attrColor('green')
	canvas:drawRect('fill', 0,totaly/2 + psum, 10,30)

	canvas:compose( 10 , totaly/2 + psum , mini.img)

	canvas:flush()

end

function round(num, idp)
  local mult = 10^(idp or 0)
  return math.floor(num * mult + 0.5) / mult
end




function waitingCanvas ()
```

```
	canvas:attrColor('black')
	canvas:drawRect('fill', 0,0, canvas:attrSize())
	canvas:compose(ooo.x, ooo.y - 20, ooo.img)

	canvas:attrColor('white')

	if sigEstado == 'game0' then
		canvas:attrFont('vera', 28)
		canvas:drawText(15 + constante,20, '-'..name.. '- Utiliza las flechas de Arriba y
Abajo')
		canvas:drawText(15 + constante,45, 'Usa la barra verde para quemar al
invasor...')
	end

	if sigEstado == 'runner' then
		canvas:attrFont('vera', 28)
		canvas:drawText(15 + constante,20, '-'..name.. '- Utiliza la flecha de arriba')
		canvas:drawText(15 + constante,45, 'La nave ha perdido energia
REABASTECER...')
	end

	if sigEstado == 'finalLevel' then
		canvas:attrFont('vera', 28)
		canvas:drawText(15 + constante,20, '-'..name.. '- Utiliza las flechas y enter')
		canvas:drawText(15 + constante,45, 'Huye de los enemigos y prepara
trampas...')
	end

	canvas:attrColor('blue')
	canvas:attrFont('vera', 40)
	canvas:drawText(100 + constante,80, 'Puntaje: '..puntaje1)

	canvas:attrColor('white')
	canvas:attrFont('vera', 40)
	canvas:drawText(102 + constante,82, 'Puntaje: '..puntaje1)


	canvas:attrFont('vera', 25)
	canvas:attrColor('red')
	canvas:drawText(210 + constante,150, ' Para continuar Presiona [ENTER]')
	canvas:attrColor('white')
	canvas:drawText(212 + constante,152, ' Para continuar Presiona [ENTER]')

	xr = xr + 2
	if xr > totalx then
	xr = 0
	end

	if (yr < math.floor(totaly/2) + 25 and rm == 0 ) then
	yr = yr + 1
	elseif ( yr == math.floor(totaly/2) + 25 and rm == 0) then
	rm = 1
	end
```

```
	if (yr > 0 and rm == 1 ) then
	yr = yr - 1
	elseif ( yr == 0 and rm == 1) then
	rm = 0
	end
	canvas:attrColor('green')
	canvas:drawRect('fill', xr, 0, 20, yr)
	canvas:attrColor('yellow')
	canvas:drawRect('fill', xr, yr, 20, totaly)
	canvas:flush()
end




function redrawEndLose ()

	if (canalRetorno==0)then
		puntaje1 = roundMyScore(puntaje1,0)
		canalScores()
		canalRetorno = 1
	end

	canvas:attrColor('black')
	canvas:drawRect('fill', 0,0, canvas:attrSize())

	canvas:compose(ooo.x, ooo.y - 20, ooo.img)
	canvas:compose(tierra.x, tierra.y, tierra.img)
	canvas:compose(totalx / 2 + 40 , 30, monsterp2.img)

	canvas:attrColor('blue')
	canvas:attrFont('vera', 30)
	canvas:drawText(20 + constante,20,'Haz perdido ' .. name)
	canvas:drawText(52 + constante,52,'La tierra ha sido destruida... ')
	canvas:attrFont('vera', 40)
	canvas:drawText(82 + constante,82,'Puntaje: ' .. puntaje1)
	canvas:attrFont('vera', 20)
	canvas:drawText(87 + constante,127,'Mejores puntuaciones: ' .. topScores)

	canvas:attrColor('white')
	canvas:attrFont('vera', 30)
	canvas:drawText(18 + constante,18,'Haz perdido ' .. name)
	canvas:drawText(50 + constante,50,'La tierra ha sido destruida... ')
	canvas:attrFont('vera', 40)
	canvas:drawText(80 + constante,80,'Puntaje: ' .. puntaje1)
	canvas:attrFont('vera', 20)
	canvas:drawText(85 + constante,125,'Mejores puntuaciones: ' .. topScores)

	canvas:flush()

end
```

```
-- function preparar envio score
function roundMyScore(num, idp)
  num = num + vidas*9
  local mult = 10^(idp or 0)
  return math.floor(num * mult + 0.5) / mult
end

-- function llamada canal retorno
function canalScores()

          local host = "172.31.9.223"

    local path = "/ws.aspx?userName=".. name .. "&score="..puntaje1
    -- local path = scape("/servicio1.html?userName".. name .. "&score="..puntaje1)

    tcp.execute(
        function ()

            tcp.connect(host, 7080)

            local url = "http://"..host..path
            local request = "GET "..url.." HTTP/1.0\n"

            request = request .. "Host: "..host.."\n\n"

            tcp.send(request)

            local result = tcp.receive("*a")

            -- este numero recorta el string para obtener los datos puede variar dependiendo del
servidor donde se aloje el servicio rest
            topScores = string.sub(result,313)
            --topScores = string.sub(result,299)
            topScores = string.gsub(topScores,"||", "  ")
            topScores = string.gsub(topScores,";", ":")


            if result then
                        if evt.error ~= nil then
                                    result = 'error: ' .. evt.error
                              end
                  end

            tcp.disconnect()

        end
    )

end




-- Comprueba si dos areas se superponen COLISION
```

```
function collide (A, B)
	local ax1, ay1 = A.x, A.y
	local ax2, ay2 = ax1+10, ay1+5
	local bx1, by1 = B.x, B.y
	local bx2, by2 = bx1+5, by1+5

	if ax1 > bx2 then
		return false
	elseif bx1 > ax2 then
		return false
	elseif ay1 > by2 then
		return false
	elseif by1 > ay2 then
		return false
	end
	--cambios de imágenes
	return true
end




-- final level
-- final level variables

local m1 = canvas:new('m1.png')
local m1 = { img=m1, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m2 = canvas:new('m2.png')
local m2 = { img=m2, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m3 = canvas:new('m3.png')
local m3 = { img=m3, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m4 = canvas:new('m4.png')
local m4 = { img=m4, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m5 = canvas:new('m5.png')
local m5 = { img=m5, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m6 = canvas:new('m6.png')
local m6 = { img=m6, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m7 = canvas:new('m7.png')
local m7 = { img=m7, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m8 = canvas:new('m8.png')
local m8 = { img=m8, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m9 = canvas:new('m9.png')
local m9 = { img=m9, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m10 = canvas:new('m10.png')
local m10 = { img=m10, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m11 = canvas:new('m11.png')
local m11 = { img=m11, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }
local m12 = canvas:new('m12.png')
local m12 = { img=m12, x=math.random(totalx) , y=math.random(totaly), dx=58, dy=39 }

local nave = canvas:new('mini2.png')
```

```
local nave = { img=nave, x= totalx/2 , y=totaly/2, dx=49, dy=28 }

local nave2 = canvas:new('mini2u.png')
local nave2 = { img=nave2, x= totalx/2 , y=totaly/2, dx=49, dy=28 }

local shoot = canvas:new('ball.png')
local shoot = { img=shoot, x= -100 , y= -100, dx=42, dy=47 }

local enemies = {m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12}
local velEnemies = 0.1*factor
local plusX = 0
local plusY = 0

local naveShoot = 0
local coordsShoot = {-100 , -100}

-- contador tiempo de inmunidad
local inmunidadTiempo = 0

-- tiempo de inmunidad
local inmunidad = 1500
local vidasExtraNivel3 = 0

function finalLevel ()

	if vidasExtraNivel3 == 0 then
		vidas = vidas + 4
		vidasExtraNivel3 = 1
	end

	nave.x =  nave.x + plusX
	nave.y =  nave.y + plusY

	if nave.x < 0 then
		plusX = 0
	end
	if nave.x > totalx - 49 then
		plusX = 0
	end

	if nave.y < 0 then
		plusY = 0
	end
	if nave.y > totaly - 28 then
		plusY = 0
	end

	canvas:attrColor('black')
	canvas:drawRect('fill', 0,0, canvas:attrSize())
	canvas:compose(ooo.x, ooo.y - 20, ooo.img)


	-- disparo de la nave
	if naveShoot == 1 then
```

```
                canvas:compose(coordsShoot[1], coordsShoot[2], shoot.img)

        end



        -- for each para cada elemento a graficar
                -- variable local para contar si aun quedan enemigos
        local counterEnemies = 0
        for Index, Value in pairs( enemies ) do
                counterEnemies = counterEnemies + 1
                inmunidadTiempo = inmunidadTiempo + 1

                if inmunidadTiempo > inmunidad + 3 then
                        inmunidadTiempo = inmunidad + 2
                end

                if (colisionador(Value, shoot)) then
                        naveShoot = 0
                        table.remove(enemies,Index)
                        -- se quita de la pantalla las coordenadas del shoot para que no
siga eliminando enemigos
                        shoot.x = -100
                        shoot.y = -100

                        puntaje1 = puntaje1 + math.random(15,20)

                end

                if inmunidadTiempo > inmunidad then
                        if (colisionador(Value, nave))then
                                vidas = vidas - 1
                                nave.x = totalx / 2
                                nave.y = totaly / 2
                                inmunidadTiempo = 0
                        end
                end


                if nave.x > Value.x then
                        Value.x = Value.x + velEnemies
                else
                        Value.x = Value.x - velEnemies
                end

                if nave.y > Value.y then
                        Value.y = Value.y + velEnemies
                else
                        Value.y = Value.y - velEnemies
                end


                --table.remove(enemies, 1)   -- eliminar un objeto del array
```

```
                graficar(Value)
        end
        --

        if counterEnemies <= 0 and vidas > 0 then
                estado = 'endg1'
        end

        if vidas <= 0 then
                estado = 'endg1mal'
        end

        if inmunidadTiempo > inmunidad then
                canvas:compose(nave.x, nave.y, nave.img)
        else
                canvas:compose(nave.x, nave.y, nave2.img)
        end



        canvas:attrColor('white')
        canvas:attrFont('vera', 25)
        canvas:drawText(20,0, 'Vidas:'..vidas..'   Puntaje : '..puntaje1 .. ' ' )


        canvas:flush()

end

function graficar(A)
                canvas:compose(A.x, A.y, A.img)
end


function colisionador(B, A)

        local ax1, ay1 = A.x, A.y
        local ax2, ay2 = ax1 + A.dx, ay1 + A.dy
        local bx1, by1 = B.x, B.y
        local bx2, by2 = bx1 + B.dx, by1 + B.dy

        if ax2 < bx1  or bx2 < ax1 then
                return false
        elseif ay2 < by1 or by2 < ay1 then
                return false
        end

        --cambios de imagenes
        return true
end
```

```
local IGNORE = false




-- Si ocurre un evento se lo captura o procesa según el estado del juego:
-- Es importante tener esta l—gica del juego separada del resto de la l—gica del juego
-- ya que por ejemplo para reutilizar el c—digo y llevar un determinado juego a
-- otra plataforma como ejemplo en el caso de dispositivos m—viles se cambiar'a esta parte
-- para eventos touch.
function handler (evt)

if  (evt.class == 'user')  then

        local now = event.uptime()


                -- camina
                if evt.time then
                        local dt = now - evt.time
                end
                --crear un movil horizontal randómico cuando un objeto en movimiento

                        event.post('in', { class='user', time=now })


        end


        if IGNORE then
                return
        end

        --************************************************************
        --************************************************************
        --Salir del juego

         if (evt.class == 'key' and evt.type == 'press' ) then
           if evt.key == 'RED' then
                                event.post {
                                        class  = 'ncl',
            type   = 'presentation',
            label  = 'fim',
            action = 'start',
          }
       end
    end
```

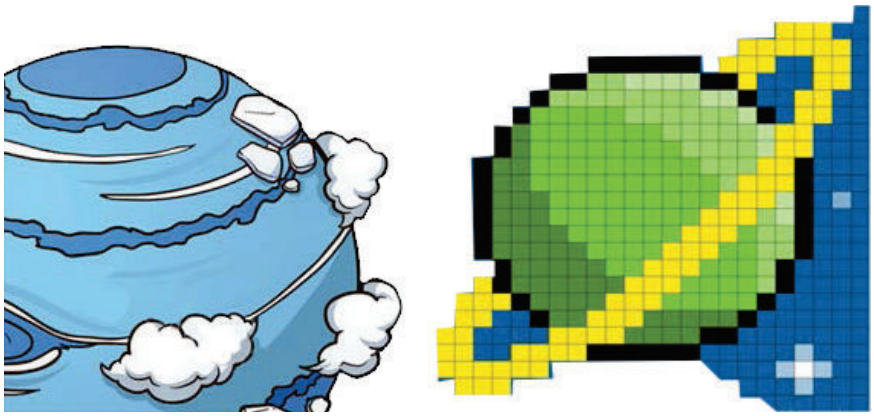
```
--************************************************************
--************************************************************
-- eventos teclado RUNNER NIVEL 2 TECLADO

if (evt.class == 'key' and evt.type == 'press' ) then
                        if estado == 'runner' then
                                    if evt.key == 'CURSOR_UP' then
                                                runupvar = 1
                                    end
                        end
            end

--************************************************************
--************************************************************
-- eventos de teclado PONG NIVEL 1
            if (evt.class == 'key' and evt.type == 'press' ) then
                        if estado == 'game0' then
                                    if evt.key == 'CURSOR_UP' then
                                                vsp = - vpaleta*factor
                                    elseif evt.key == 'CURSOR_DOWN' then
                                                vsp =  vpaleta*factor
                                    end
                        end
            end

--************************************************************
--************************************************************
-- eventos teclado final level NIVEL 3

            if (evt.class == 'key' and evt.type == 'press' ) then
                        if estado == 'finalLevel' then
                                    if evt.key == 'CURSOR_UP' then
                                                plusY = - 0.4
                                    elseif evt.key == 'CURSOR_DOWN' then
                                                plusY =  0.4
                                    elseif evt.key == 'CURSOR_LEFT' then
                                                plusX = - 0.4
                                    elseif evt.key == 'CURSOR_RIGHT' then
                                                plusX =  0.4
                                    elseif evt.key == 'ENTER' then
                                                naveShoot = 1
                                                coordsShoot = {nave.x, nave.y}
                                                shoot.x = nave.x
                                                shoot.y = nave.y
                                    end

                        end

            end
```

```
		if (evt.class == 'key' and evt.type == 'press' and estado == 'waiting') then
		-- iniciar el juego PRINCIPAL por NIVEL
		estado = sigEstado

		end



--*************************************************************
--*************************************************************
--*************************************************************
-- Codigo eventos de teclado reutilizable
--*************************************************************
--*************************************************************
	if (evt.class == 'key' and evt.type == 'press') then
		if estado == 'intro' then

			if evt.key == 'CURSOR_UP' then
				pk1 = pk1 - 1
				if pk1 < 0 then
				pk1 = pk1 + 1
				else
				pointer.y = pointer.y - 40
				end


			elseif evt.key == 'CURSOR_DOWN' then

				pk1 = pk1 + 1
				if pk1 > 3 then
					pk1 = pk1 - 1
				else
					pointer.y = pointer.y + 40
				end

			elseif evt.key == 'CURSOR_LEFT' then

				pk2 = pk2 - 1
				if pk2 < 0 then
				pk2 = pk2 + 1
				else
				pointer.x = pointer.x - 38.18
				end

			elseif evt.key == 'CURSOR_RIGHT' then
				pk2 = pk2 + 1
				if pk2 > 10 then
				pk2 = pk2 - 1
				else
				pointer.x = pointer.x + 38.18
				end

			elseif evt.key == 'ENTER' then
```

```
        if keycodes[pk2][pk1] == '|' then
                                                name = string.sub(name, 0, string.len(name) -
1)
                                        elseif keycodes[pk2][pk1] == '!' then
                                          estado = 'waiting'
                                          event.post('in', { class='user', time=now })
                                        else
                                                if string.len(name) == 15 then
                                                else
                                                name = name..keycodes[pk2][pk1]
                                                end
                                        end
                                end
                        end
                end

        --*********************************************************************
        --*********************************************************************
        -- Control del flujo del juego pantallas de inicio, espera y niveles
   -- Muestra las escenas segœn el estado

        if estado == 'intro' then
           redrawName()

        elseif estado == 'game0' then
                        --NIVEL 1
           pongGame()
           sigEstado = 'runner'
        elseif estado == 'runner' then
                        --NIVEL 2
                         runner()
        elseif estado == 'finalLevel' then
                        --NIVEL 3
                        finalLevel()
        elseif estado == 'endg1' then
                        --FIN EXITOSO
                        redrawfin1()
        elseif estado == 'endg1mal' then
                        --FIN FALLIDO
                        redrawEndLose()
        elseif estado == 'waiting' then
                        --MENSAJES E INSTRUCCIONES ENTRE NIVELES
                        waitingCanvas()
        end

end

event.register(handler)
```

ANEXO 2 IMÁGENES DISEÑADAS PARA EL VIDEOJUEGO

Planetas Tierra y Saturno:



Nave Espacial Sprites

Enemigos:

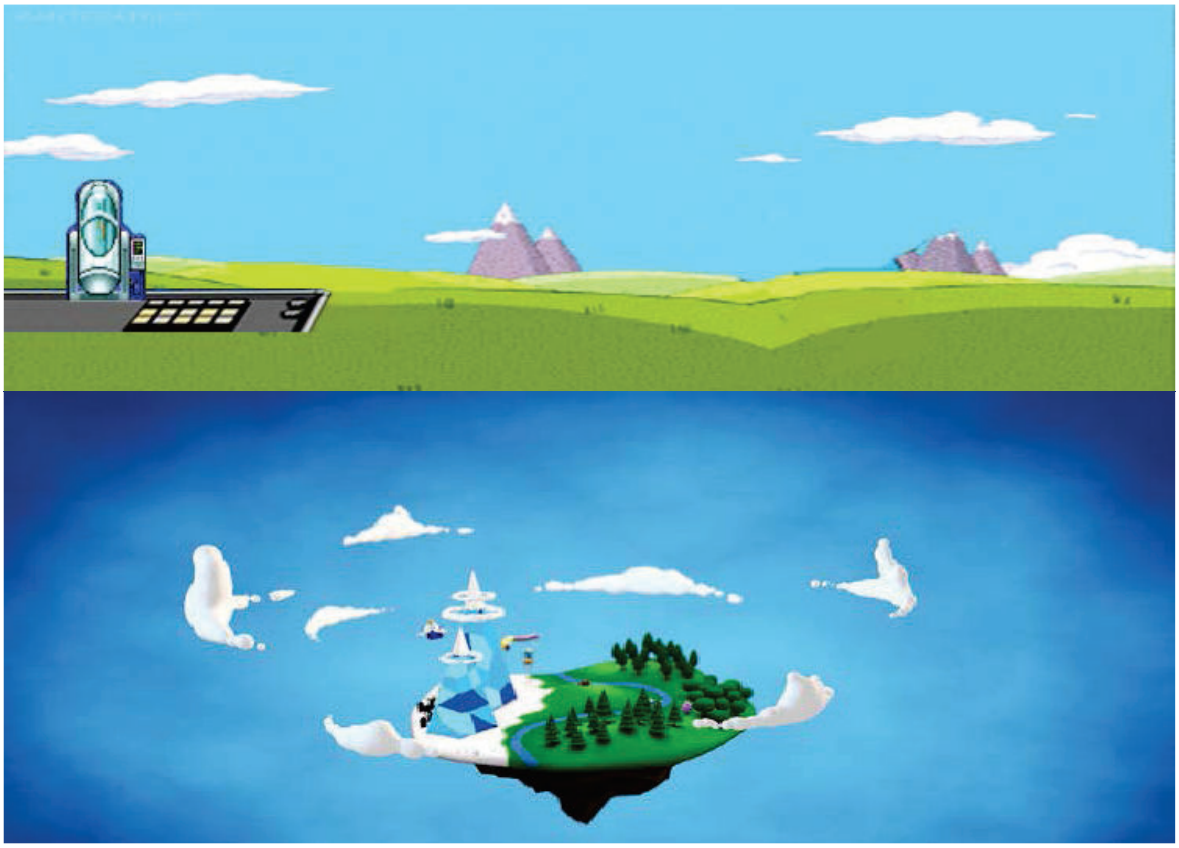
Monstruo Sprites

Alienígenas Invasores:

Proyectil:

Tanque:

Fondos de escenarios:



Teclado QWERTY en pantalla:

Nota: todas las imágenes y vectores utilizados o en este proyecto provienen de fuentes Royalty Free.