UNIVERSIDAD DEL ACONCAGUA

FACULTAD DE CIENCIAS SOCIALES Y ADMINISTRATIVAS

LICENCIATURA EN INFORMÁTICA
Y DESARROLLO DE SOFTWARE

DESARROLLO DE VIDEOJUEGOS

Tesina para optar al grado de
Licenciado en Informática y Desarrollo de Software

LUIS JESÚS ARCE

Tutor: Prof. ROSANA GIMÉNEZ

Mendoza, agosto de 2011

CALIFICACIÓN

ÍNDICE

RESUMEN

Esta tesina titulada *Desarrollo de Videojuegos*, trata acerca de las tecnologías que pueden utilizarse para llevar a cabo un proyecto de desarrollo de videojuegos.

En una primera instancia se establecen los objetivos generales de la tesina, entre los que podemos encontrar: el análisis y puesta a prueba de un motor de videojuegos de última generación, la presentación de otras aplicaciones externas que sirvan de apoyo al desarrollo de videojuegos, y el desarrollo de un juego de computadora. Seguidamente se plantea el videojuego a desarrollar: "Waku Dice" y se realiza un análisis sobre dos motores de videojuegos modernos: UDK y Unity, enunciando sus características generales más significativas. Consecutivamente se elige uno de ellos para llevar a cabo el desarrollo del videojuego "Waku Dice". Luego se proponen varios programas externos al Game Engine que permiten completar el proceso creativo del videojuego. Se selecciona la metodología de desarrollo y junto a ella se especifican las actividades más significativas de cada etapa del ciclo de vida y los artefactos que surgen de los distintos procesos. Finalmente se presentan los detalles del videojuego desarrollado.

Descriptores [videojuego, game engine, ingeniería de software]

I. INTRODUCCIÓN

I.1. Videojuegos

Un videojuego es concebido como un medio de entretenimiento, en donde se incluye a uno o varios usuarios, llamados *Players* o simplemente *Jugadores*, los cuales mantienen una interacción constante con varias interfaces, como pueden ser los joysticks o controles, teclado, mouse, entre otros, y un dispositivo de video (monitor de PC, TV, Realidad Virtual, etc).

Los videojuegos pueden recrear situaciones virtuales de cualquier índole (fantasía, acción, lógica, deporte, terror, etc.), entornos en donde las reglas de la física pueden romperse y distorsionarse, mundos fantásticos o inclusive situaciones con un realismo extraordinario. Todo esto se logra gracias a la conjunción de numerosas artes multimediales que estimulan los sentidos y encienden los sentimientos de los jugadores. Así es como un videojuego, a través de determinadas reglas, sumerge al jugador en una situación en donde se deben resolver problemas, eliminar obstáculos, cumplir objetivos y superar metas para llegar al final y obtener algún tipo de recompensa.

Teniendo en cuenta que fueron débilmente considerados en el pasado (generalmente eran desarrollados por uno o dos programadores con poca experiencia), los videojuegos como medio de entretenimiento, han evolucionado de manera abrupta en los últimos años, convirtiéndose en una industria multimillonaria, superando inclusive a la industria del cine, en casos de superproducciones de varios años de desarrollo que cuentan con cientos de programadores, artistas gráficos y de sonido, narración, marketing, leyes, inclusive actores!. Mientras que en el pasado, no se necesitaba más de tres personas y el desarrollo se prolongaba por no más de un trimestre.

Desarrollar un videojuego implica conocer de varias disciplinas y poseer varias habilidades, pasando por la creatividad y recorriendo varias ciencias formales y sociales, cuyo proceso va mucho

más allá de un típico desarrollo de software. Así es como los videojuegos unifican el arte, la ciencia y la tecnología.

I.1.1. Motor de Videojuegos

Un M***otor de Videojuegos*** *(*en inglés ***Game Engine***) es una aplicación de software que ofrece todas las herramientas necesarias para el diseño y desarrollo completo de un videojuego, disponiendo de un motor de renderizado para gráficos 2D y 3D, detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes, streaming, administración de memoria y mucho más.

I.2. Problema de investigación

Actualmente existen numerosas herramientas informáticas para la creación de Videojuegos, por eso es prioritario en una primera instancia, fijar los objetivos y premisas para desarrollar el videojuego, y luego seleccionar las herramientas o tecnologías de desarrollo. Por ejemplo, si se va a realizar un juego de aviones en 3D, se necesitará algún programa para manipular objetos y escenarios en 3D (como Blender o 3D Studio Max), interfaces de usuario (como Photoshop, GIMP, Inkscape), Sonido (como Audacity o Cool-Edit) y además se deberá trabajar con un Motor de Videojuegos o Game Engine apropiado, es decir, que sea lo suficientemente flexible para producir ese tipo de juegos. En cambio, si el videojuego a desarrollar es una aventura gráfica en 2D, probablemente se utilizará un motor con otras características que beneficien la producción de ese tipo de juegos y no será necesario utilizar programas de edición 3D.

Crear un videojuego sencillo con un Game Engine moderno, es una buena manera de ponerlo a prueba y analizarlo. No obstante, al existir multitud de ellos, se debe seleccionar de una extensa lista basándose en la propia experiencia y en datos conocidos que provienen del videojuego (o la idea del

mismo) que se desea producir.

I.3. Proposición

La propuesta para esta tesina, es llevar a cabo una selección de tecnologías que hacen al desarrollo de un videojuego, analizando un Game Engine de última generación y las aplicaciones externas que complementan el trabajo (programas de sonido, gráfica y multimedia en general). Dicho análisis no se limita a ser teórico únicamente, sino que pretende incursionar en la práctica de la utilización del Game Engine a través de la realización de un videojuego, añadiendo al caso, una metodología de desarrollo que permita llevar adelante el proyecto.

I.4. Objetivos del Proyecto

- Analizar y poner a prueba un Game Engine de última generación.
- Listar y proponer otras aplicaciones externas que sirvan de apoyo al desarrollo de videojuegos.
- Desarrollar un juego de computadora sencillo con las tecnologías propuestas y analizadas: Game Engine y Aplicaciones externas.

I.5. Videojuego propuesto

Nombre del juego:

"Waku dice".

Descripción general:

El juego se basa en una secuencia de colores generada por la computadora que debe ser repetida por el usuario. Es similar al juego "Simón dice". Posee cuatro niveles de dificultad con 10 partidas por cada nivel. En una partida se genera una secuencia aleatoria de colores que el jugador deberá repetir correctamente para ganar y pasar a la siguiente partida.

Mecánica:

Cuando comienza la partida, la computadora generará una secuencia de colores, la cual podrá escucharse además de visualizarse en la pantalla. Una vez finalizada la secuencia, es el turno del jugador, el cual deberá realizar la misma secuencia para ganar la partida.

Para seguir una secuencia, el jugador debe pulsar (hacer click), sobre las esferas de color que rodean la prisión de Waku. Por ejemplo: si la computadora genera la siguiente secuencia: Verde-Azul-Rojo-Verde, el jugador deberá hacer click en las esferas de color en el orden correspondiente, de esa manera completará satisfactoriamente la partida. Si el jugador se equivoca en la secuencia, perderá la partida y se le descontará una vida.

Cada vez que el jugador gana una partida, estará más cerca de liberar a Waku, esto se puede observar en la barra de progreso que aparece en la esquina inferior izquierda de la pantalla.

Cuando se completa el 100% de la barra de progreso, Waku será liberado de su prisión y el jugador ganará el nivel.

I.5.1. Premisas para el desarrollo

Para el videojuego propuesto se definieron algunas premisas a cumplir. Esto permite fijar características escenciales que se tomarán en cuenta a la hora de seleccionar las herramientas de desarrollo (Game Engine y aplicaciones externas).

Las premisas son las siguientes:

- El videojuego debe ser del género *Serious Game*.
- El videojuego debe tener elementos 2D y 3D: esto permite analizar la potencia del motor de videojuegos en ambos aspectos (renderizado y gráficos).
- El videojuego debe ser lo más completo posible, en términos de multimedia e interactividad: de este modo se puede indagar sobre otros aspectos del game engine como el manejo de música y sonidos, iluminación, sistemas de partículas, interacción con el usuario, etc. Además, se hará necesario el uso de programas externos al motor de videojuegos, como herramientas de desarrollo y soporte.
- El videojuego no debe contener violencia: esto es una regla moral personal, cuya filosofía es colocar a las tecnologías informáticas, particulamente el desarrollo de videojuegos, hacia propósitos éticos y educativos.

I.6. Marco Teórico-Conceptual

I.6.1. Historia de los videojuegos

La siguiente información muestra una línea de tiempo con los sucesos más importantes en la historia de los videojuegos desde sus comienzos hasta el año 2001. Dicha información ha sido traducida y adaptada para esta tesina desde el libro "*The Ultimate History Of Video Games*"[1].

1889

Fusajiro Yamacuchi establece *Marufuku Company*, para fabricar y distribuír *Hanafuda*, un juego de cartas japonesas.

1932

La compañía *Connecticut Leather s*e establece por un inmigrante ruso llamado Maurice Greenberg para distribuir los productos de cuero a los zapateros.

1951

Yamauchi cambia el nombre de *Marufuku Co. Ltd*. a *Nintendo*, un término que significa "dejar la suerte al cielo".

Estados Unidos aprueba nuevas leyes que regulan las máquinas tragamonedas (slot machines). Marty Bromley, que gestiona las salas de juego en las bases militares en Hawai, compra máquinas y abre *Service Games*, mejor –conocido como *SEGA*.

David Rosen, que regresa del servicio en la Fuerza Aérea de los EE.UU. durante la Guerra de Corea, abre un negocio de retratos de pintura en Japón.

1 Steven L. Kent. *The Ultimate History Of Video Games* (1º Edición). New York, Three Rivers Press, 2001.

1954

David Rosen abre *RosenEnterprises* y comienza a enviar cabinas de fotos a Japón.

1956

Rosen importa a Japón un valor de $200.000 en juegos electromecánicos que funcionan con monedas.

1958

Willy Higinbotham, físico de *Brookhaven National Laboratories* en Nueva York inventa un sistema interactivo de *ping-pong*, juego que se muestra en un osciloscopio.

1961

Un estudiante del *MIT (Massachusetts Institute of Technology)*, Steve Rusell, crea *Spacewar*, el primer juego de computadora interactivo.

1962

Nolan Bushnell entra a la escuela de Ingeniería en la Universidad de Utah.

Rosen Enterprises, la mayor compañía de entretenimiento de Japón, se fusiona con *Service Games* (que ahora cuenta con máquinas tocadiscos en más de 6.000 localidades), para formar *Sega Enterprises*.

1965

Nolan Bushnell consigue un trabajo de verano en un carnaval de *Salt Lake City*, donde está a

cargo de la mitad de los juegos.

1966

Ralph Baer comienza una investigación sobre los juegos interactivos de televisión en *Sanders Associates.*

Sega lanza *Periscope*, un juego que se convierte en un éxito en Japón, tanto que las empresas de EE.UU. y Europa comienzan a importarlo. Este es el primer exportador de juegos de entretenimiento de Japón. Debido a los altos costos de envío, los propietarios de juegos de arcade de EE.UU. cobran a los jugadores $0.25 por juego, determinando así lo que se convertrá en el precio estándar de los mismos.

1968

Ralph Baer patenta su juego interactivo de television.

1969

Gulf & Western compra Sega.

Nolan Bushnell se gradúa en la Universidad de Utah y acepta un trabajo en California.

1970

Magnavox licencia el juego de televisión de Ralph Baer de *Sanders Associates*.

1970

Bushnell comienza a trabajar en una versión arcade del *Spacewar* llamada *Computer Space*.

1971

Nutting Associates compra *Computer Space* de Nolan Bushnell y lo contrata para que ayude a fabricarlo.

Nutting comienza a comercializar *Computer Space*, el primer videojuego de la máquina de arcade.

1972

Magnavox comienza demostrando *Odyssey* en exhibiciones privadas. Bushnell asiste a una demostración de la consola el 24 de Mayo, en Burlingame, California.

Bushnell deja *Nutting* y comienza *Syzygy* con su socio Ted Dabney. Conclusión de que el nombre *Syzygy* ya está utilizado, por lo que renombran su compañía a *Atari*.

Al Alcorn, ingeniero de *Atari*, crea el famoso *Pong*.

Magnavox lanza *Odyssey*.

Magnavox demanda a *Atari* por considerar que Pong infringe las patentes de Ralph Baer.

1973

Taito, *Williams*, y *Midway* entran en el negocio de los videojuegos.

1975

Atari crea un prototipo de la unidad *Home Pong* y vende la idea a Sears Roebuck.

Namco comienza a hacer juegos de video.

En busca de fondos, Nolan Bushnell se acerca al inversor de capital de riesgo Don Valentine para su financiación. *Midway Games* importa un juego de *Taito* llamado *Gunfight*, el primer juego que utiliza un microprocesador.

1976

La compañía *Connecticut Leather*, ahora conocida como *Coleco*, presenta *Telstar*, un juego de Tennis de televisión.

Fairchild Camera & Instrument lanza *Channel F*, el primer juego de casa programable para el uso de cartuchos.

Exidy Games lanza *Death Race*, un juego en el que los jugadores deben conducir y atropellar figuras humanas. Las protestas sobre el juego aparecen a los 60 minutos de su lanzamiento.

Bushnell vende *Atari* a *Warner Communications* por $28 millones.

1977

Atari abre el primer *Pizza Time Theatre*, originalmente creado como un lugar donde los niños podían ir a comer pizza y jugar videojuegos.

Atari lanza el *Video Computer System*, también conocido como el *Atari 2600*.

Mattel Electronics presenta un sistema LED basado en juegos de video portátiles.

Shigeru Miyamoto se une a *Nintendo*.

Bally lanza la consola *Bally Professional Arcade*.

Nintendo lanza su primer videojuego doméstico en Japón.

1978

Bushnell es forzado a salir de *Atari* y compra los derechos del *Pizza Time Theatre*.

Ray Kassar se convierte en el Director Ejecutivo (CEO) de *Atari*

Nintendo lanza *Othello*, su primer juego de arcade.

Atari lanza *Football* y *Midway* lanza *Space Invaders*. Ambos juegos generan altos records de negocios.

Magnavox lanza *Odyssey2*.
Cinematronics lanza *Space Wars*, una adaptación arcade del juego *Spacewars* (creado en el *MIT* por Steve Rusell).

1979

Se funda *Capcom* en Japón.

Atari lanza *Lunar Lander*, su primer juego con gráficos vectoriales. Posteriormente, *Atari* lanza *Asteroids*, y se convierte en la compañía de juegos con más éxitos vendidos.

El diseñador de juegos de *Atari*, Warren Robinett, introduce el concepto de "Huevos de Pascua" (*Easter Eggs*) a los videojuegos por ocultar una habitación con su nombre en un juego de la 2600 llamado *Aventure*.

Mattel Electronics presenta la consola de juegos *Intellivision*.

Milton Bradley lanza *Microvision*, el primer sistema de *juegos de bolsillo* programable.

1980

Atari lanza *Space Invaders* para la *Video Computer System*. La práctica de vender versiones domésticas (para jugar en casa) de algunos éxitos de arcade ha comenzado.

Programadores renegados que huyen de *Atari* crean *Activision*. Son los primeros "desarrolladores terceros" de videojuegos para la *Video Computer System*.

Namco lanza *Pac-Man*, el juego de arcade más popular de todos los tiempos. Más de 300.000 unidades se venden en todo el mundo.

Minoru Arakawa abre *Nintendo of America*.

Williams lanza *Defender*.

1981

Nintendo lanza el juego de arcade *Donkey Kong*.

Atari lanza *Pac-Man* para la *Video Computer System*.

Atari lanza *Tempest*.

U.S. arcades revenues reach $5 billion as Americans spend more than 75,000 man-hours playing
video games.

En EE.UU. la industria de los arcades alcanza un ingreso de $5 billones, donde los estadounidenses pasan más de 75.000 horas-hombre jugando videojuegos.

Arnie Katz, Bill Kunkel, y Joyce Worley comienzan la publicación de *Electronic Games*, la primer revista sobre videojuegos.

1982

Coleco lanza la consola *Colecovision*.

Atari gana una demanda acusando a *Magnavox* por infringir su licencia de *Pac-Man* con el juego *K.C. Munchkin*.
Atari lanza *E.T.* para la *Video Computer System*.

Activision lanza *Pitfall* para la *Video Computer System*.

Atari lanza la consola *5200*.

General Consumer Electronics lanza la consola *Vectrex*.

Midway lanza *Ms. Pac-Man*, el mayor juego de arcade de la historia americana.

Cuando *Warner Communications* anuncia que las ventas de *Atari* no han satisfecho las

predicciones, sus acciones caen un 32 por ciento.

1983

Nolan Bushnell abre una compañía de arcades llamada *Sente Games*.

Yu Suzuki se une a *Sega*.

Sega lanza su primer consola doméstica en Japón, la *SG 1000*.

Cinematronics lanza *Dragon's Lair*, el primer juego de arcade en incorporar la tecnología *laser-disk*.

El ex ejecutivo de *Philip Morris*, James Morgan, reemplaza a Ray Kassar como jefe de *Atari*.

1984

Nintendo lanza la *Family Computer* (*Famicom*) en Japón.
Isao Okawa y David Rosen aquieren *Sega Enterprises* por $38 millones.

Coleco comienza a comercializar la *Adam Computer*.

Hisao Oguchi y Yuji Naka se unen a *Sega*.

Warner Communications vende *Atari Corporation* a *Commodore Computers* pero conserva la división de arcade como *Atari Games*.

1985

Nintendo prueba los mercados de la consola *Famicom* en Nueva York como *Nintendo*

Entertainment System (*NES*).

El matemático ruso Alex Pajitnov diseña *Tetris*.

1986

Nintendo of America lanza *NES* en todo EE.UU.

Sega lanza su consola *Sega Master System.*

Atari lanza la consola *7800*.

1987

Nintendo publica *The Legend of Zelda*.

NEC lanza en Japón la consola híbrida de 16-bits/8-bits *PC-Engine*.

Sega devela su consola de 16-bits *Mega Drive*.

1988

Square Soft publica *Final Fantasy*.

Atari Games lanza juegos para la *NES* sin licencia bajo su nuevo sello *Tengen*.

Tonka adquiere los derechos de distribución en EE.UU. de la *Sega Master System*.

Coleco se declara en bancarrota.

1989

NEC lleva la *PC-Engine* a los EE.UU. y la libera con el nombre de *TurboGrafx*.

Sega lanza la consola *Mega Drive* en EE.UU. como *Genesis*.

Nintendo lanza la *Game Boy* en todo el mundo.

1990

Nintendo y *Atari* acuden a los tribunals por los derechos de *Tetris*.

Nintendo lanza *Super Mario Bros 3*. Uno de los juegos de cartucho más exitosos de todos los tiempos.

SNK presenta su consola de 24-bits *NeoGeo* en los EE.UU.

1991

Nintendo of America lanza la consola *Super NES* (*SNES*).

Sega se recrea con una nueva mascota y lanza *Sonic The Hedgehog*.

Galoob Toys lanza *Game Genie (una serie de cartuchos para trucar videojuegos)*.

Capcom lanza el juego de arcade *Street Fighter II*.

1992

Con *Genesis* vendiendo más que *Super NES, Sega* toma el control del mercado de consolas en EE.UU.

Sega lanza *Sega CD*, periférico para la consola *Genesis*.

1993

Panasonic comienza a comercializar la videoconsola de 32-bits *3DO Multiplayer*.

Atari lanza la consola de 64-bits *Jaguar*.
Broderbund publica el juego *Myst* para las computadoras *Macintosh*.

Id Software publica *Doom* para *PC*.

Virgin Interactive Entertainment publica *The 7th Guest* para *PC CD-ROM*.

Los senadores Joseph Lieberman (D. of Connecticut) and Herb Kohl (D. of Wisconsin) ponen en marcha audicencias en el Senado sobre la violencia en los Videojuegos.

1994

The Interactive Digital Software Association es creada en respuesta a las audiencias del Senado.

Nintendo lanza *Donkey Kong Country* y retoma el control del mercado de consolas en EE.UU.

Sega lanza *32X*, un periférico que aumenta el poder de la *Genesis*.

Sega lanza la consola *Saturn* en Japón.

Sony lanza la *PlayStation* en Japón.

1995

Sega lanza *Saturn* en los Estados Unidos.

Sony lanza *PlayStation* en los Estados Unidos.

Nintendo lanza *Virtual Boy* en los Estados Unidos.

Nintendo da a conocer la consola de 64-bits *Nintendo 64* en Japón.

1996

Jack Tramiel vende *Atari Corporation* a un fabricante de unidades de disco, *JTS*.

Nintendo lanza la *Nintendo 64* en los Estados Unidos.

Nintendo descontinúa la producción de *Virtual Boy*.

Sony anuncia el juego *Crash Bandicoot*.

1997

Sega descontinúa la producción de *Saturn*.

Bandai lanza *Tamagotchi*.

Tiger lanza *game.com* una videoconsola portátil.

Gumpei Yokoi, el creado de la *Game Boy*, muere en un accidente automovilístico.

DreamWorks, *Universal*, y *Sega* se unen para formar una nueva línea de super arcades llamada *GameWorks*.

Nintendo lanza *Goldeneye 007* para la *Nintendo 64*.
Square Soft publica *Final Fantasy VII* para la *PlayStation*.

1998

Nintendo lanza *The Legend of Zelda: Ocarina of Time* para *Nintendo 64*.

Pokemon, una línea de juegos de rol para *Game Boy* que ha desencadenado furor en Japón, llega a los Estados Unidos donde se inicia una locura similar.

1999

JTS se declara en bancarrota y vende propiedades de Atari a *Hasbro Interactive*.

SNK lanza la consola portátil *NeoGeo Pocket Color* en los Estados Unidos.

Sega lanza la consola *Dreamcast* en los Estados Unidos.

2000

Toshiba y *Samsung* anuncian planes de vender una DVD-consola llamada *Nuon*.

Sony lanza la *PlayStation 2* en Japón.

Microsoft da a conocer los planes para la consola *Xbox* en la *Conferencia de Desarrolladores de Videojuegos*.

Sega lanza *SegaNet* como un servicio de internet para la *Dreamcast*.

Sony lanza la PlayStation 2 en los Estados Unidos.

SNK frena la producción y las ventas de *NeoGeo Pocket Color* en los Estados Unidos.

2001

Sega frena la producción de la *Dreamcast*.

El president de *Sega*, Isao Okawa, pierde la vida.

Nintendo lanza *Game Boy Advance* en Japón (Marzo) y en los Estados Unidos (Junio).

Nintendo lanza *GameCube* en los Estados Unidos.

Microsoft lanza la *Xbox* en todo el mundo.

1.6.2. Género de videojuego

Existe una serie de cualidades que definen a un videojuego y lo categorizan bajo lo que se denomina *Género de Videojuego*. Los géneros agrupan o clasifican bajo un mismo nombre a los videojuegos que tienen un conjunto de elementos o características similares. Así podemos encontrar algunos géneros como: aventura, disparos, serios, estrategia, lucha, terror, plataformas, rol, musicales, simulación, deportivos, entre otros.

Conocer de antemano el género de un videojuego a desarrollar, permite enumerar algunas características que luego se tomarán en cuenta a la hora de seleccionar el Game Engine. Cabe mencionar también que los géneros de videojuegos no son excluyentes entre sí, esto significa que un videojuego puede ser de varios géneros diferentes, por ejemplo: "acción y estrategia" o "terror y disparos".

I.6.3. Serious Game o Juego Serio

Lucio Margulis explica de un manera sencilla qué son los juegos serios, el video que contiene dicha explicación se encuentra en la ruta **/multimedia/Explicacion - Juego Serio** del DVD adjunto con esta tesina.

Para una definición más precisa se puede consultar el libro ***Serious Games - Games That Educate, Train, and Inform*** [2], donde se explica lo siguiente:

2 Michael, David; Chen, Sande. *Serious Games: Games That Educate, Train, and Inform*. Primera edición.

¿Qué es un "juego serio"?

Pregúntele a la mayoría de los desarrolladores de juegos y a los jugadores hardcore sobre los "juegos serios", y aprenderá que todos los juegos son serios. En otras palabras, los desarrolladores y los jugadores toman sus juegos muy en serio. Ellos viven para hacer juegos, y viven para jugar. ¿Qué podría ser más serio que eso?

Para la población en general, sin embargo, el término "juegos serios" suena como una contradicción. Las dos palabras parecen excluirse mutuamente. ¿Cómo puede algo ser serio y un juego a la vez?

Así, de nuevo, ¿qué es un juego serio?

Una simple explicación de muchos profesionales del ámbito, con algunas reservas y salvedades, es la siguiente: *Un juego serio es un juego en el que la educación (en sus diversas formas) es el principal objetivo, en lugar de entretenimiento*.

Esta definición ayuda a aclarar la contradicción aparente, pero sin duda también genera confusión en algunas personas (tanto desarrolladores de videojuegos como educadores) que ven incompatibles al entretenimiento y la educación. Es la afirmación de este libro, que no sólo la educación y el entretenimiento no son conflictivas entre sí, sino que hay muchos lugares donde los dos conceptos se superponen y donde cada parte puede utilizar las herramientas de la otra para lograr sus objetivos.

Mientras que la frase "juegos serios" es relativamente reciente, y todavía nuevo para muchas personas, hay un término más antiguo que la mayoría se reconoce: ***edutainment*** (educación y entretenimiento).

Edutainment, o la educación a través del entretenimiento, es un término que entró en uso común en la década de 1990 con la aparición de los ordenadores personales "multimedia". Edutainment no se limita a los juegos de video y se refiere a cualquier forma de educación que también trata de entretener. A menudo se refiere a los videojuegos con fines educativos, especialmente para los preescolares y los nuevos lectores.

La siguiente clasificación de Juegos Serios fue extraída desde la publicación original *"¿Qué son los Serious Games?"* [3] por José Vicente Pons Alfonso. Dicha información fue actualizada y adaptada para su presentación en esta tesina.

Clasificación de Serious Games.

- **Advergaming**

  Los advergames son videojuegos interactivos que permiten una exposición continuada del usuario ante la marca o producto publicitado.

  Ejemplo: el juego de motos de *Repsol YPF* [4] para promocionar su portal web.

3 José Vicente Pons Alfonso. *¿Qué son los "Serious Games"*?. http://www.exelweiss.com/blog/356/serious-games-juegos-serios/

4 El juego de motos de Repsol YPF está disponible en la dirección web: http://www.repsol.com/es_es/ventajas_del_portal/juegos/simuladores/Simulador_motociclismo.aspx

- **Juegos educativos y de entrenamiento**

  Juegos que tratan de formar con aprendizaje asistido. Los juegos educativos más comunes son los orientados a niños, normalmente clasificados por materias y edades.

  Por otra parte, en los juegos de entrenamiento, las personas son capaces de moverse e interactuar libremente, experimentando la vida como simulaciones, como si estuvieran realmente allí. La experiencia interactiva y realista ayuda a las personas a aprender más rápidamente, recordar mejor los procedimientos y desarrollar las competencias esenciales.

  Ejemplo: *Virtual Safety Training & Education Platform* [5]

  Otro ejemplo de este tipo de videojuego son los juegos militares, o que tratan varios aspectos del entrenamiento militar (desde combates de guerrilla a pilotaje de aviones). Son juegos por encargo y "mods" pagados directamente por las agencias militares.

5 *Virtual Safety Training & Education Plataform*. Disponible en world wide web: http://www.vstep.nl/

Ejemplo: *America's Army* [6]

- **Salud**

  Los juegos sobre la salud (también llamados "juegos de bienestar" o "health games"), tratan temas relacionados con la prevención de enfermedades, la vida sana, el ejercicio, etc. Es el tipo de juego más investigado por claros beneficios sociales que aportan.

  Ejemplo: *ReMission* [7], juego que trata sobre el tratamiento de células cancerígenas.

6 *America's Arrmy*. Disponible en world wide web: http://www.americasarmy.com/

7 *ReMission*. Disponible en world wide web: http://www.re-mission.net/site/game/index.php

- **Politicos y Sociales**

  Estos son juegos diseñados para educar al público sobre sus derechos y obligaciones como ciudadanos, para fomentar los comportamientos cívicos y las normas de convivencia. Las ONG los utilizan para denunciar injusticias o para promover la conciencia social.

  Algunos grupos políticos también los utilizan para comunicar algunos aspectos de sus campañas o de sus gobiernos.

  Ejemplo: *Food Force* [8], juego que pretende enseñar a los niños acerca de los desafíos logísticos de entregar ayuda alimentaria en una gran crisis humanitaria.



- **Negocios**

  Son juegos en los que se simulan estrategías relacionadas con situaciones económicas y de control de empresas.

  Ejemplo: *Capitalism2* [9], un juego de estrategia que cubre varios aspectos de negocio que se podrían encontrar en el mundo real.

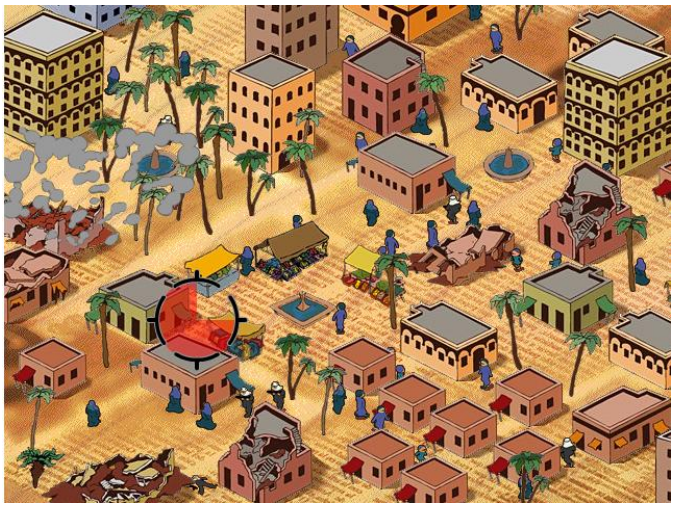
8 *Food Force*. Disponible en world wide web: http://www.wfp.org/how-to-help/individuals/food-force

9 *Capitalism2*. Disponible en world wide web: http://www.enlight.com/capitalism2/

- **"News Games"**

  Son juegos periodísticos (del inglés *news*, es decir, noticia) que informan sobre eventos recientes o expresan un comentario editorial.

  Ejemplo: *September 12th* [10], un juego creado por Gonzalo Frasca que denuncia el uso de la violencia para resolver el problema del terrorismo.



10 *September 12th*. Disponible en world wide web: http://www.newsgaming.com/games/index12.htm

Para concluír el análisis sobre serious games, desde el portal *http://www.seriousgames.es* se ha desarrollado una interesante lista de juegos serios categorizados por subgénero. Se puede acceder a dicha lista desde la siguiente dirección: http://www.seriousgames.es/2009/04/21/lista-de-serious-games/

I.6.4. Ingenieria de Software

La tecnología que comprende un proceso, un juego de métodos y un conjunto de herramientas se llama ingeniería del software.

El proceso

¿Qué es?: Cuando trabaja para construir un producto o un sistema, es importante seguir una serie de pasos predecibles (un mapa de carreteras que le ayude a obtener el resultado oportuno de calidad). El mapa de carreteras a seguir es llamado proceso del software.

¿Quién lo hace?: Los ingenieros de software y sus gestores adaptan el proceso a sus necesidades y entonces lo siguen. Además las personas que han solicitado el software tienen un papel a desempeñar en el proceso del software.

¿Por qué es importante?: Porque proporciona estabilidad, control y organización a una actividad que puede, si no se controla, volverse caótica.

¿Cuáles son los pasos?: A un nivel detallado, el proceso que adoptemos depende del software que estamos construyendo. Un proceso puede ser apropiado para crear software de un sistema de aviación, mientras que un proceso diferente por completo puede ser adecuado para la creación de un sitio web.

¿Cuál es el producto obtenido?: Desde el punto de vista de un ingeniero de software, los productos obtenidos son programas, documentos y datos gue se producen como consecuencia de las actividades de ingeniería del software definidas por el proceso.

Pero, ¿qué es exactamente el proceso del software desde un punto de vista técnico?: definimos un proceso de software como un marco de trabajo de las tareas que se requieren para construir software de alta calidad.

El proceso define un marco de trabajo para un conjunto de áreas clave de proceso que se deben establecer para la entrega efectiva de la tecnología de la ingeniería del software. Las áreas claves del proceso forman la base del control de gestión de proyectos del software y establecen el contexto en el que se aplican los métodos técnicos, se obtienen productos del trabajo (modelos, documentos, datos, informes, formularios, etc.), se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.

Los métodos

Los métodos de la ingeniería del software indican cómo construir técnicamente el software. Los métodos abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Los métodos de la ingeniería del software dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Las herramientas

Las herramientas de la Ingeniería del software proporcionan un enfoque automático o semi-automático para el proceso y para los métodos. Cuando se integran herramientas para que la información creada por una herramienta la pueda utilizar otra, se establece un sistema de soporte para el desarrollo del software llamado ingeniería del software asistida por computadora (CASE).

Fases genéricas de la ingeniería de software

El trabajo que se asocia a la ingeniería del software se puede dividir en tres fases genéricas, con independencia del área de aplicación, tamaño o complejidad del proyecto.

La fase de **definición** se centra sobre el ***qué***. Es decir, durante la definición, el que desarrolla el software intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones de diseño existen, y qué criterios de validación se necesitan para definir un sistema correcto. Por tanto, han de identificarse los requisitos clave del sistema y del software. Aunque los métodos aplicados durante la fase de definición variarán dependiendo del paradigma de ingeniería del software (o combinación de paradigmas) que se aplique, de alguna manera tendrán lugar tres tareas principales: ingeniería de sistemas o de información, planificación del proyecto del software y análisis de los requisitos.

La fase de **desarrollo** se centra en el ***cómo***. Es decir, durante el desarrollo un ingeniero del software intenta definir cómo han de diseñarse las estructuras de datos, cómo ha de implementarse la función dentro de una arquitectura de software, cómo han de implementarse los detalles procedimentales, cómo han de caracterizarse interfaces, cómo ha de traducirse el diseño en un lenguaje de programación (o lenguaje no procedimental) y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo variarán, aunque las tres tareas específicas técnicas deberían ocurrir siempre: diseño del software, generación de código y prueba del software.

La fase de **mantenimiento** se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software y a cambios debidos a las mejoras producidas por los requisitos cambiantes del cliente.

Modelo de proceso del software

Para resolver los problemas reales de una industria, un ingeniero del software o un equipo de ingenieros debe incorporar una estrategia de desarrollo que acompañe al proceso, métodos y herramientas junto con las fases genéricas discutidas anteriormente. Esta estrategia a menudo se llama modelo de proceso o paradigma de ingeniería del software.

Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizarse, y los controles y entregas que se requieren.

Modelo evolutivo de proceso del software

Se reconoce que el software, al igual que todos los sistemas complejos, evoluciona con el tiempo. Los requisitos de gestión y de productos a menudo cambian conforme a que el desarrollo proceda haciendo que el camino que lleva al producto final no sea real; las estrictas fechas tope del mercado hacen que sea imposible finalizar un producto completo, por lo que se debe introducir una versión limitada para cumplir la presión competitiva y de gestión; se comprende perfectamente el conjunto de requisitos de productos centrales o del sistema, pero todavía se tienen que definir los detalles de extensiones del producto o sistema. En estas y en otras situaciones similares, los ingenieros del software necesitan un modelo de proceso que se ha diseñado explícitamente para acomodarse a un producto que evolucione con el tiempo.

El modelo lineal secuencial se diseña para el desarrollo en línea recta. En esencia, este enfoque en cascada asume que se va entregar un sistema completo una vez que la secuencia lineal se haya finalizado.

El modelo de construcción de prototipos se diseña para ayudar al cliente (o al que desarrolla) a comprender los requisitos. En general, no se diseña para entregar un sistema de producción.

En ninguno de los paradigmas de ingeniería del software se tiene en cuenta la naturaleza evolutiva del software. Los modelos evolutivos son iterativos. Se caracterizan por la forma en que permiten a los ingenieros del software desarrollar versiones cada vez más completas del software.

El modelo incremental

El modelo incremental combina elementos del modelo lineal secuencial (aplicados repetidamente) con la filosofía interactiva de construcción de prototipos. El modelo incremental aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un **incremento** del software. Por ejemplo, el software de tratamiento de textos desarrollado con el paradigma incremental podría extraer funciones de gestión de archivos básicos y de producción de documentos en el primer incremento; funciones de edición más sofisticadas y de producción de documentos en el segundo incremento; corrección ortográfica y gramatical en el tercero; y una función avanzada de esquema de página en el cuarto.

Se debería tener en cuenta que el flujo del proceso de cualquier incremento puede incorporar el paradigma de construcción de prototipos.

Cuando se utiliza un modelo incremental, el primer incremento a menudo es un producto esencial. Es decir, se afrontan requisitos básicos, pero muchas funciones suplementarias (algunas conocidas, otras no) quedan sin extraer. El cliente utiliza el producto central (o sufre la revisión detallada). Como un resultado de utilización y/o de evaluación, se desarrolla un plan para el incremento siguiente. El plan afronta la modificación del producto central a fin de cumplir mejor las necesidades del cliente y la entrega de funciones, y características adicionales. Este proceso se repite siguiendo la entrega de cada incremento, hasta que se elabore el producto completo.

Información obtenida de: *Ingeniería del Software. Un enfoque práctico*.[11]

11 Roger S. Pressman. *Ingeniería del Software – Un enfoque práctico* (5º edición).

I.6.5. El Software Libre y su filosofía

Según GNU[12]. El Software Libre es una cuestión de *libertad*, no de precio.

- La libertad de ejecutar el programa, para cualquier propósito (libertad 0).
- La libertad de estudiar cómo trabaja el programa, y cambiarlo para que haga lo que usted quiera (libertad 1). El acceso al código fuente es una condición necesaria para ello.
- La libertad de redistribuir copias para que pueda ayudar al prójimo (libertad 2).
- La libertad de distribuir copias de sus versiones modificadas a terceros (la 3ª libertad). Si lo hace, puede dar a toda la comunidad una oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición necesaria para ello.

Un programa es software libre si los usuarios tienen todas esas libertades. Entonces, debería ser libre de redistribuir copias, tanto con o sin modificaciones, ya sea gratis o cobrando una tarifa por distribución, a cualquiera en cualquier parte. El ser libre de hacer estas cosas significa, entre otras cosas, que no tiene que pedir o pagar el permiso.

También debería tener la libertad de hacer modificaciones y usarlas en privado, en su propio trabajo u obra, sin siquiera mencionar que existen. Si publica sus cambios, no debería estar obligado a notificarlo a alguien en particular, o de alguna forma en particular.

La libertad de ejecutar el programa significa la libertad para cualquier tipo de persona u organización de usarlo en cualquier tipo de sistema de computación, para cualquier tipo de trabajo y propósito, sin estar obligado a comunicarlo a su programador, o alguna otra entidad específica. En esta libertad, el propósito de los usuarios es el que importa, no el propósito de los programadores. Como usuario es libre de ejecutar un programa para sus propósitos; y si lo distribuye a otra persona, también es libre para ejecutarlo para sus propósitos, pero usted no tiene derecho a imponerle sus propios propósitos.

12 *GNU – La definición de Software Libre*: http://www.gnu.org/philosophy/free-sw.es.html

La libertad de redistribuir copias debe incluir las formas binarias o ejecutables del programa, así como el código fuente; tanto para las versiones modificadas como para las no lo están. (Distribuir programas en forma de ejecutables es necesario para que los sistemas operativos libres se puedan instalar fácilmente). Resulta aceptable si no existe un modo de producir un formato binario o ejecutable para un programa específico, dado que algunos lenguajes no incorporan esa característica, pero debe tener la libertad de redistribuir dichos formatos si encontrara o programara una forma de hacerlo.

Para que la 1ª y 3ª libertad, para realizar cambios y publicar versiones mejoradas, tengan sentido; debe tener acceso al código fuente del programa. Por consiguiente, el acceso al código fuente es una condición necesaria para el software libre. El *código fuente ofuscado no es código fuente real*, y no cuenta como código fuente.

La 1ª libertad incluye la libertad de usar su versión modificada en lugar de la original. Una manera importante de modificar un programa es fusionando subrutinas y módulos libres disponibles. Si la licencia del programa dice que no puede fusionar un módulo existente con una debida licencia, así como si le requiere ser el titular de los derechos de autor de lo que agregue, entonces la licencia es demasiado restrictiva para calificarla como libre.

La 3ª libertad incluye la libertad de liberar sus versiones modificadas como software libre. Una licencia también puede permitir otras formas de relicenciarlas, en otras palabras, no tiene que ser una licencia de copyleft (el copyleft es un método general para hacer un programa libre, exigiendo que todas las versiones modificadas y extendidas del mismo sean también libres). No obstante, una licencia que requiera que las versiones modificadas no sean libres, no se puede considerar como una licencia libre.

Software Libre no significa “que no sea comercial”. Un programa libre debe estar disponible para el uso comercial, la programación comercial y la distribución comercial. La programación comercial de software libre ya no es inusual. Puede haber pagado dinero para obtener copias de

software libre, o puede haber obtenido copias sin costo. Pero sin tener en cuenta cómo obtuvo sus copias, siempre tiene la libertad de copiar y modificar el software, incluso de vender copias.

I.6.6. Aspectos relacionados con un Game Engine

Para comprender mejor la arquitectura de un Game Engine y sus características fundamentales se recomienda la lectura del libro *3D Game Engine Architecture* [13] , cuyo contenido introduce los siguientes temas:

- Core Systems
- Scene Graphs and Renderers
- Advanced Scene Graph Topics
- Advanced Rendering Topics
- Collision Detection
- Physics
- Applications
- Coding Conventions

Aclaración: La revisión de dicho libro es una recomendación para ampliar la información que se presenta en los capítulos siguientes y no una condición estricta para continuar la lectura de esta tesina.

13 David H. Eberly. *3D Game Engine Architecture* (1º edición).

II. DESARROLLO DE LA TESINA

CAPÍTULO 1: Game Engine o Motor de Videojuegos

En este capítulo se presentan las características fundamentales de dos motores de última generación: Unity3D y UDK. También se realiza la elección del Game Engine para llevar a cabo el desarrollo del videojuego propuesto en esta tesina.

1.1. Variedad de Motores

Existe actualmente una amplia gama de motores de videojuegos, con diferentes tipos de licencias y orientados a cumplir distintos tipos de propósitos. Se puede encontrar motores comerciales y gratuitos, con metodologías 2D o 3D, inclusive que brindan soluciones de juegos a variadas plataformas (Windows, Linux, android, etc.).

Existe un sitio web, ***Mod DB*** [14], que presenta una extensa lista de aproximadamente 247 motores, que pueden ser tomados en cuenta a la hora de desarrollar un videojuego. Desde la misma se puede ingresar a los detalles de información de cada Game Engine y observar algunos datos interesantes como: plataformas, compañía, sitio web oficial, licencia, valoración, estadísticas, screenshots, entre otros.

14 *Mod-DB*: http://www.moddb.com/engines

Uno de los propósitos de esta tesina, es seleccionar un Game Engine de última generación para examinar sus características principales y realizar un videojuego con dicho motor. Pero se debe tener en cuenta que, actualmente, las grandes empresas de videojuego trabajan con tecnologías de vanguardia, las cuales pueden costar miles de dólares. Afortunadamente existe una nueva tendencia por parte de algunas empresas que desarrollan motores de videojuegos, que los impulsa a colocar en el mercado versiones gratuitas (generalmente limitadas en algún aspecto) de sus herramientas para que las personas que deseen aprender, puedan hacerlo sin tener que comprar una versión full. Entre estas empresas podemos encontrar las conocidas ***Epic Games*** [15] y ***Unity Technologies*** [16], las cuales ofrecen versiones gratuitas y descargables de sus famosos motores de videojuegos.

1.1.1. UDK – Unreal Development Kit

UDK es un completo y profesional framework de desarrollo, lanzado por Epic Games en noviembre de 2009. Se basa en el ***Unreal Engine,*** que es el motor de juegos desarrollado por la misma compañía e implementado en 1998 en el shooter en primera persona Unreal.

15 *Epic Games*: http://www.epicgames.com/

16 *Unity Technologies*: http://unity3d.com/company/

Perfil	
Plataformas	PC, X360, PS3, iPhone, iPad
Compañía	Epic Games
Sitio Oficial	Udk.com
Licencia	UDK es gratuito para propósitos no comerciales (por ejemplo para fines educativos). No obstante existen licencias comerciales particulares y a nivel de compañía. Ver detalles en: http://www.udk.com/licensing
Valoración	9.5 (182 votos desde el sitio Mod DB)

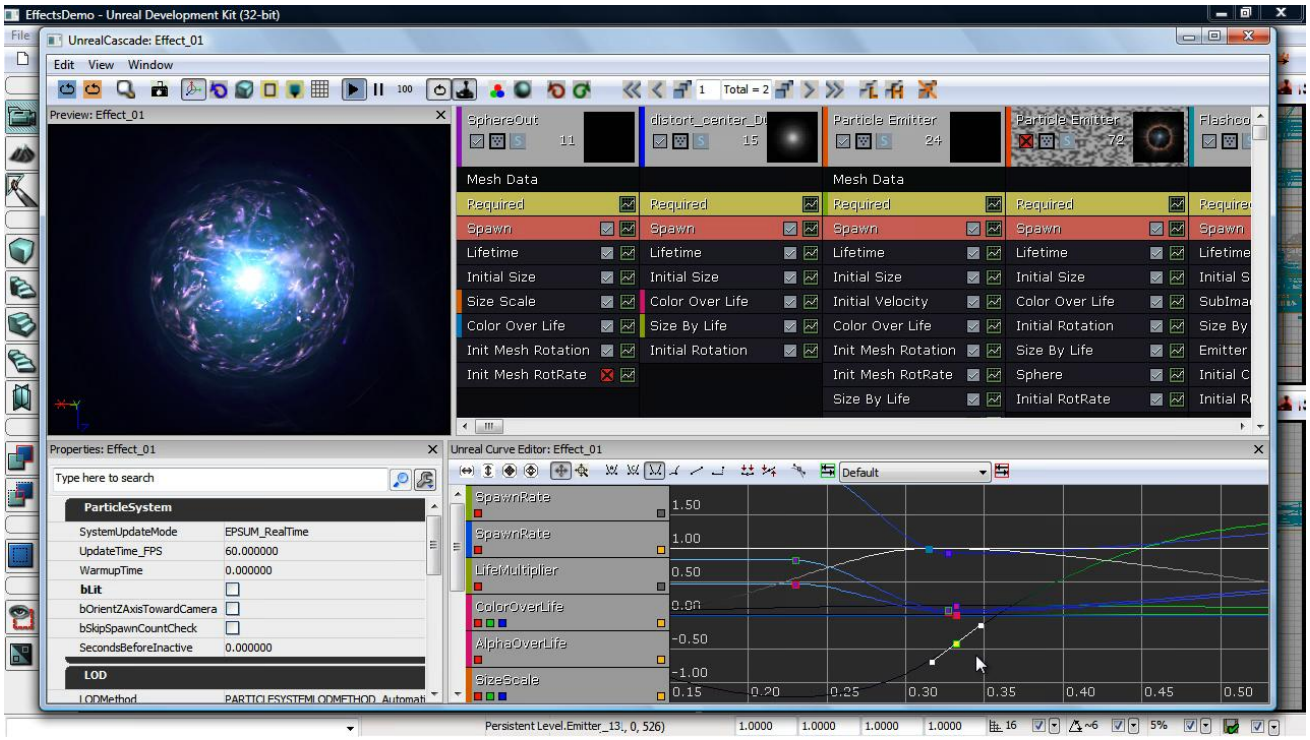
Características	
Completo entorno de edición	Unreal Engine 3 ofrece un entorno plenamente integrado de edición a través del famosos Unreal-Editor (todas las herramientas clave del motor son accesibles a través de él). El nuevo Unreal-Content-Browser permite gestionar activos como las mallas, materiales, sonidos y animaciones, más fácil que nunca. Ver detalles en: https://www.udk.com/features-editing.html
Poder de renderizado	UDK le trae Gemini, el sistema de renderizado multihilo. Gemini crea una realidad visual exuberante y proporciona la potencia necesaria para realizar simulaciones realistas. Se pueden obtener resultados impresionantes con este procesador flexible y altamente optimizado. UDK posee características de color de 64 bits HDR rendering pipeline, gamma-corrector, procesador de espacio de color lineal que proporciona una inmaculada precisión de color. Además soporta una amplia gama de efectos post-procesamiento, tales como el desenfoque de movimiento, la profundidad de campo, bloom, oclusión de ambiente y materiales definidos por el artista. Ver detalles en: https://www.udk.com/features-rendering.html
Animación avanzada	El sistema de animación de Unreal le da control sobre total sobre cada detalle, cada músculo y cada hueso. Haga su mundo más vibrante y

	añada más personalidad con animaciones detallistas. Ver detalles en: https://www.udk.com/features-animation.html
Scripting de gran alcance	UDK presenta a Unreal-Script, un sistema totalmente integrado de alto nivel, que implementa un lenguaje de programación orientado a objetos, y Unreal-Kismet, una solución avanzada de scripting visual. Estas herramientas permiten a cualquier persona crear un juego de gran alcance con un control prácticamente ilimitado. Ver detalles en: https://www.udk.com/features-scripting.html
Físicas del mundo real	UDK le da peso a todas sus creaciones a través del NVIDIA's PhysX System. Puede visualizarse una física intensa utilizando Unreal-PhAT y entornos destructibles que sumergirán a los jugadores. Ver detalles en: https://www.udk.com/features-physics.html
Iluminación y Sombras	Las funciones de iluminación y sombra más avanzadas, incluyendo la iluminación global de Unreal-Lightmass. Todas estas características son fácilmente añadidas a sus creaciones a través de Unreal-Editor. Ver detalles en: https://www.udk.com/features-lighting.html
Cinematografía	Unreal-Matinee inyecta escenas estilo de película en su juego, lo que permite colocar cada cámara, objeto y explosión dentro del mundo e incluso reproducirlo en tiempo real. Ver detalles en: https://www.udk.com/features-cinematics.html
Terreno	El sistema de terrenos de UDK le permite personalizar el lugar y la vegetación, las estructuras y los innumerables puntos de interés en el mundo de su juego. El terreno en UDK es fácil de editar y modificar para crear niveles y mundos. Ver detalles en: https://www.udk.com/features-terrain.html
Red	UDK proporciona una arquitectura de red flexible y de alto nivel adecuada para variados géneros de juegos, así como también para proyectos de simulación.

	Ver detalles en: https://www.udk.com/features-networking.html
Materiales y Shaders en tiempo real	UDK viene equipado con un sistema de materiales de gran alcance, con una interfaz visual que le permitirá crear Shaders en tiempo real y de arbitrariedad compleja sobre la marcha. Ver detalles en: https://www.udk.com/features-shaders.html
Audio	UDK está diseñado para ofrecer a los desarrolladores la libertad creativa y un control toal sobre el diseño de sonido. Ver detalles en: https://www.udk.com/features-audio.html
Efectos de Partículas	UDK permite crear efectos dinámicos de partículas, como lluvia o nieve, que dan vida a sus juegos. Ver detalles en: https://www.udk.com/features-particle-effects.html
Inteligencia Artificial	El sistema de Inteligencia Artificial ofrece dos rubustos sistemas para la navegación de IA. UDK soporta la creación automática y el uso de mallas de navegación, logrando que los todos los personajes sean controlados por la IA, incrementando la conciencia espacial de su entorno y la capacidad de tomar decisiones más inteligentes. Ver detalles en: https://www.udk.com/features-ai.html
Computación Distribuída	Unreal-Swarm es un sistema de trabajo de distribución escalable, optimizado para redes de alta velocidad en PC multi-núcleo. Ver detalles en: https://www.udk.com/features-swarm.html
Entornos destruibles	UDK le da un control total sobre la destrucción. Su herramienta de fractura permite crear mundos deformables interactivos. Además permite de crear todo tipo de entornos destruibles y objetos que se rompen como sucedería en la vida real. Ver detalles en: https://www.udk.com/features-destructible.html
Codec de video Bink	Bink comprime en una mejor calidad que DVD, aumentando hasta tres veces la velocidad de reproducción. Bink también utiliza menos memoria en tiempo de ejecución que otros codecs. Ver detalles en: https://www.udk.com/features-bink.html

Editor de vegetación	SpeedTree le permite generar vegetación en tiempo real, animación de árboles con el viento, exuberantes selvas, bosques espesos y mucho más. Ver detalles en: https://www.udk.com/features-speedtree.html
Animación facial	El sistema FaceFX permite a los desarrolladores crear animaciones realistas a partir de archivos de audio. Ver detalles en: https://www.udk.com/features-facefx.html

Imágenes:



Requerimientos mínimos:

- Windows XP SP2 or Windows Vista
- 2.0+ GHz processor
- 2 GB system RAM
- SM3-compatible video card
- 3 GB Free hard drive space

Requerimientos recomendados:

- Windows 7 64-bit
- 2.0+ GHz multi-core processor
- 8 GB System RAM
- NVIDIA 8000 series or higher graphics card
- Plenty of HDD space

Requerimientos mínimos para desarrolladores DX11:

- Windows Vista
- 2.0+ GHz processor
- 2 GB system RAM
- DX11 Graphics Card:
- Nvidia: 400 series or above
- ATI: 5000 series or above
- 3 GB Free hard drive space

Para concluír la presentación de UDK, se ha realizado una recopilación de contenido multimedia que se encuentra en la ruta **/multimedia/udk** del DVD adjunto con esta tesina.

Toda la información sobre UDK fue extraída, adaptada y traducida desde el sitio oficial [17].

1.1.2. Unity3D

Unity es una herramienta integrada de desarrollo para la creación de videojuegos u otro contenido interactivo como visualizaciones arquitectónicas o animaciones 3D en tiempo real.

Unity ganó el Wall Street Journal 2010 Premio a la Innovación Tecnológica en la categoría de

17 Sitio oficial de UDK: http://www.udk.com

software [18]. En 2009, Unity Techologies fue nombrada por Gamasutra como una de "Las 5 mejores empresas de videojuegos de 2009" [19].

Perfil	
Plataformas	PC, X360, Wii, PS3, Linux, Mac,iPhone, iPad, Android, Web (plugin)
Compañía	Unity Technologies
Sitio Oficial	Unity3d.com
Licencia	Existen dos tipos de licencias principales: Unity y Unity-Pro. La primera es gratuita (con algunas limitaciones) y está orientada a estudiantes y personas que deseen aprender a trabajar con el game engine. La versión Pro no es gratuita, pero tiene características adicionales. Tanto Unity como Unity-Pro incluyen el entorno de desarrollo, tutoriales, ejemplos de proyectos y el contenido, el apoyo a través de la comunidad de foros, wiki, y las futuras actualizaciones de la versión. Ver detalles en: http://unity3d.com/unity/licenses
Valoración	8.9 (153 votos desde el sitio Mod DB)

Características	
Entorno integrado de desarrollo - Editor	Entorno integrado de desarrollo con jerarquía, edición visual, inspector de propiedades y vista previa del videojuego. Ver detalles en: http://unity3d.com/unity/editor/
Publicación	Soporte para publicación en múltiples plataformas (PC, X360, Wii, PS3, Linux, Mac, iPhone, iPad, Android). Ver detalles en: http://unity3d.com/unity/features/deployment
Activos (Assets)	Los activos cargados en Unity se importan de forma automática y se

18 "*The WSJ Technology Innovation Award Winners, Category by Category*": The Wall Street Journal. http://online.wsj.com/article/SB10001424052748703904304575497473735761294.html

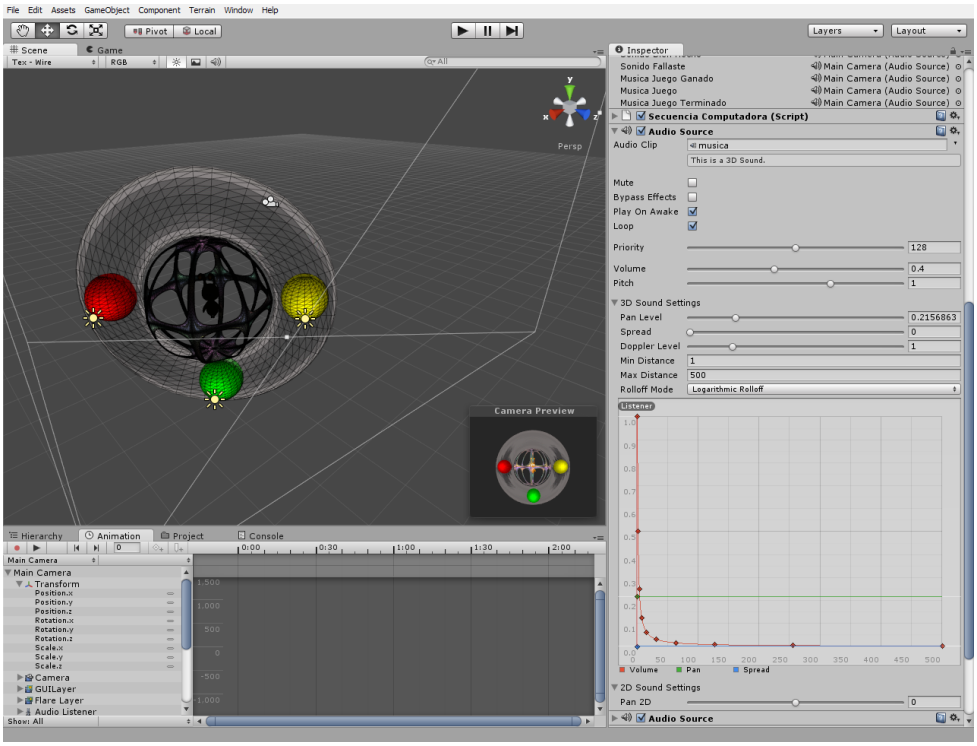
19 "*Gamasutra's Best of 2009: Top 5 Game Companies*": Gamasutra. http://www.gamasutra.com/view/news/26243/Gamasutras_Best_Of_2009_Top_5_Game_Companies.php

	vuelven a importar si el archivo se actualiza. Unity soporta la integración con 3ds Max, Maya, Blender, ZBrush, Cinema 4D, Cheetah 3D, entre otros. Ver detalles en: http://unity3d.com/unity/features/asset-importing
Efectos	Soporte para "bump mapping", "reflection mapping", "parallax mapping", "screen space ambient occlusion", sombras dinámicas usando "shadow maps", "render to texture" y el post procesamiento de efectos en pantalla completa.
Shaders	El Unity-Shader-System combina facilidad de uso, flexibilidad y rendimiento. Todos los Shaders incorporados se integran perfectamente con cualquier tipo de luz, con o sin cookies. Además puede escribir sus propios Shaders con el potente lenguaje de Unity: ShaderLab con Cg y GLSL. Ver detalles en: http://unity3d.com/unity/features/shaders
Gráficos	Unity cuenta con una fuente altamente optimizada de gráficos para DirectX y OpenGL. Mallas de animación, sistemas de partículas y un sistema avanzado de iluminación y sombras. Ver detalles en: http://unity3d.com/unity/features/graphics
Terreno	Puede crear increíbles paisajes con densa vegetación que funcionarán sin problemas en hardware de gama baja, además de ocupar muy poco espacio en disco. Ver detalles en: http://unity3d.com/unity/features/terrains
Red	Agregue funcionalidades multijugador como chat y puntuaciones en línea. Unity brinda solución a todas las necesidades de red para sus videojuegos. Ver detalles en: http://unity3d.com/unity/features/networking
Físicas avanzadas	Llena de vida a los objetos del juego con las funcionalidades provistas por el motor de NVIDIA's PhysX. Ver detalles en: http://unity3d.com/unity/features/physics

Audio y Video	Mezcla en tiempo real gráficos en 3D con streaming de audio y vídeo. Ver detalles en: http://unity3d.com/unity/features/audio-and-video
Scripting	Unity soporta tres lenguajes de Scripting: JavaScript, C#, y una versión modificada de Python llamada Boo. Los tres son igual de rápidos e interoperables. Además pueden utilizar librerías .NET con soporte a base de datos, expresiones regulares, XML, acceso a ficheros y funciones de red. El Scripting frecuentemente es considerado limitado y lento. Pero en Unity los scripts son compilados a código nativo y se ejecuta casi tan rápido como C++. Ver detalles en: http://unity3d.com/unity/features/scripting
Iluminación y Sombras	Unity ofrece un sistema de iluminación altamente optimizado con lightmaps y sombras de tiempo real. Ver detalles en: http://unity3d.com/unity/features/shadow-and-light
	Existe una gran cantidad de conocimiento y soporte que puede aprovechar. Desde guías de usuario con referencia de Scripting hasta tutoriales en PDF. Además, existe una comunidad activa y abierta de foros, IRC y Wiki, que le dará las respuestas que necesita en cualquier momento. La documentación en español también es muy buena y variada. Ver detalles en: http://unity3d.com/unity/features/documentation
Motor de Renderizado	Para la versión 3 de Unity se reescribió la prestación de renderizado, logrando más flexibilidad y potencia. Esto significa que la prestación es en general un 50% más rápido que antes. Ver detalles en: http://unity3d.com/unity/engine/rendering

Imágenes:


Bar


File Edit Assets GameObject Component Terrain Window Help
Pivot Local
Layers Layout
Scene
Game
Persp
Inspector
Sonido Fallaste
Musica Juego Ganado
Musica Juego
Musica Juego Terminado
Main Camera (Audio Source)
Secuencia Computadora (Script)
Audio Source
Audio Clip
musica
This is a 3D Sound.
Mute
Bypass Effects
Play On Awake
Loop
Priority
128
Volume
0.4
Pitch
1
3D Sound Settings
Pan Level
0.2156863
Spread
0
Doppler Level
1
Min Distance
1
Max Distance
500
Rolloff Mode
Logarithmic Rolloff
Listener
Camera Preview
Hierarchy
Animation
Project
Console
Main Camera
Transform
Camera
GUILayer
Flare Layer
Audio Listener
Volume
Pan
Spread
2D Sound Settings
Pan 2D
Audio Source

Requerimientos Mínimos:

- Windows: XP SP2 or later; Mac OS X: Intel CPU & "Leopard" 10.5 or later. Note that Unity was not tested on server versions of Windows and OS X.
- Graphics card with 64 MB of VRAM and pixel shaders or 4 texture units. Any card made in this millennium should work.
- Using Occlusion Culling requires GPU with Occlusion Query support (some Intel GPUs do not support that).
- The rest only depends on the complexity of your projects!.

La lista completa de requerimientos se puede ver en: http://unity3d.com/unity/system-requirements

Para concluír la presentación de Unity3D, se ha realizado una recopilación de contenido multimedia que se encuentra en la ruta **/multimedia/unity** del DVD adjunto con esta tesina.

Toda la información de Unity3D fue extraída, traducida y adaptada desde el sitio oficial [20].

1.2. Selección del Motor

Unity y UDK poseen ciertas características que los hacen especiales para el desarrollo del videojuego propuesto para esta tesina.

20 Sitio oficial de Unity3D: http://www.unity3d.com

Característica (Game Engine)	Game Engine	
	Unity	UDK
Versión gratuita	Sí	Sí
Documentación oficial	En inglés. Muy completa	En inglés. Muy completa
Documentación en español	Buena documentación en español. Comunidades hispanohablantes.	Buena documentación en español. Comunidades hispanohablantes.
Requerimientos mínimos de Hardware	Aceptable para máquinas de gama baja, media y alta (aunque también depende mucho del proyecto que se desarrolle)	Aceptable para máquinas de gama media y alta (aunque también depende mucho del proyecto que se desarrolle)
Tecnología de vanguardia	Sí. Unity es un motor de última generación que cubre prácticamente todos los aspectos del desarrollo de un videojuego.	Sí. UDK es un motor de última generación que cubre prácticamente todos los aspectos del desarrollo de un videojuego.
Alcance	Alto. Unity es una tecnología nueva que crece día a día de una forma asombrosa, tanto el software en sí mismo como las comunidades que lo llevan al progreso.	Muy Alto. UDK posee una trayectoria impecable en el mundo del desarrollo de videojuegos, los proyectos profesionales creados con esta herramienta lo demuestran.
Mecanismo	Herramientas visuales y programación vía Scripts.	Herramientas visuales y programación vía Scripts.
Multiplataforma	Sí: PC, X360, Wii, PS3, Linux, Mac, iPhone, iPad, Android, Web (with plugin)	Sí: PC, X360, PS3, iPhone, iPad
Compatibilidad con programas externos	Alta	Media

Sin embargo existen otros motores que poseen algunas de estas características, como por ejemplo el ***CryEngine*** [21], quien presenta suficientes razones como para ser tomado en cuenta, aunque la ausencia de una comunidad hispanohablante y documentación en español hace que el motor sea apartado de la lista de posibles candidatos. Otros motores simplemente no tienen la potencia o el alcance de los mencionados anteriormente; muchos no son multiplataforma y no tienen versiones gratuitas para descargar. Además debemos tener en cuenta que la selección sólo incluye a aquellos basados en motores gráficos de gran potencia y que cuentan con una gran variedad de herramientas intrínsecas (es decir, dentro del mismo paquete de software) que permiten la realización de todos los aspectos del videojuego. A esto hay que sumarle la compatibilidad con otros programas de diseño y edición gráfica 2D, 3D y sonido.

En el capítulo anterior se presentaron algunas características generales (premisas para el desarrollo) que debía tener el juego, y por lo tanto, estas características deben ser soportadas también por el Game Engine. En el cuadro siguiente se muestran las cualidades generales del juego y la compatibilidad con los motores propuestos.

Característica (Videojuego)	Compatibilidad con el Game Engine	
	Unity	UDK
El videojuego debe ser del género "***Serious Game***".	Soportado: Su alcance le permite desarrollar juegos de múltiples géneros.	Soportado: Su alcance le permite desarrollar juegos de múltiples géneros.
El videojuego debe tener elementos 2D y 3D.	Soportado: Unity se basa en un motor gráfico 3D. No obstante, esto no lo limita a trabajar sólo en 3D, las interfaces de usuario y otros elementos pueden estar definidos en 2D.	Soportado: UDK se basa en un motor gráfico 3D. No obstante, esto no lo limita a trabajar sólo en 3D, las interfaces de usuario y otros elementos pueden estar definidos en 2D.

21 CryEngine: http://mycryengine.com/

El videojuego debe ser lo más completo posible, en términos de multimedia e interactividad.	Soportado: Unity posee una gran variedad de herramientas de multimedia e interactividad que le dan vida a los videojuegos desarrollados con dicho motor.	Soportado: UDK posee una gran variedad de herramientas de multimedia e interactividad que le dan vida a los videojuegos desarrollados con dicho motor.
El videojuego no debe contener violencia.	Soportado: esta característica es propia del videojuego y no influye en la selección del game engine.	Soportado: esta característica es propia del videojuego y no influye en la selección del game engine.

Como se puede observar, los motores propuestos cumplen ampliamente las expectativas. Pero para llevar a cabo este proyecto sólo debe seleccionarse uno y afortunadamente, existen dos características que aclaran el proceso de selección.

Tanto Unity como UDK son multiplataforma, pero Unity tiene una ventaja que podría ser explotada en el marco de esta tesina o para futuros proyectos, ya que permite publicar los juegos en un navegador web, a través de un plugin especial creado para tal propósito. Esto significa que se puede desarrollar un videojuego y posteriormente colocarlo en un sitio web para que los usuarios lo prueben de forma online. Esto es similar a lo que se puede hacer con la tecnología **Flash** y su reproductor **Flash Player** [22], pero con la solidez interna de un motor de videojuegos de última generación. El plugin es totalmente gratuito y puede descargarse desde el sitio web de Unity.

Otra característica que hace a Unity especial para llevar a cabo este proyecto, es que tiene menos requisitos de hardware que UDK.

Motor seleccionado: *Unity3D*.

22 Flash Player: http://get.adobe.com/es/flashplayer/

CAPÍTULO 2: Programas Externos

En este capítulo se propone la utilización de distintas aplicaciones de Software Libre, externas al Game Engine, que permiten llevar adelante las etapas y procesos creativos en el desarrollo del videojuego. La decisión de utilizar Software Libre se debe a la existencia de una filosofía bien aceptada mundialmente, que apoya el uso de este tipo de programas y que ofrece la libertad de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Además, se presenta para cada caso, una alternativa de software propietario.

2.1. Gráfica 2D

2.1.1. Gimp

GIMP [23] (GNU Image Manipulation Program) es un programa de edición o manipulación de imágenes digitales. Sirve también para retoque fotográfico, composición y creación de imágenes. Además es libre y gratuito. Forma parte del proyecto *GNU* [24] y está disponible bajo la *Licencia pública*

23 Página oficial de GIMP: http://www.gimp.org/

24 GNU: http://www.gnu.org/home.es.html

general de GNU[25].

Algunas de sus funciones principales son las siguientes:

- Customizable Interface (interfaz personalizable)
- Photo Enhancement (mejora de fotografías)
- Digital Retouching (retoque digital)
- Hardware Support (soporte de dispositivos hardware como *tablets* o controladores MIDI)
- File Formats (soporte a múltiples formatos de archivo como TIFF, JPEG, GIF, PNG, PSD, BMP, entre otros)
- Supported Plataforms (soporte a varias plataformas como GNU/Linux, Microsoft Windows XP/Vista, Mac OS X, Sun OpenSolaris, FreeBSD)

2.1.2. Adobe Photoshop CS5

Como alternativa al software libre GIMP, se aconseja la utilización del software propietario Adobe Photoshop CS5.

25 Licencia pública general de GNU: http://www.gnu.org/copyleft/gpl.html

Adobe Photoshop CS5 [26] es el estándar de la industria de software de edición de imágenes, que se utiliza en todo el mundo por fotógrafos profesionales, fotógrafos aficionados, y los diseñadores que desean perfeccionar sus imágenes digitales.

Algunas de sus funciones principales son las siguientes:

- **Acceso fácil a las funciones de edición claves**

  Obtenga solo las herramientas que necesita en el momento preciso cuando esté llevando a cabo actividades de edición habituales. Los paneles de ajustes y máscaras le guían para que realice ediciones precisas.

- **Herramientas creativas extraordinarias**

  Consiga efectos pictóricos realistas con la función de mezcla de colores en lienzos y cree trazos de pincel naturales. Fusione múltiples exposiciones en imágenes de alto rango dinámico (HDR) sorprendentes. Mueva, elimine, deforme o estire los elementos de las imágenes.

- **Fotografía de última generación**

  Combine fácilmente múltiples exposiciones para ampliar el rango dinámico con más impacto, precisión y fidelidad que nunca. Convierta a blanco y negro con métodos nuevos. Utilice las herramientas Sobreexponer, Subexponer y Esponja para conservar de forma inteligente los detalles de colores y tonos.

26 Página oficial de Adobe Photoshop CS5: http://www.adobe.com/es/products/photoshop.html

- **Procesamiento de *imágenes sin procesar***

  Disfrute de resultados superiores cuando convierta imágenes sin procesar con el plugin de Camera Raw 6, que ofrece compatibilidad con más de 275 modelos de cámaras además de opciones de edición no destructiva, de modo que pueda experimentar sin dañar la foto original.

- **Herramientas de composición automáticas**

  Cree fácilmente una sola imagen de una serie de fotos con distinto punto focal, consiguiendo una fusión de color y sombras suave y la extensión de la profundidad de campo. Y aproveche la alineación de capas automática y precisa.

- **Integración con Adobe Photoshop Lightroom**

  Disfrute de la integración completa con Photoshop Lightroom (se vende por separado), que le permitirá desplazar archivos de Lightroom a Photoshop CS5 para crear imágenes de HDR, imágenes panorámicas y documentos de Photoshop de varias capas.

- **Gestión eficaz del contenido multimedia**

  Gestione el contenido multimedia de forma visual con el software de Adobe Bridge CS5.

Disfrute de las posibilidades de cambiar los nombres de los lotes de forma más flexible y de acceder a archivos de su trabajo con el panel personalizable Adobe Mini Bridge.

- **Opciones de impresión eficaces**

  Realice impresiones sobresalientes de forma más rápida con la automatización, las secuencias de comandos y un cuadro de diálogo de impresión de más fácil navegación.

- **Amplia compatibilidad de formatos de archivo**

  Importe y exporte cientos de formatos de imagen, como PSD, AI, PDF, NEF, CRW, TIFF, BMP, Cineon, JPEG, JPEG2000, FXG, OpenEXR, PNG y Targa, así como formatos de vídeo, como 3G, FLC, MOV, AVI, DV Stream, Image Sequence, MPEG-4 y FLV.

- **Capacidad de ampliación**

  Personalice Photoshop y amplíe sus capacidades con paneles personalizables creados por los desarrolladores para realizar tareas específicas, visite Adobe Photoshop Marketplace para acceder a plugins y recursos de terceros, y la Ayuda de la Comunidad de Adobe para consultar consejos y trucos.

Para mayor información sobre las funcionalidades de Adobe Photoshop CS5 se puede acceder a la *sección de funciones* [27] de la página oficial.

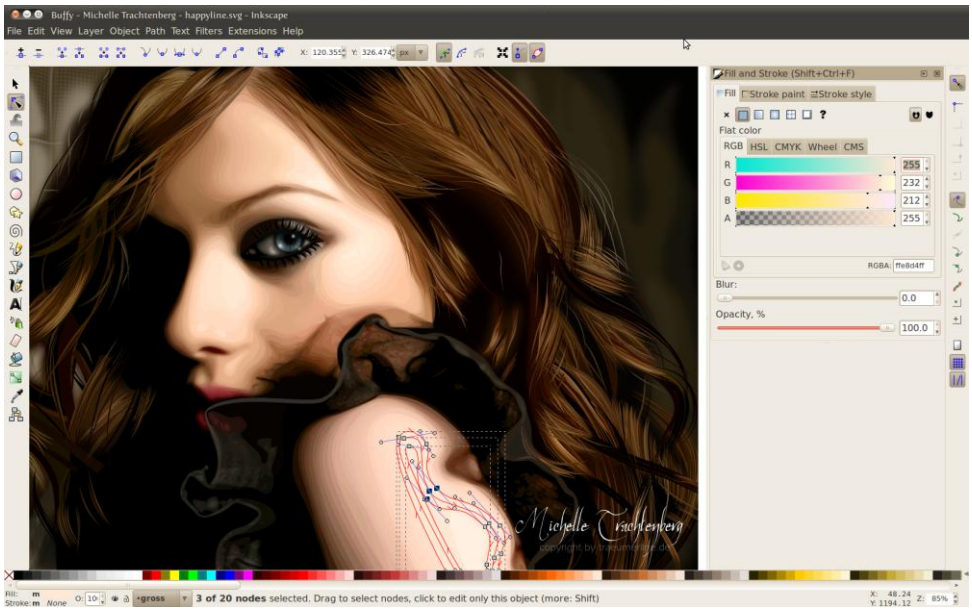
2.1.3. Inkscape

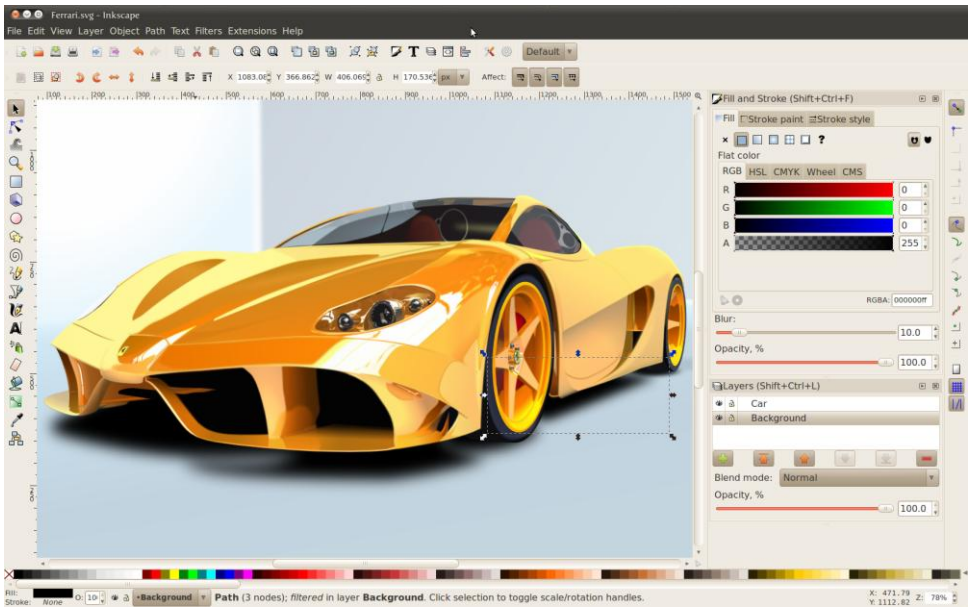
Inkscape[28] es un editor de gráficos vectoriales de código abierto, con capacidades similares a Illustrator, Freehand, CorelDraw o Xara X, usando el estándar de la W3C: el formato de archivo Scalable Vector Graphics (SVG). Las características soportadas incluyen: formas, trazos, texto, marcadores, clones, mezclas de canales alfa, transformaciones, gradientes, patrones y agrupamientos. Inkscape también soporta meta-datos Creative Commons, edición de nodos, capas, operaciones complejas con trazos, vectorización de archivos gráficos, texto en trazos, alineación de textos, edición de XML directo y mucho más. Puede importar formatos como Postscript, EPS, JPEG, PNG, y TIFF y exporta PNG asi como muchos formatos basados en vectores.

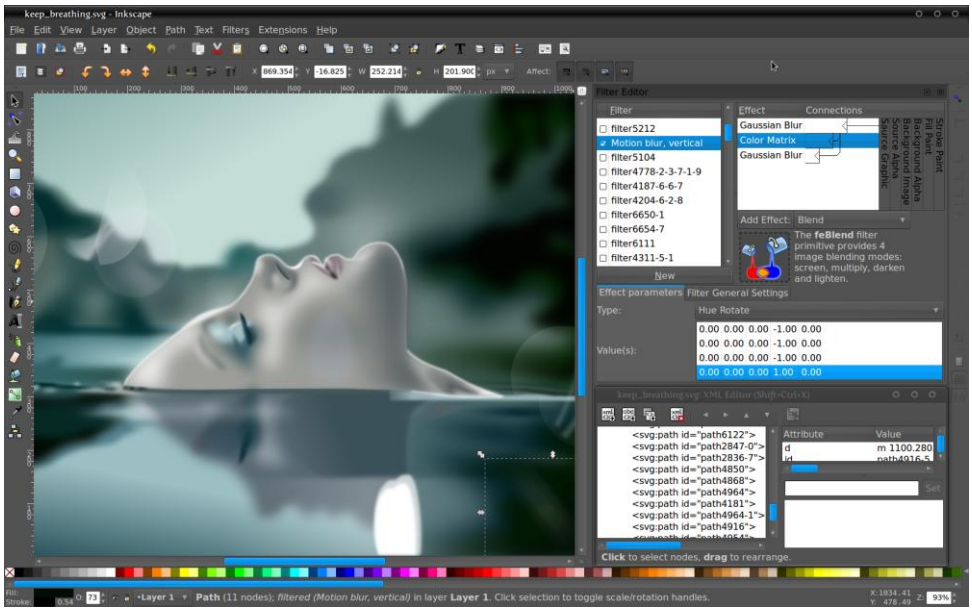
Capturas de pantalla del programa:

27 Funciones de Adobe Photoshop CS5: http://www.adobe.com/es/products/photoshop/features.html

28 Página oficial de Inkscape: http://inkscape.org/index.php?lang=es


Buffy - Michelle Trachtenberg - happyline.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
Fill and Stroke (Shift+Ctrl+F)
Fill Stroke paint Stroke style
Flat color
RGB HSL CMYK Wheel CMS
R 255
G 232
B 212
A 255
Blur: 0.0
Opacity, % 100.0
Michelle Trachtenberg
3 of 20 nodes selected. Drag to select nodes, click to edit only this object (more: Shift)


Ferrari.svg - Inkscape
File Edit View Layer Object Path Text Filters Extensions Help
Fill and Stroke (Shift+Ctrl+F)
Fill Stroke paint Stroke style
Flat color
RGB HSL CMYK Wheel CMS
Blur: 10.0
Opacity, % 100.0
Layers (Shift+Ctrl+L)
Car
Background
Blend mode: Normal
Opacity, % 100.0
Path (3 nodes); filtered in layer Background. Click selection to toggle scale/rotation handles.



2.1.4. Adobe Illustrator CS5

Como alternativa al software libre Inkscape se presenta el software propietario **Adobe Illustrator CS5**[29].

Adobe Illustrator CS5 le permite crear gráficos y el tipo de formato vectorial. Los gráficos vectoriales son independientes de la resolución, lo que significa que puede cambiar el tamaño de un gráfico de una tarjeta de visita a una valla publicitaria y que mantendrá su alta calidad. Illustrator

29 Página oficial de Adobe Illustrator CS5: http://www.adobe.com/es/products/illustrator.html

también te permite crear activos que pueden ser fácilmente distribuidos en diferentes medios: impresos, Web, interactivo, video y móviles.

Algunas de sus funciones principales son las siguientes:

- **Herramientas sofisticadas de dibujo vectorial**

  Cree diseños distinguidos con herramientas precisas de creación de formas, con pinceles pictóricos y fluidos así como con controles de ruta avanzados.

- **Tipografía avanzada**

  Diseño de texto para cualquier medio con controles profesionales para estilos de párrafo y carácter, apoyo a OpenType ®, efectos transparentes, y mucho más.

- **Degradados y transparencia**

  Interactúe con degradados directamente en los objetos y controle la transparencia de colores individuales en degradados y mallas de degradado.

- **Dibujo con perspectiva**

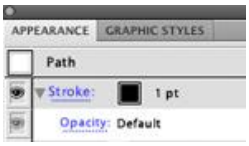
  Utilice cuadrículas de perspectiva para dibujar formas y escenas con una perspectiva linear precisa de 1, 2 o 3 puntos, y crear una apariencia realista de profundidad y distancia.

- **Múltiples mesas de trabajo**

  Trabaje en hasta 100 mesas de trabajo de distintos tamaños en un solo archivo, que están organizadas, tienen nombre y se visualizan del modo que desee.

- **Edición directa en el panel Apariencia**

  Edite directamente las características del objeto en el panel Apariencia, y elimine la necesidad de abrir los paneles Relleno, Trazo o Efectos.



- **Mejoras de dibujo**

  Dibuje por detrás, ignorando el orden de apilamiento. Dibuje o coloque una imagen en el interior, y se creará de forma instantánea una máscara de recorte.

- **Integración con Adobe CS Review**

  Cree y comparta revisiones en línea para clientes en su oficina o en el otro lado del mundo con Adobe CS Review, un servicio en línea de CS Live.

- **Formatos de archivos gráficos estándares del sector**

  Trabaje con prácticamente cualquier tipo de archivo gráfico: PDF, EPS, FXG, Photoshop (PSD), TIFF, GIF, JPEG, SWF, SVG, DWG, DXF, y muchos más.

- **Gráficos y textos nítidos para diseños web y dispositivos móviles**

  Cree y alinee objetos vectoriales de forma precisa en la cuadrícula de píxeles del archivo para obtener gráficos de trama nítidos y limpios. Utilice las opciones de suavizado de texto para crear marcos de texto individuales.

Para mayor información sobre las funcionalidades de Adobe Illustrator CS5 se puede acceder a la *sección de funciones* [30] de la página oficial.

30 Funciones de Adobe Illustrator CS5: http://www.adobe.com/es/products/illustrator/features.html

2.2. Gráfica 3D

2.2.1. Blender

Blender [31] es la suite de creación de contenidos 3D gratuita y de código abierto, disponible para los principales sistemas operativos bajo la *Licencia pública general de GNU*.

Algunas funciones son las siguientes:

Interfaz

- Ventana flexible y totalmente configurable. Múltiples configuraciones de pantalla.
- Deshacer acciones en todos los niveles.
- Fuentes anti-alias con soporte de traducción internacional.
- Editor de scripts Python.
- Interfaz gráfica de usuario para los scripts de Python.
- Interfaz coherente con todas las plataformas.

Rigging

- Modo de creación rápido de esqueleto.
- Pintura interactiva 3D.
- Rápida envoltura basada en pieles.
- Skinning automático.

31 Sitio oficial de Blender: http://www.blender.org/

Animación

- Animación de personajes y editor de poses.
- Sistema de restricciones de animación.
- Reproducción de audio, mezcla y soporte para la edición de sincronización de sonido.
- Línea de tiempo que ofrece acceso rápido a múltiples funciones de reproducción.
- Lenguaje Python para animaciones personalizadas y de procedimiento.

Plataformas soportadas

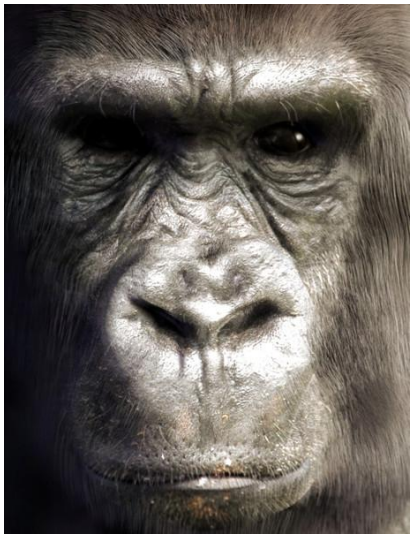
- Windows XP, Vista, 7.
- Mac OS X.
- Linux .
- FreeBSD.
- Blender funciona en sistemas de 32 y 64 bits.

Para obtener una descripción técnica exhaustiva de todas las funcionalidades de Blender, se puede acceder a la *sección de funciones* [32] de la página oficial.

Imágenes de ejemplo:

32 Funciones de Blender: http://www.blender.org/features-gallery/features/

Gorilla by Everett Gunther



Steampunk Nemean Lion by Kevin Hays

Cosmic Girl by Jiro Sugiyama

Sign of the Juggernaught by Derek Watts

La galería de imágenes completa [33] se encuentra en el sitio web oficial de Blender.

33 Galería Blender: http://www.blender.org/features-gallery/gallery/art-gallery/

2.2.2. Autodesk 3ds Max

Como alternativa al programa de software libre Blender, se presenta el software propietario **Autodesk 3ds Max** [34].

Autodesk 3ds Max es un poderoso paquete de modelado 3D, confiable y accesible, que provee funciones de animación, renderizado y composición. También es un favorito entre los animadores de personajes y artistas de videojuegos. 3ds Max ofrece:

- **Núcleo de gráficos ultra acelerado**: Incrementos drásticos del rendimiento y la calidad visual en la ventana gráfica.
- **Texturas procedimentales básicas**: 80 texturas procedimentales básicas nuevas para conseguir una gran variedad de acabados.
- **Dinámica de cuerpos rígidos mRigid**: El motor NVIDIA® PhysX® le permite crear simulaciones dinámicas de cuerpos rígidos directamente en la ventana gráfica.
- **Renderizador iray**: Resultados casi fotorrealistas más previsibles sin preocuparse por los parámetros de renderización.
- **Mejoras de desajuste UVW**: Un nuevo método de mapeado para crear mapas UVW mejores en menos tiempo.
- **Mejoras de pintura y escultura**: Los nuevos pinceles de conformado, transformación y

34 Página oficial de Autodesk 3ds Max: http://usa.autodesk.com/3ds-max/

restricción a spline aumentan el control sobre las pinceladas y sus efectos.

Imágenes de ejemplo:

Capcom Co., Ltd.

Ubisoft Montreal: Assassin's Creed: Brotherhood

2.3. Edición de Sonido

2.3.1. Audacity

Audacity [35] es un editor de audio libre, fácil de usar y multilingüe para Windows, Mac OS X, GNU/Linux y otros sistemas operativos. Puede usar Audacity para:

35 Sitio oficial de Audacity: http://audacity.sourceforge.net/

- Grabar audio en vivo.
- Convertir cintas y grabaciones a sonido digital o CD.
- Editar archivos Ogg Vorbis, MP3, WAV y AIFF.
- Cortar, copiar, unir y mezclar sonidos.
- Cambiar la velocidad o el tono de una grabación.
- Y mucho más.

Algunas de sus funciones principales son las siguientes:

- **Grabar**

Audacity puede grabar audio en directo usando un micrófono o un mezclador, así como digitalizar grabaciones de cintas de cassette, discos de vinilo y minidiscs. Con algunas tarjetas de sonido puede incluso capturar streaming de audio.

- ✓ Grabar de un micrófono, la línea de entrada u otras fuentes.
- ✓ Crear pistas sobre otras ya existentes para crear grabaciones multipista.
- ✓ Grabar hasta 16 canales a la vez (requiere hardware multicanal).
- ✓ Los medidores de niveles pueden monitorizar el volumen antes, durante y después de la grabación.

- **Importar y exportar**

Importar archivos de sonido, editarlos y combinarlos con otros archivos o nuevas grabaciones. Exportar grabaciones en varios formatos de sonido.

- ✓ Importar y exportar archivos WAV, AIFF, AU y Ogg Vorbis.
- ✓ Importar sonido en formato MPEG (incluyendo archivos MP2 y MP3) con libmad.
- ✓ Exportar MP3s con el codificador opcional LAME.
- ✓ Crear archivos WAV o AIFF para almacenarlos en CD de sonido.

- ✓ Importar y exportar todos los formatos compatibles con libsndfile.
- ✓ Abrir archivos de sonido "crudos" (sin cabeceras) mediante el comando "Importar Raw".

- **Edición**
  - ✓ Edición sencilla mediante cortar, copiar, pegar y borrar.
  - ✓ Utiliza ilimitados niveles de deshacer (y rehacer) para volver a cualquier estado anterior.
  - ✓ Rápida edición de archivos grandes.
  - ✓ Edita y mezcla un número ilimitado de pistas.
  - ✓ Utiliza la herramienta de dibujo para alterar cada punto de muestro.
  - ✓ Desvanece el sonido suavemente con la herramienta "envolvente".

- **Efectos**
  - ✓ Cambiar el tono sin alterar el tempo y viceversa.
  - ✓ Eliminar ruidos estáticos, silbidos, tarareos u otros ruidos de fondo constantes.
  - ✓ Alterar las frecuencias con la ecualización, filtros FFT y amplificar los bajos.
  - ✓ Ajustar los volúmenes con el compresor, amplificar y normalizar los efectos.

- **Otros efectos incluidos**
  - ✓ Eco
  - ✓ Fase
  - ✓ Wahwah
  - ✓ Inversión

- **Calidad de sonido**
  - ✓ Graba y edita muestras de 16 bits, 24 bits y 32 bits (de coma flotante).
  - ✓ Graba hasta un máximo de 96 kHz.
  - ✓ Las frecuencias de muestreo y formatos son convertidos mediante un proceso de alta calidad.
  - ✓ Mezcla pistas con diferentes frecuencias de muestreo o formatos y Audacity los

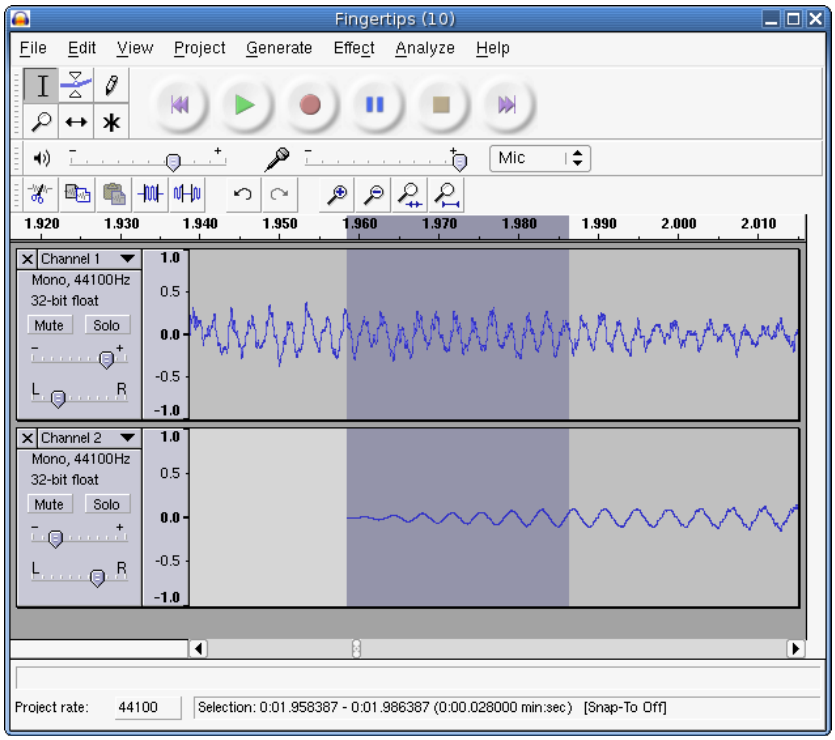
convertirá automáticamente en tiempo real.

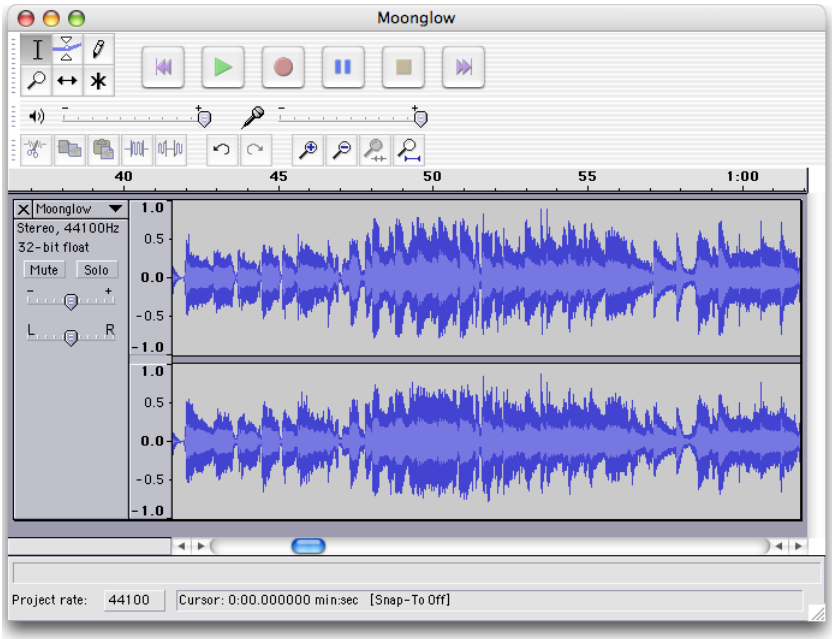
- **Plug-ins**
  - ✓ Añada nuevos efectos con los plug-ins LADSPA.
  - ✓ Audacity incluye algunos plug-ins de ejemplo creados por Steve Harris.
  - ✓ Cargar plug-ins VST para Windows y Mac, con el Activador VST (opcional).
  - ✓ Crear nuevos efectos con el lenguaje de programación Nyquist incluido.

- **Análisis**
  - ✓ Modo espectrógrafo para visualizar frecuencias.
  - ✓ Comando "Análisis de espectro" para obtener un análisis detallado de las frencuencias.

- **Libre y multi-plataforma**
  - ✓ Con una Licencia Pública General de GNU (GPL).
  - ✓ Corre bajo Mac OS X, Windows y GNU/Linux.

Capturas de pantalla (página siguiente):


Fingertips (10)
File Edit View Project Generate Effect Analyze Help
Mic
1.920 1.930 1.940 1.950 1.960 1.970 1.980 1.990 2.000 2.010
Channel 1
Mono, 44100Hz
32-bit float
Mute Solo
L R
1.0 0.5 0.0 -0.5 -1.0
Channel 2
Mono, 44100Hz
32-bit float
Mute Solo
L R
1.0 0.5 0.0 -0.5 -1.0
Project rate: 44100
Selection: 0:01.958387 - 0:01.986387 (0:00.028000 min:sec) [Snap-To Off]


Moonglow
40 45 50 55 1:00
Moonglow
Stereo, 44100Hz
32-bit float
Mute Solo
L R
1.0 0.5 0.0 -0.5 -1.0
1.0 0.5 0.0 -0.5 -1.0
Project rate: 44100
Cursor: 0:00.000000 min:sec [Snap-To Off]

2.3.2. Adobe Audition CS5

Como alternativa al programa de software libre Audacity se presenta el software propietario **Adobe Audition CS5** [36].

El software **Adobe Audition CS5** proporciona las herramientas profesionales que necesita para dotar del mejor sonido a sus producciones de vídeo y audio. Gestione de forma eficaz una amplia gama de tareas de producción de audio, incluidas la grabación, mezcla y restauración de sonido.

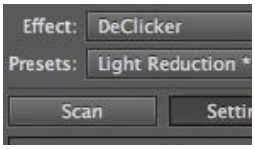
Algunas de sus funcionalidades principales son:

- **Compatibilidad nativa con Mac**
  Grabe, mezcle, edite y masterice el audio con un potente conjunto de herramientas de audio multiplataforma: Adobe Audition CS5 funciona nativamente en Mac OS X v10.5 y v10.6, así como en Windows®.

36 Página oficial de Adobe Audition CS5: http://www.adobe.com/es/products/audition.html

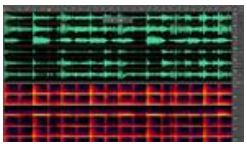
- **Herramientas potentes de mejora y restauración**

  Corrija los problemas de audio con rapidez. Desde la herramienta Reducción adaptativa de ruido y el nuevo diagnóstico como DeClicker, hasta el pincel de limpieza de audio para corregir visualmente los problemas de audio, Adobe Audition ofrece potentes funciones para perfeccionar la producción de audio.



- **Edición espectral y de forma de onda**

  Trabaje en la vista de forma de onda y en la visualización espectral en Mac o Windows. Edite el audio mediante la tradicional vista de forma de onda o la visualización espectral basada en frecuencias, que simplifica el aislamiento y la eliminación del ruido no deseado.

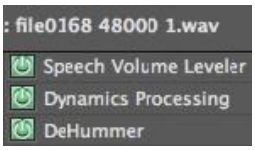


- **Mezcla multipista**

  Combine varias pistas de diálogo, fondos musicales, efectos de sonido y mucho más para crear bandas sonoras y experiencias de audio evocadoras. Divida o combine los clips, arrástrelos para copiarlos y automatice el volumen y panoramización en clips o pistas enteras.

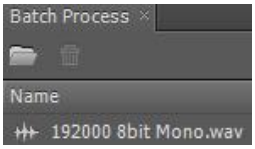
- **Potentes efectos DSP**

  Mejore el audio con los efectos profesionales y aproveche la potente compatibilidad con VST y Audio Unit de Mac. Corrija las producciones de audio pobres, diseñe paisajes sonoros, cree ambiente y haga que su banda sonora se corresponda con la alta calidad de sus imágenes.



- **Procesamiento por lotes**

  Con el procesamiento por lotes acelerará las tareas tediosas. Adobe Audition automatizará las operaciones, como la conversión de la velocidad de muestreo o el formato de varios archivos.

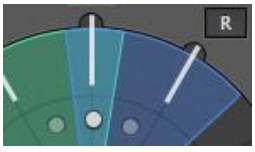


- **Intercambio de proyectos con otras aplicaciones de edición no lineal y estaciones de trabajo de audio digital**

  Mueva sesiones fácilmente entre Adobe Audition y las herramientas de Avid Pro Tools mediante las funciones integradas de importación y exportación de OMF de Adobe Audition. Comparta archivos con otras aplicaciones de edición no lineal a través del intercambio de XML.

- **Compatibilidad nativa multicanal 5.1**

  Edite archivos multicanal y mezcle sus proyectos en sonido 5.1 envolvente. Adobe Audition CS5 incluye una Panorámica envolvente en la vista Mezclador, así como un nuevo efecto de reverberación envolvente, y Amplificación, un procesador de ganancia multicanal.



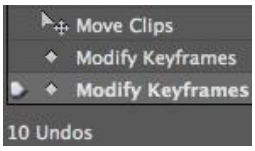
- **Contenido exento de costes por derechos de autor en Resource Central**

  Utilice el nuevo panel Resource Central de Audition para acceder a miles de efectos de sonido,

bucles y fondos musicales exentos de costes por derechos de autor, que podrá arrastrar y soltar directamente en sus proyectos creativos. Arrastre y suelte directamente en sus proyectos.

- **Archivos de sesión XML con almacenamiento automático**

  Guarde sus sesiones en el formato XML, que es eficaz, flexible y no destructivo. Deshaga los cambios no deseados en el nuevo panel Historial. Importe sesiones de Adobe Audition 3 y, a continuación, guárdelas como XML.



- **Panel de metadatos XMP simplificado**

  Visualice y edite los metadatos XMP mediante una interfaz simplificada. La compatibilidad con metadatos XMP incluye el esquema Broadcast WAV (BWF), lo que hace posibles los flujos de trabajo automatizados de Adobe Creative Suite® Production Premium y otros sistemas de producción de radio y televisión.

Para mayor información sobre las funcionalidades de Adobe Audition CS5 se puede acceder a la *sección de funciones* [37] de la página oficial.

37 Funciones de Adobe Audition CS5: http://www.adobe.com/es/products/audition/features.html

CAPÍTULO 3: Proceso de desarrollo

Una vez seleccionado el ***Game Engine*** y los ***programas externos***, comienza el ***proceso de desarrollo***. En este capítulo se presentan las diferentes actividades y etapas que corresponden al ciclo de vida del proyecto "Waku dice".

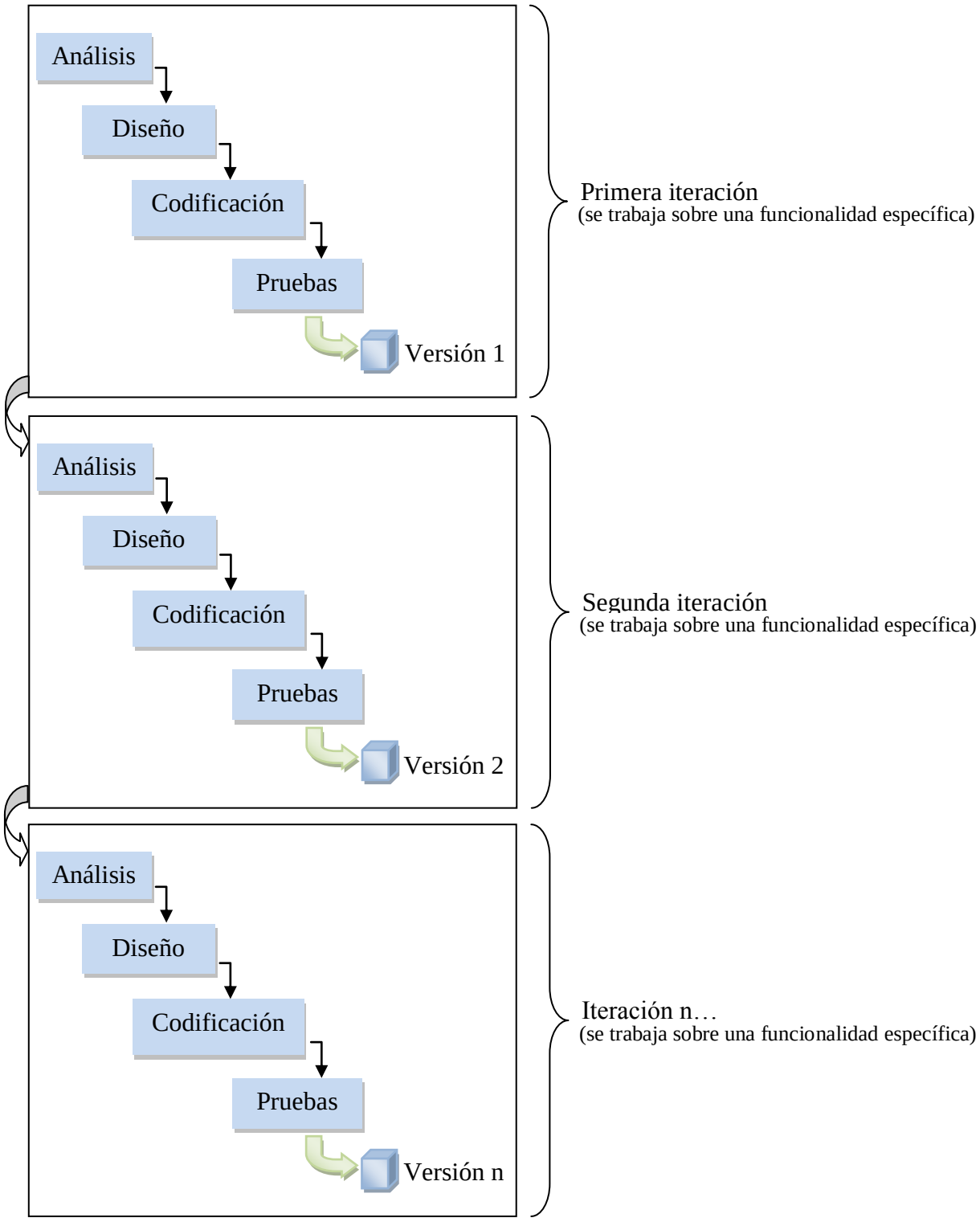
3.1. Modelo de proceso del videojuego

Para llevar a cabo el desarrollo del videojuego "Waku dice", se decidió utilizar un **Modelo iterativo e incremental**.

Algunas características de esta metodología son:

- El progreso se puede medir en periodos cortos de tiempo (ventaja, el cliente obtiene resultados más imediatos).
- Las pruebas de software y la integración son constantes (ventaja que permite atacar riesgos y detectar errores desde el principio).
- Construir funcionalidades modularmente es menos riesgoso que construír un sistema grande y de una sola vez (ventaja que pretende mejorar una debilidad del modelo en cascada tradicional).
- El cliente debe estar presente en todas las etapas del proceso de desarrollo (dependiendo de caso puede ser una ventaja o una desventaja).
- Si se detecta un error grave, sólo se desecha el último incremento o se soluciona en la próxima iteración.
- No es necesario disponer de los requerimientos de todas las funcionalidades en el comienzo del proyecto (ventaja que pretende mejorar una debilidad del modelo en cascada tradicional).

El diagrama siguiente muestra resumidamente el ciclo de vida que caracteriza a este modelo.



3.2. Artefactos

Es importante saber que, como en cualquier proyecto de software, el proceso de desarrollo de videojuegos también produce una serie de artefactos que ayudan a comprender mejor el ámbito del problema. Estos artefactos pueden ser visuales y descriptivos, como los diagramas de casos de uso, diagramas de clases (si se utiliza UML como lenguaje de modelado), y las diferentes plantillas que servirán para entender la naturaleza intrínseca del juego.

Existe un documento emitido por Gerardo Abraham Morales Urrutia, Claudia Esther Nava López, Luis Felipe Fernández Martínez y Mirsha Aarón Rey Corral, llamado *Procesos de desarrollo para Videojuegos* [38]**,** que introduce una nueva metodología llamada *Huddle*, análoga a SCRUM. Dicha metodología es muy reciente y es abierta al público, provee de muchas herramientas útiles, como la forma de trabajo y algunos artefactos que pueden producirse durante el proceso de desarrollo.

A continuación presentaremos una plantilla extraída de dicho documento: *la plantilla del documento de diseño.*

3.2.1. Plantilla del documento de diseño

38 Gerardo Abraham Morales Urrutia, Claudia Esther Nava López, Luis Felipe Fernández Martínez y Mirsha Aarón Rey Corral. «Instituto de Ingeniería y Tecnología (Universidad Autónoma de Ciudad Juárez).» ***Procesos de desarrollo para Videojuegos***. Enero-Abril de 2010. http://www2.uacj.mx/IIT/CULCYT/enero-abril2010/7%20Art%204.pdf.

Campo	Descripción
Concepto	
Título	El título del juego, debe ser un nombre que capte la atención del jugador y del lector del documento. A grandes rasgos, debe de incluir el concepto del juego. El titulo debe ser algo memorable.
Estudio/Diseñadores	El nombre del estudio y/o del diseñador o diseñadores del documento.
Género	El género abarca que tipo de juego será. Simulación, FPS, Rol, etc.
Plataforma	Qué hardware se requiere para jugarlo. Computadora Personal, Xbox 360, PS3, etc.
Sinopsis de jugabilidad y contenido	En uno o dos párrafos, describir la esencia de jugar el juego. Incluir un poco del contenido que tendrá, historia, personajes, objetivo, etc.
Categoría	Comparar con uno o varios juegos existentes y enfatizar en las diferencias y características principales de este juego.
Licencia	Describir si el juego está basado en un libro o en una película. Si es original, se puede omitir este campo o describir el por qué puede convertirse en una franquicia.
Mecánica	Describir la jugabilidad y el control del juego. ¿Qué hace el jugador? ¿Qué usa para lograr sus objetivos?
Tecnología	Enlistar que hardware y software se requiere para producir el juego. Desde lenguaje de programación hasta editor de sonidos.
Público	¿A quién va dirigido el juego? ¿Quién lo jugará? Se puede describir una demografía como niños o adolescentes, sin embargo, es más sencillo describir un tipo de jugador, ya sea casual, competitivo o veterano.
Versión	La versión del documento. Debe ser un número y no una fecha. (Ver el campo de Historial de Versiones)
Historial de versiones	
El Documento de Diseño es un artefacto que siempre estará sujeto a cambios, por lo tanto, un control para las diferentes versiones del documento y de los cambios que se han hecho es esencial. El número	

<table>
<tr><td colspan="2">de versión varía de acuerdo a si es un cambio mínimo o uno muy radical. El historial no se incluye en un documento que se somete a revisión por una empresa o grupo de desarrolladores debido a que incluye fechas, esto para evitar que juzguen la idea del juego como un concepto viejo.</td></tr>
<tr><td colspan="2">Visión general del juego</td></tr>
<tr><td colspan="2">Debe de establecer la visión y el enfoque del juego que guiará al proyecto hasta el final del proceso. El resumen debe mencionar lo más interesante, las ventajas y lo original del juego. ¿Por qué las personas jugarían este juego? La estructura de los párrafos es similar a un ensayo, una introducción debe de abarcar todos los aspectos importantes mientras que los párrafos subsecuentes deben detallar lo mencionado en la introducción. Al final, la conclusión debe dejar al lector entusiasmado y emocionado por jugar el juego.</td></tr>
<tr><td colspan="2">Mecánica del juego</td></tr>
<tr><td colspan="2">Esta sección esencialmente describe lo que el jugador puede hacer y cómo puede hacerlo. Describir las acciones del jugador, de preferencia en secuencia a cómo será en el juego.</td></tr>
<tr><td>Cámara</td><td>Describir el tipo de cámara que se utilizará. Es decir, qué perspectiva tiene el jugador ante lo que está viendo en el juego, si es 3D o 2D, vista isométrica, en primera persona, etc.</td></tr>
<tr><td>Periféricos</td><td>¿Qué periféricos utilizará el jugador para lograr los objetivos mencionados? Incluir todos los que apliquen: teclado, mouse, gamepad, micrófono, etc.</td></tr>
<tr><td>Controles</td><td>Describir los botones y teclas que invoquen las acciones mencionadas en la sección de Mecánica del Juego.</td></tr>
<tr><td>Puntaje</td><td>Explicar de qué manera el juego se mantiene al tanto de los logros del jugador. Incluir también si existe una tabla de puntajes que compare los mismos entre los jugadores, ya sea de manera local o en línea.</td></tr>
<tr><td>Guardar/Cargar</td><td>Describir cómo el jugador guarda su progreso de los objetivos logrados en el juego y cómo puede continuar los objetivos pendientes. De igual manera, describir los dispositivos de almacenamiento que se usarán o si el juego tiene un sistema de contraseñas.</td></tr>
<tr><td colspan="2">Estados del juego</td></tr>
</table>

Un estado del juego se refiere al lugar en donde se encuentra el jugador durante el juego, es decir, si el jugador está en el Menú Principal, está jugando un Juego Multijugador, está en el Menú de Pausa, etc. Los diagramas deben representar visualmente las relaciones entre los estados, si del Menú Principal se puede ir al Menú de Opciones, ¿Cómo lo hace? ¿Qué se ejecuta? ¿Qué interfaz muestra?	
Niveles	
Los juegos comúnmente se dividen en niveles o en mapas secuenciales dentro de los cuales se debe cumplir con ciertos objetivos para progresar en el juego. Existen juegos en los cuales los niveles solo cambian a razón de la dificultad y los objetivos siguen siendo los mismos, de igual manera se deben describir esos cambios en esta sección.	
Título del nivel	El nombre del nivel.
Encuentro	Describir si es el primer nivel, un tutorial o un bonus, en otras palabras, ¿Cuándo es que el jugador llega a este nivel?
Descripción	Una descripción detallada del nivel.
Objetivos	¿Qué debe de hacer el jugador para terminar el nivel? Este campo también debe incluir si el jugador tiene que resolver ciertos acertijos o derrotar a cierto enemigo para progresar.
Progreso	Describir que ocurre cuando el jugador termina el nivel.
Enemigos	Si el nivel tiene enemigos que el jugador debe enfrentar, éstos se enlistan en este campo, de lo contrario este campo puede ser omitido.
Ítems	Enlistar los objetos que el jugador o los enemigos pueden usar y que aparecen en este nivel, este campo se puede omitir si no existen dichos objetos.
Personajes	Los personajes que aparecen en el nivel, de igual manera, este campo puede ser omitido si no existen personajes en el juego.
Música y efectos de sonido	Describir la música de este nivel al igual que los efectos de sonido de ambiente que contiene.
Progreso del juego	
Enlistar de manera secuencial o por medio de un diagrama de flujo los eventos o niveles que el jugador	

debe de pasar para progresar en el juego. Existen juegos que tienen distintos modos de juego, en ese caso se requieren varias listas y/o diagramas.	
Personajes	
Los personajes principales y secundarios que aparecerán en el juego. Esta sección se puede omitir si el juego no tiene personajes.	
Nombre	El nombre del personaje.
Descripción	Describir detalladamente el físico del personaje, si es humano o extraterrestre, su vestimenta, etc.
Imagen	Fotografía o dibujo conceptual del personaje.
Concepto	Describir la conducta y comportamiento, al igual que los motivos del personaje. Mencionar también si es el enemigo principal o el protagonista. El concepto también puede relatarse como una historia del personaje, detallando en las relaciones con otros personajes del juego.
Encuentro	¿Cuándo aparece este personaje en el juego?
Habilidades	Enlistar las habilidades del personaje.
Armas	Enlistar las armas del personaje.
Ítems	Enlistar los objetos del personaje.
Personaje No-jugable	Si el personaje no es controlable por el jugador, describir su propósito para el juego y/o para el jugador.
Enemigos	
Los enemigos obstaculizan el progreso del jugador, pueden ser máquinas, otros personajes, monstruos, etc.	
Nombre	El nombre del enemigo.
Descripción	Describir detalladamente el físico del enemigo así como también su comportamiento.
Encuentro	¿Cuándo aparece este enemigo en el juego?
Imagen	Fotografía o dibujo conceptual del enemigo.
Habilidades	Enlistar las habilidades del enemigo.

Armas	Enlistar las armas del enemigo.
Ítems	Enlistar los objetos del enemigo.
Habilidades	
Los personajes y los enemigos llegan a tener ciertas habilidades fuera de las acciones comunes, en esta sección se describen cada una de ellas.	
Armas	
En esta sección se describen las armas que aparecerán en el juego.	
Ítems	
Todos los objetos especiales que ayudan al jugador a realizar los objetivos y progresar en el juego se mencionan aquí.	
Guión	
En esta sección se incluyen todos los diálogos del juego. Estos pueden ser muy variantes o inexistentes dependiendo de la naturaleza del juego. El guión debe de incluir encabezados, nombres, diálogo, acción y transiciones.	
Logros	
Describir los varios logros o hitos que el jugador obtiene mientras progresa en el juego. Estos pueden otorgar medallas, personajes secretos o puntos extra.	
Códigos secretos	
Describir los códigos secretos que el jugador puede ingresar, lo que hacen y cómo son ingresados.	
Miembros del equipo	
Información de las personas que trabajarán en el proyecto, incluye su nombre, el rol o roles que desempeñan y medios por los cuales se les puede contactar.	
Detalles de producción	
Antes de entrar a la etapa de Producción, se definen en el documento algunos detalles del proyecto.	
Fecha de Inicio	¿Cuándo empieza la etapa de Producción del proyecto?
Fecha de Terminación	¿Cuándo termina la etapa de Producción del proyecto?
Presupuesto	Una estimación aproximada del presupuesto del juego.

Esta plantilla ayuda a comprender mejor el proyecto y el videojuego en general (fue modificada

para el caso en particular del videojuego propuesto en esta tesina). Tambien se deben documentar algunos ítems de configuración importantes como los scripts de código y objetos. Para ello se plantean las siguientes plantillas:

3.2.2. Plantilla de Scripts

ID	Nombre
Un identificador único para el Script	El nombre del Script identificable en el Game Engine
Descripción	
Una descripción general de lo que hace el Script	
Código	
El código fuente del Script	

3.2.3. Plantilla de Objetos

ID	Nombre	Tipo	Clase	Script asociado	Dirección
Un identificador único para el Objeto	El nombre del Objeto identificable en el Game Engine	Tipo de Objeto	La clase a la que pertenece (ver diagrama de clases)	El script de código asociado al objeto (puede ser más de uno o ninguno)	La dirección desde la que se puede acceder al objeto en el Game Engine
Imagen					
Una imagen del objeto (puede no tener)					
Descripción					
Una descripción general del objeto					

3.3. El ciclo de vida

Cada iteración producida en el ciclo de vida debe producir un incremento en la funcionalidad del producto final y se deben llevar a cabo las cuatro etapas fundamentales del ciclo de vida: análisis, diseño, codificación y pruebas. A continuación se presentarán las actividades más significativas de cada etapa.

3.3.1. Etapa de Análisis

Cuando comienza una iteración, se ingresa directamente en la etapa de análisis. Esta etapa es de fundamental importancia porque en ella se define lo que se va a llevar a cabo durante toda la iteración, las metas y objetivos a cumplir, así como la funcionalidad que se desarrollará. Además, sobre el inicio de esta etapa se responden las siguientes preguntas.

1- ¿Se cumplieron los objetivos planteados en la iteración anterior?

2- ¿Se cumplió con los plazos?, ¿Cómo se pueden mejorar las estimaciones de tiempo?

3- ¿Hubo algún problema durante la iteración?, ¿Cómo se puede corregir?, ¿Cómo se puede prevenir para que no ocurra de nuevo?

4- ¿Qué conocimientos sobre el Game Engine se obtuvieron al llevar a cabo la iteración?

Estas preguntas son fundamentales para tener un control sobre el progreso del proyecto en general, además se atacan riesgos que pueden surgir durante las iteraciones y permite realizar una evaluación de las ventajas y desventajas del Motor de Videojuegos sobre el desarrollo de una funcionalidad específica. A partir de estas preguntas se realiza un trabajo de **planificación** sobre la

nueva iteración a ejecutar.

Haciendo hincapié en la funcionalidad, se realizan los siguientes pasos:

1. Se propone Añadir o modificar una funcionalidad específica.

   Por ejemplo:
   + Nueva funcionalidad: colocar un sistema de partículas que simule un efecto visual de explosión cuando el jugador gana la partida.

2. Se recolectan datos y la información necesaria para realizar la funcionalidad.

   Por ejemplo:
   + Precondición: el ambiente no puede estar demasiado iluminado, de lo contrario no podrá visualizarse el efecto de la explosión y su expansión por el escenario.

3. Se analiza los riesgos y la factibilidad técnica de llevar a cabo la funcionalidad.

   Por ejemplo:
   + Riesgo: una explosión con demasiadas partículas podría sobrecargar la memoria de video de la PC. Existe riesgo de tener que modificar esta funcionalidad en el futuro para lograr un mejor rendimiento.
   + Factibilidad técnica: crear y configurar un sistema de partículas es una tarea sencilla.

4. Se proponen las metas y objetivos a cumplir durante la iteración.

   Por ejemplo:
   + Esta funcionalidad debe codificarse a lo sumo en 3 días.
   + La iteración completa debe durar como máximo 5 días (se incluyen los días de pruebas).

5. Se comienza a trabajar sobre el modelo de análisis.

   Este es un modelo conceptual previo al modelo de diseño, es decir, es un bosquejo del diseño del programa particulamente orientado a la funcionalidad que se va a desarrollar en esta iteración (es siempre menos formal que el modelo de diseño).

Una vez que se ha trabajado lo suficiente sobre el modelo de análisis, se puede proseguir a la siguiente etapa.

3.3.2. Etapa de Diseño

El objetivo principal de esta etapa es transformar el modelo conceptual (modelo de análisis) en un modelo físico y concreto (modelo de diseño), es decir, un "plano" de la implementación. Esta etapa produce una serie de artefactos (que hacen al modelo) que deben ser mantenidos y actualizados durante toda la etapa de desarrollo del software. Un claro ejemplo son las plantillas presentadas en la sección 3.2 (Artefactos), diagramas de Casos de Uso y diagramas de Clases.

Generalmente, las actividades que se realizan en la etapa de diseño son las siguientes (haciendo referencia a una funcionalidad específica):

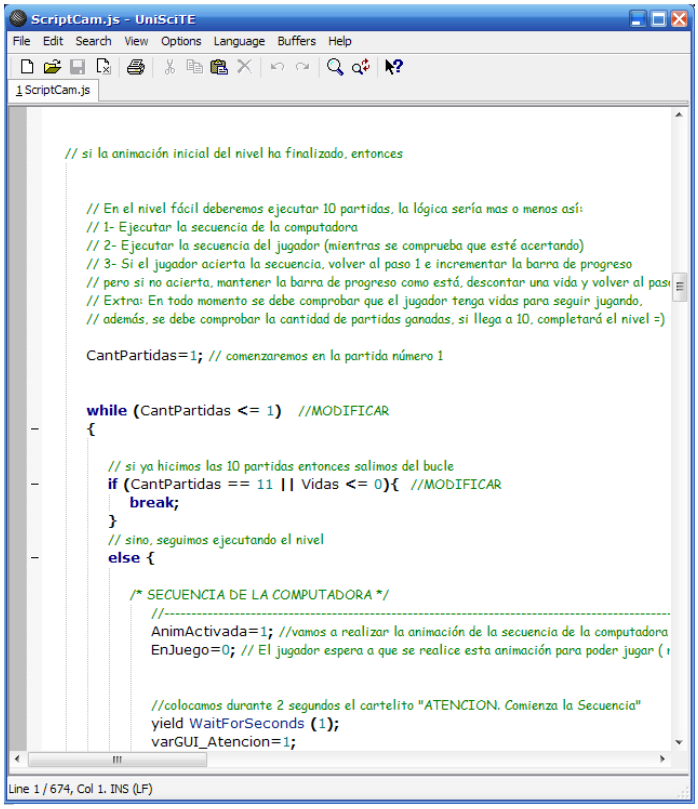
1. Se diseña la arquitectura.
   Se identifican nodos, configuraciones, escenas, niveles, pantallas, clases relevantes.
2. Se realizan bocetos de las gráficas.
   Se realizan los bocetos de las interfaces y gráficos en general (modelos 3D, modelos 2D, colores a utilizar, etc).
3. Se trabaja sobre el diagrama de clases.
   Identificando clases de diseño, interacciones entre objetos, etc.
4. Se trabaja sobre el diagrama de casos de uso.
   Identificando atributos, relaciones y operaciones…
5. Se actualizan o se crean nuevas plantillas de Script y Objetos.

De todas las actividades anteriores surge el modelo de diseño, que no es más ni menos que un conjunto de diagramas, plantillas, gráficos y anotaciones que servirán a la etapa siguiente.

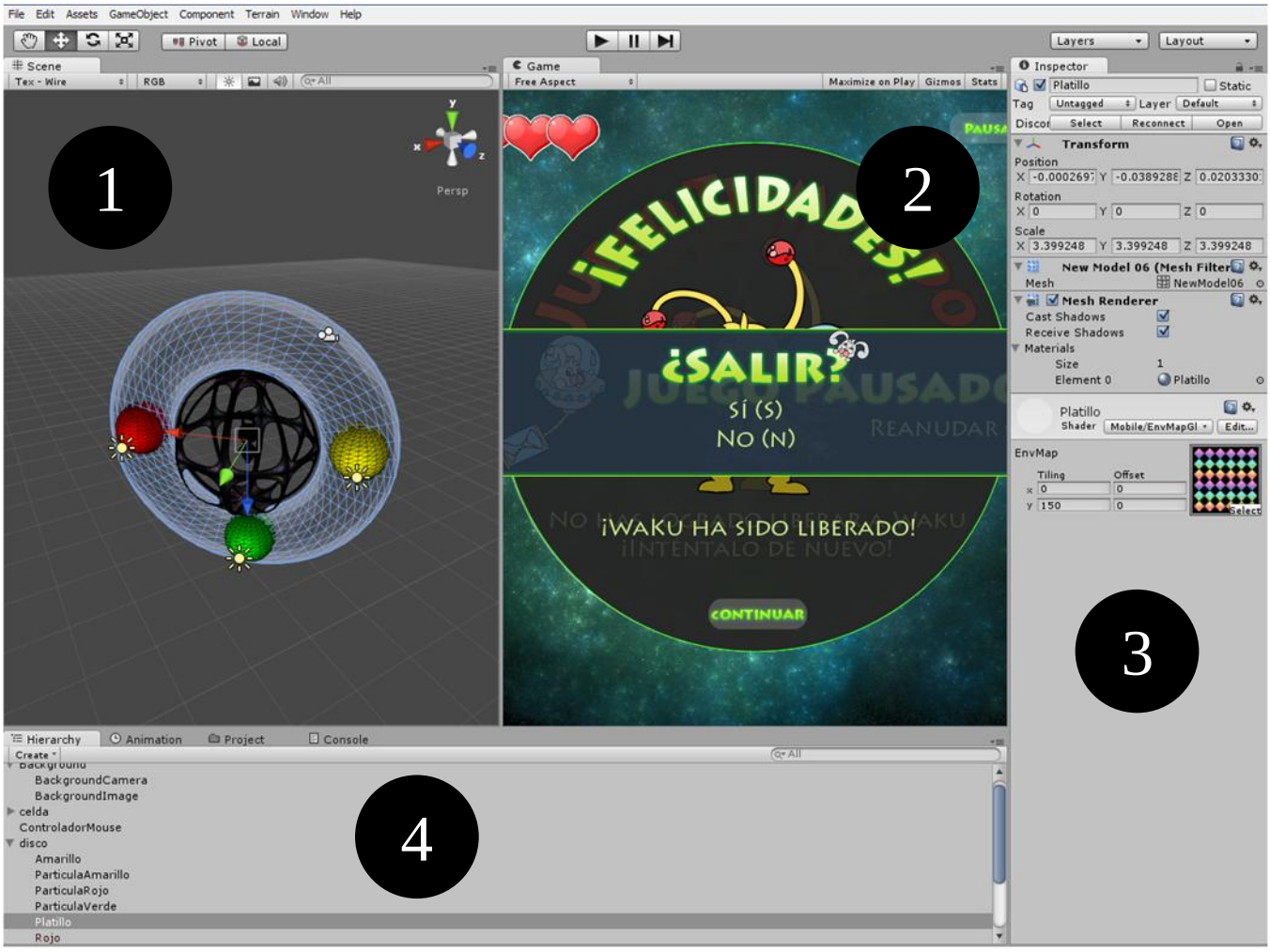
3.3.3. Etapa de Codificación

Al comienzo de esta etapa se debe contar con un modelo de diseño. El mismo permite llevar a cabo la etapa de codificación, en la que se lleva a código fuente todo lo realizado en la etapa anterior. Pero esa no es la única tarea, sino que además se crean los gráficos, interfaces y sonidos utilizando los programas mencionados en el capítulo anterior.

La codificación se realiza directamente desde el Game Engine, ya que el mismo posee un editor de código (o editor de scripts) llamado Uniscite en Windows y Unitron en Mac.



Unity3D también posee una interfaz de usuario amigable que permite la manipulación de todos los elementos del videojuego. Esta es, a grandes rasgos, la siguiente:



1- Scene o Vista de Escena

En Unity3D los juegos pueden dividirse en escenas. Generalmente las escenas se corresponden a los distintos niveles o pantallas de un juego, aunque podría ser de otra manera (eso se define en la etapa de diseño, cuando se diseña la arquitectura del juego).

La Vista de Escena es el área de construcción y edición de Unity. Provee de un entorno 3D para crear una escena, es decir: colocar objetos y editarlos (color, posición, tamaño), diseñar terrenos (con una herramienta que permite esculpir montañas, añadir líquidos como lagos, océanos y cataratas, o crear diferentes tipos de suelo como arena, pasto, roca, etc.), incluír skybox (o cielo), trabajar con generadores de partículas (explosiones, fuego, nieve, etc.) y muchas cosas más.

2- Game o Vista del Juego

Unity3D permite ejecutar el juego sin tener que salir del editor (o del programa), es decir que no se necesita generar un archivo ejecutable del videojuego para poder probarlo, sino que se puede reproducir desde la misma interfaz. Esto es muy favorable en tiempos de diseño porque permite obtener una vista previa rápida del juego con la opción de cambiar las resoluciones y probar distintas configuraciones. Así también es muy positivo en la etapa de codificación, ya que permite detectar errores y/o ajustar los elementos del juego, inclusive probar el comportamiento del mismo.

La barra de reproducción permite poner el juego en movimiento y como se puede observar, se puede parar o pausar en cualquier momento; esto es realmente útil cuando se realizan tareas de Debug, ya que permite visualizar el valor que toman las distintas variables en un momento dado y las configuraciones de los objetos del videojuego.

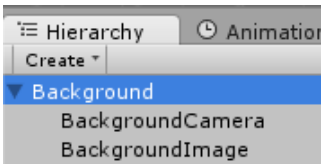
3- Vista de Inspector

El inspector muestra los componentes de un objeto seleccionado (por ejemplo los scripts asociados) y permite editar las propiedades del mismo. También permite ver las configuraciones generales del videojuego (proyect settings, render settings, etc.).

4- Otras Vistas

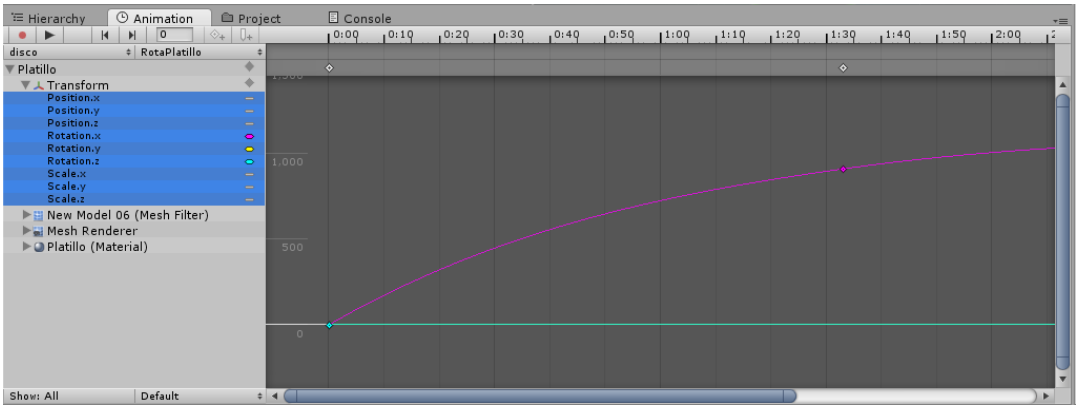
- **Hierarchy o Vista de Jerarquía**

  Esta vista permite visualizar una lista de todos los objetos en la escena actual. Es decir que cuando cargamos una escena en Unity, se actualiza la lista de objetos según corresponda. Cabe destacar que se puede organizar la lista agrupando los diferentes objetos en otros objetos "contenedores", cuyo propósito en el juego no es más que contener y agrupar a otros objetos con el fin de ordenar u organizar la estructura.



  En la imagen se puede observar como el objeto "Background" es un objeto "contenedor", es decir que no posee un comportamiento identificable dentro de la mecánica del juego, pero sirve para organizar la estructura de fondos de pantalla.

- **Animation o Vista de Animación**

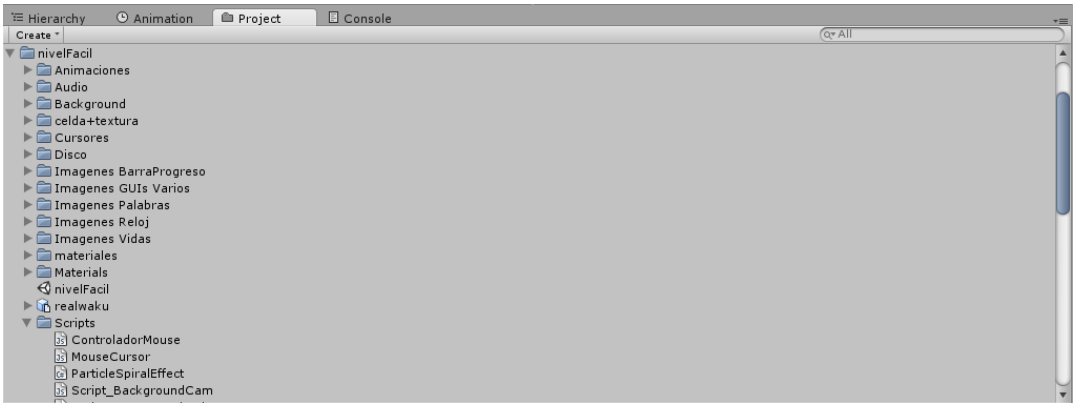


  Esta vista permite crear una animación para un objeto seleccionado. En el juego de "Waku Dice" puede identificarse una animación de rotación en el Platillo cuando comienza el nivel.

Esta herramienta es muy útil cuando se desea aplicar movimientos a un personaje (saltar, caminar, correr, agacharse, golpear, etc.) o a otros elementos, por ejemplo el movimiento rotatorio de un planeta, el vuelvo de las aves, etc.

- **Project o Vista de Proyecto**

  La Vista de Proyecto presenta una lista (muy extensa en algunos casos) de todos los Assets (activos) y librerías del videojuego. Todos los activos importados (modelos 3D, gráficos 2D, sonidos, texturas, etc.) se almacenan aquí para poder ser utilizados en el videojuego, también se encuentran los Scripts y Prefabs (objetos prefabricados).



Con las herramientas de Unity mensionadas anteriormente se puede realizar la codificación completa del videojuego. En cada iteración se codifica una nueva funcionalidad o se modifica una existente, lo que produce un incremento en el producto final y un progreso en el proyecto.

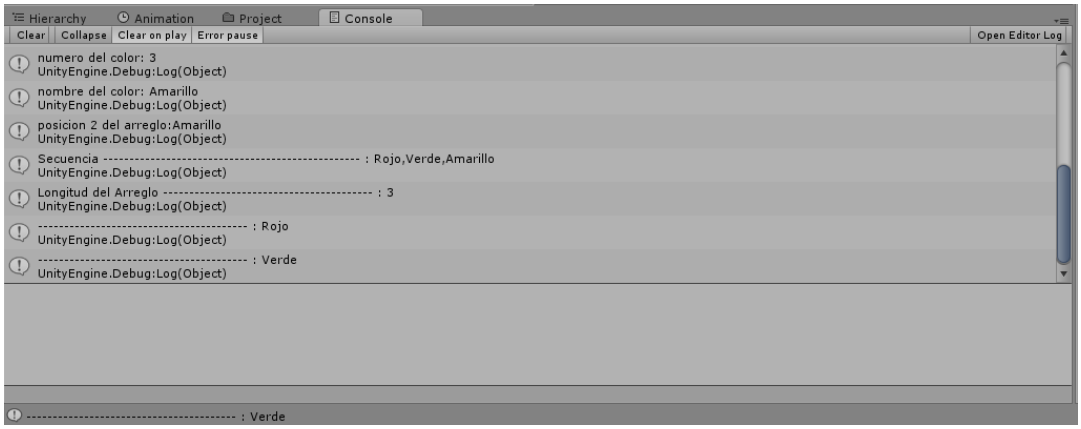
Como resultado de las actividades de codificación, se obtiene un sistema testeable. Es decir, se logra un un incremento en el software que debe estar necesariamente sujeto a pruebas, detección y corrección de errores. Además se actualizan las plantillas de Script y Objetos creadas en la etapa anterior.

3.3.4. Etapa de Pruebas

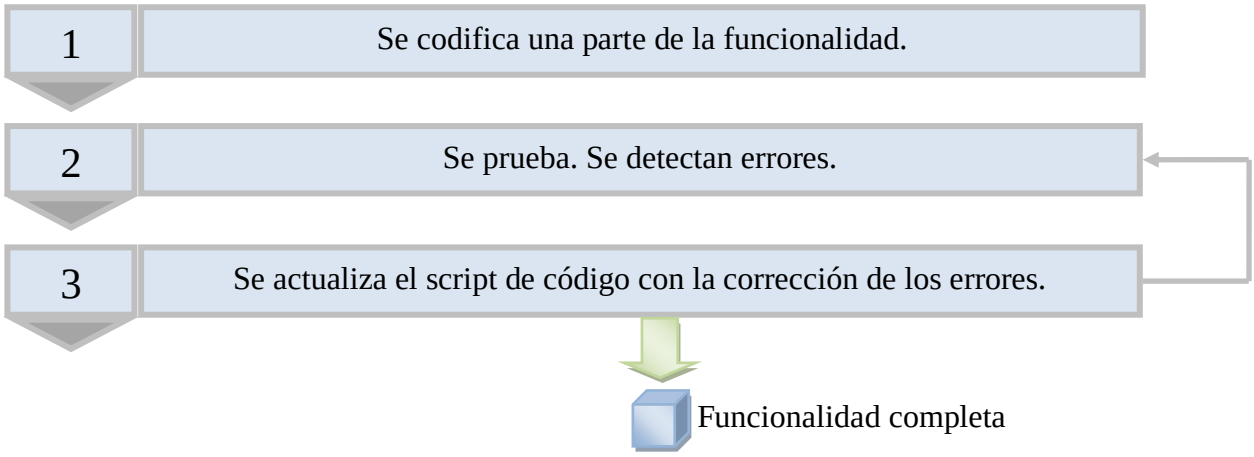
De las herramientas mencionadas en la sección anterior, sólo queda presentar la vista que permite realizar las pruebas, detectar y corregir los errores del programa.

- **Console o Vista de Consola**

  Es muy útil para tareas de Debug, ya que permite mostrar el estado de las variables del juego mientras éste se ejecuta en la vista previa (o Vista del Juego). Permite pausar la ejecución si hay algún error para poder detectarlo y corregirlo.



La etapa de pruebas está fuertemente relacionada con la etapa de codificación, existe un real solapamiento entre ellas, ya que a medida que se va codificando el videojuego se va probando y los errores se van corrigiendo en ese mismo instante. El proceso es el siguiente:


1
Se codifica una parte de la funcionalidad.
2
Se prueba. Se detectan errores.
3
Se actualiza el script de código con la corrección de los errores.
Funcionalidad completa

CAPÍTULO 4: Videojuego “Waku dice”

En este capítulo se presentan los puntos más significativos que surgieron durante el desarrollo del videojuego propuesto para esta tesina: “Waku Dice”.

4.1. Objetivos del videojuego

- Fomentar la utilización de videojuegos no violentos.
- Ayudar a desarrollar habilidades de percepción, razonamiento y observación.
- Potenciar actitudes como las de auto-confianza, auto-disciplina o perseverancia en la búsqueda de soluciones.
- Ejercitar la memoria.
- Desarrollar la atención.

4.2. Alcances y Limitaciones

- El software funcionará bajo la plataforma Windows.
- El juego será gratuito, pero no se facilitarán los archivos fuente del mismo.
- El juego podrá estar desarrollado en 2D y/o 3D.
- El juego deberá tener planteado al menos 3 (tres) niveles de dificultad, y se debe implementar al menos 1(un) nivel de dificultad.
- El juego presentará interfaces simples y amigables.
- El videojuego creado será del tipo POST-INSTRUCCIONAL: ya se ha recibido la enseñanza sobre un tema, y mediante el juego se realizan actividades para reforzar lo aprendido. Por lo tanto, el juego sirve para consolidar el aprendizaje.

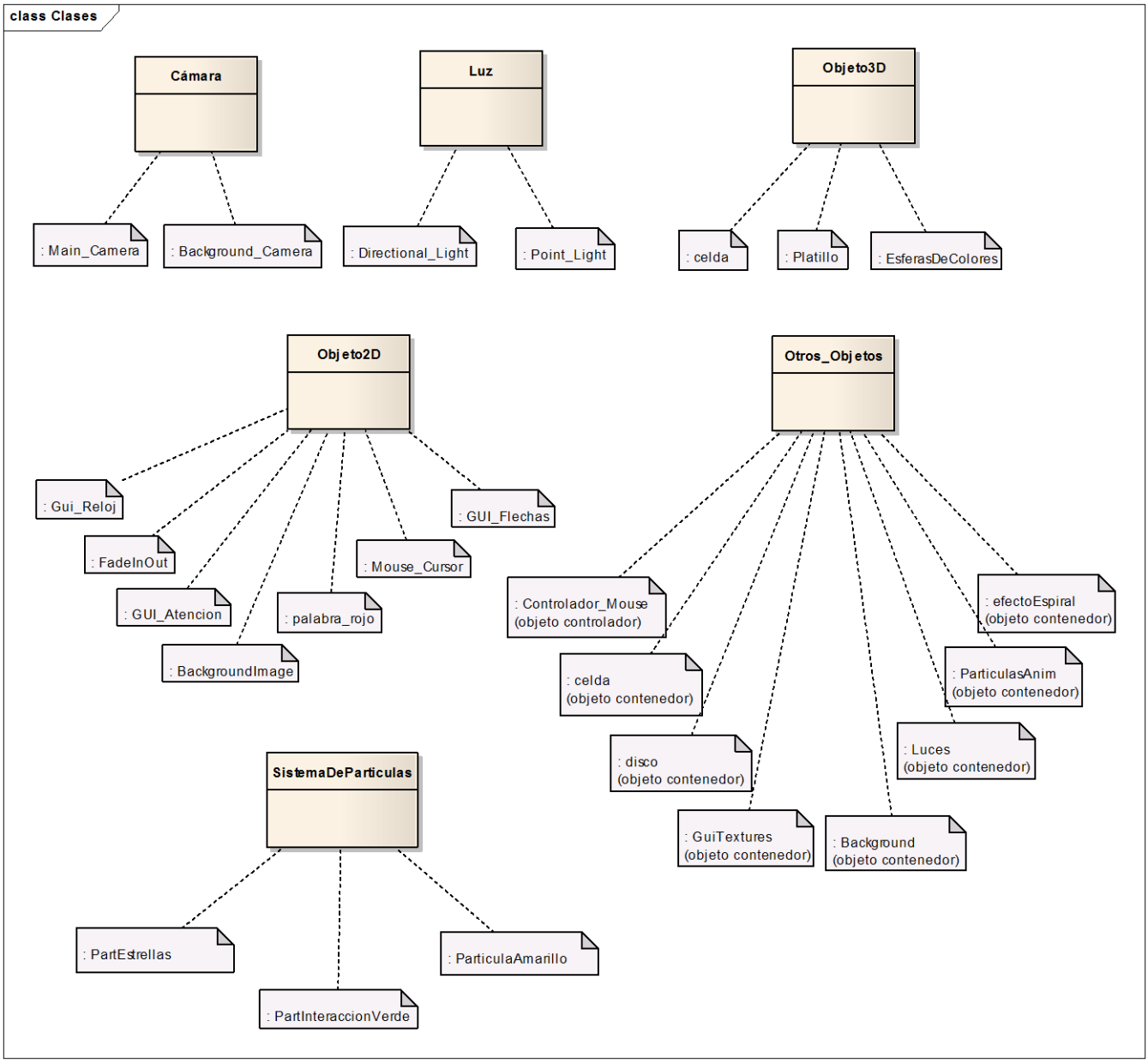
4.3. Desarrollo: Nivel Fácil

Uno de los alcances propuestos para el videojuego define que se deben plantear tres niveles de dificultad como mínimo y desarrollar al menos uno, el cual sería desarrollado utilizando el Game Engine previamente seleccionado. El nivel escogido para el desarrollo es el **nivel fácil** (primer nivel del juego).

Durante el ciclo de vida se llevaron a cabo las actividades descriptas en el capítulo anterior, utilizando la metodología mencionada en la sección 3.1 del mismo capítulo. Como resultado se logró completar y depurar todos los artefactos detallados en la sección 3.2. Estos son los siguientes:

4.3.1. Diagrama de clases

Utilizando UML como lenguaje de modelado, en cada etapa de diseño se fue refinando el diagrama de clases. El mismo pretende mostrar a modo conceptual los distintos tipos de objetos que intervienen en el primer nivel del juego. Podemos decir entonces que es, a modo esquemático, el esqueleto que permite “categorizar” a los distintos objetos del nivel fácil.



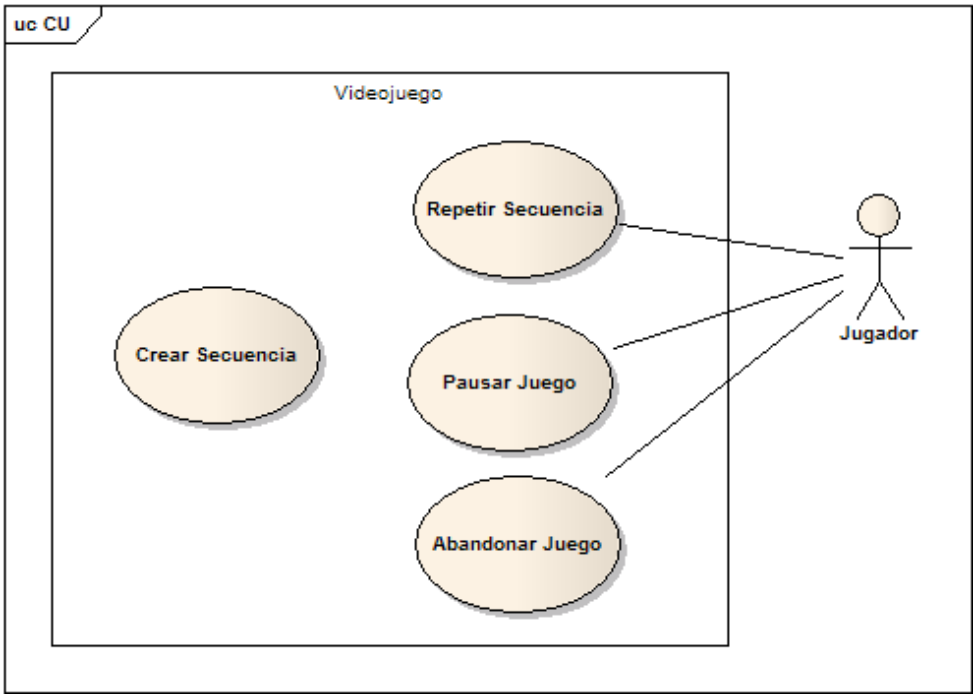
Como se puede observar en la imagen, existen seis clases diferentes: **Cámara, Luz, Objeto3D, Objeto2D, Otros_Objetos y SistemaDeParticulas**. Estas Clases no están implementadas en el código directamente, sino que están presentes en el modelo para poder definir bajo algún nombre, la clasificación de los diferentes objetos del nivel fácil. Esto significa que el modelo de clases **no sirve para conocer la estructura, conexiones y disposición de los diferentes objetos dentro del Game Engine**. Para ello se cuenta con la siguiente lista, obtenida de la Vista de Jerarquía de Unity.

- Estructura y disposición de objetos en el Game Engine (Hierarchy o Vista de de Jerarquía)

La lista anterior nos permite visualizar los diferentes tipos de objetos y como están ordenados en el Motor de Videojuegos.

4.3.2. Diagrama de Casos de Uso

El diagrama de casos de uso general que se presenta a continuación esquematiza, a grandes rasgos, la interacción del usuario con el sistema (videojuego).



- **Crear secuencia**

  Este caso de uso lo lleva a cabo la computadora, cuando genera una secuencia aleatoria de colores por partida. Esta secuencia es almacenada en memoria para luego compararse con la secuencia del jugador y poder determinar si el jugador acertó o se equivocó.

- **Repetir Secuencia**

  El jugador repite la secuencia que generó la computadora en esa partida, las dos secuencias se comparan, y si el jugador acertó entonces ganará la partida, en caso contrario perderá la partida y se le restará una vida.

- **Pausar juego**

  En cualquier momento, el jugador debe poder pausar el juego.

- **Abandonar Juego**

  En cualquier momento, el jugador debe poder abandonar el juego.

4.3.3. Plantilla del documento de diseño

Campo	Descripción
Concepto	
Título	Waku Dice
Estudio/Diseñadores	Games Ink (Estudio ficticio)
Género	Serious Game
Plataforma	▪ El juego fue probado satisfactoriamente en una PC hogareña Pentium 4 con 1Gb de RAM y 256Mb de Video. ▪ Windows XP / Vista / 7
Sinopsis de jugabilidad y contenido	El juego se basa en una secuencia de colores generada por la computadora que debe ser repetida por el usuario. Posee cuatro niveles de dificultad con 10 partidas por cada nivel. En una partida se genera una secuencia aleatoria de colores que el jugador deberá repetir utilizando el mouse. Cuando el jugador logra completar las 10 partidas de un nivel, terminará el juego satisfactoriamente logrando liberar a Waku de la celda espacial que lo mantiene prisionero.
Categoría	Existen actualmente juegos similares desarrollados para otras plataformas, bajo el nombre de “Simón Dice”, en su mayorìa orientados a la Web. En general son producciones relativamente sencillas, no comerciales, que no suponen grandes trabajos de desarrollo. “Waku dice” implementa varias mejoras sobre el juego original y está desarrollado en un motor de última generación como es Unity3D; esto asegura una buena escabilidad y actualización, inclusive, el soporte a otras plataformas.

Licencia	El juego es gratuito y no posee licencia comercial.
Mecánica	El jugador puede utilizar el mouse para seguir una secuencia, haciendo click sobre los respectivos colores. Otros controles importantes son las teclas: "P" de Pausa y "Esc" de Salir.
Tecnología	Desarrollado principalmente en Unity (Game Engine) y con el soporte de otros programas externos como GIMP, Inkscape, Audacity y Blender.
Público	Principalmente Niños y Adolescentes, pero puede ser jugado por el público en general.
Versión	1.0 (versión del documento)

Historial de versiones

Se posee actualmente una sola versión del Documento de Diseño, la 1.0

Visión general del juego

El juego se basa en una secuencia de colores generada por la computadora que debe ser repetida por el usuario. En una partida se genera una secuencia aleatoria de colores que el jugador deberá repetir poniendo a prueba sus capacidades de atención, observación y memoria.

Algunos puntos fuertes del videojuego son los siguientes:

- Es gratuito y podrá descargarse de internet.
- Tiene distintos niveles de dificultad.
- Se puede compartir y distribuir libremente.
- Está pensado para los niños, con la filosifía "Jugar, divertirse y aprender".
- No se recurre a la violencia para llamar la atención del jugador, sino al uso de la astucia, la observación y el razonamiento.

Mecánica del juego

Durante la ejecución del Nivel Fácil el "Player" podrá:

1- Jugar su turno, siguiendo una secuencia que previamente fue generada por la computadora y reproduciéndola a través de los objetos "esfera" de color que aparecen en la pantalla.

2- Pausar el juego.

3- Salir del juego.

Cámara	La cámara tiene una perspectiva 3D para el juego en general.
Periféricos	Teclado y Mouse.
Controles	1- Jugar su turno con el mouse. 2- Pausar el juego con la tecla “P”. 3- Salir del juego con la tecla “Esc”.
Puntaje	Existe una barra de progreso que lleva de una forma gráfica la cantidad de partidas ganadas por el jugador en el Nivel Fácil. Cuando la barra de progreso se completa significa que se ha completado el nivel con éxito.
Guardar/Cargar	No existen éstas opciones para el juego.

Estados del juego

Estados en la escena del Nivel Fácil:

- Esperando secuencia: El jugador espera a que la computadora genere una secuencia aleatoria de colores.
- Jugando turno: El jugador está jugando su turno, intentando acertar la secuencia anteriormente emitida por la computadora.
- En pausa: El jugador pulsó la tecla “P” y el juego se ha pausado.
- Por salir: El jugador pulsó la tecla “Esc” para salir pero todavía le falta confirmar la acción (tecla “S”) o rechazarla (tecla “N”).

Estados en las otras escenas:

- Introducción: El jugador está iniciando el juego y observando las pantallas de introducción.

Niveles

En el videojuego propuesto existen cuatro niveles de dificultad, de los cuales sólo se desarrolló por completo el primero (nivel fácil). Los otros tres niveles están planteados en el proyecto y se podrá completar su desarrollo en el futuro.

Detalle de los niveles planteados:

- El nivel fácil posee 3 colores: azul, rojo y amarillo.
  Las secuencias que genera la computadora serán de 2 (dos) o 3 (tres) colores por partida. El

nivel finaliza cuando se hayan completado 10 partidas (se libera a Waku y se termina nivel).

- El nivel medio tiene 4 colores: azul, rojo, verde y amarillo.
  Las secuencias que genera la computadora serán de 2 (dos) o 3 (tres) colores por partida. El nivel finaliza cuando se hayan completado 10 partidas (se libera a Waku y se pasa de nivel).

- El nivel difícil posee 6 colores: azul, rojo, verde, amarillo, violeta y naranja.
  Las secuencias que genera la computadora serán de 3 (tres) o 4 (cuantro) colores por partida. El nivel finaliza cuando se hayan completado 10 partidas (se libera a Waku y se pasa de nivel).

- El nivel muy difícil tiene 9 colores: azul, rojo, verde, amarillo, violeta, naranja, cian, magenta y marrón.
  Las secuencias que genera la computadora serán de 4 (cuatro) a 6 (seis) colores por partida. El nivel finaliza cuando se hayan completado 10 partidas (se libera a Waku y finaliza el juego).

Título del nivel	Nivel Fácil
Encuentro	Es el primer nivel del videojuego. El jugador deberá necesariamente completar este nivel para poder desbloquear los otros tres niveles de mayor dificultad.
Descripción	El nivel fácil posee 3 colores: azul, rojo y amarillo. Las secuencias que genera la computadora serán de 2 (dos) o 3 (tres) colores por partida.
Objetivos	El nivel finaliza cuando se hayan completado 10 partidas (se libera a Waku y se termina nivel).
Progreso	Cuando el jugador termina el nivel, gana su recompensa, que es liberar al personaje principal del juego “Waku” de la prisión espacial que lo mantiene cautivo. Además, finalizar el nivel fácil permite desbloquear el nivel medio.
Enemigos	-

<table>
<tr><td>Ítems</td><td>-</td></tr>
<tr><td>Personajes</td><td>Waku</td></tr>
<tr><td>Música y efectos de sonido</td><td>Se utilizó Audacity para editar la música y los efectos de sonidos incluídos en el videojuego.</td></tr>
<tr><td colspan="2">Progreso del juego</td></tr>
<tr><td colspan="2">1 - La computadora genera una secuencia aleatoria de colores.
2- El jugador trata de acertar la misma secuencia.
3- Si el jugador acierta la secuencia
3.1- Si el progreso es igual al 100% (osea que ha completado todas las partidas)
3.1.1- Gana el juego y finaliza el nivel satisfactoriamente
3.2- Si el progreso no es igual al 100% (todavía no completa todas las partidas)
3.2.1- Gana la partida actual y comienza una nueva partida. Vuelve al paso 1
4- Si el jugador no acierta la secuencia
4.1- Si al jugador le quedan vidas disponibles
4.1.1- Pierde la partida y se descuenta una vida. Vuelve al paso 1
4.2- Si al jugador no le quedan vidas disponibles
4.2.1- Pierde el juego y finaliza el nivel sin lograr liberar a Waku</td></tr>
<tr><td colspan="2">Personajes</td></tr>
<tr><td>Nombre</td><td>Waku</td></tr>
<tr><td>Descripción</td><td>Es un extraterrestre y es el personaje principal del videojuego.</td></tr>
<tr><td>Imagen</td><td>  </td></tr>
<tr><td>Concepto</td><td>Waku es un personaje espacial, amigo del jugador y protagonista del videojuego.</td></tr>
<tr><td>Encuentro</td><td>Waku aparece durante todo el juego.</td></tr>
</table>

Habilidades	-
Armas	-
Ítems	-
Personaje No-jugable	Waku no es un personaje controlable por el jugador. Permanece prisionero durante el desarrollo del juego hasta que el mismo Player se convierte en el héroe del videojuego al liberarlo de su prisión espacial.
Enemigos	
No existen enemigos identificables en el juego	
Nombre	-
Descripción	-
Encuentro	-
Imagen	-
Habilidades	-
Armas	-
Ítems	-
Habilidades	
-	
Armas	
-	
Ítems	
-	
Guión	
-	
Logros	
Logro: Al finalizar el nivel fácil, el jugador libera al personaje Waku y desbloquea el nivel medio.	
Códigos secretos	
-	
Miembros del equipo	
Jesús Arce (Argentina): Programador y diseñador multimedial, principal responsable del proyecto.	

Diego Pantoja (Chile): Diseñador gráfico. Colaborador del proyecto. Valentina Estevez: Diseñadora Gráfica. Colaboradora del proyecto.	
Detalles de producción	
Fecha de Inicio	Lunes 6 de Septiembre de 2010
Fecha de Terminación	Viernes 20 de Mayo de 2011
Presupuesto	Desarrollar el videojuego no tiene costo alguno gracias a la utilización de tecnologías libres y la disposición de una versión gratuita del Game Engine Unity.

4.3.4. Plantilla de Scripts

Para simplificar la legibilidad y la lectura de este proyecto, la plantilla de Script se adjunta al trabajo como un Apartado (Ver Apartado 1: Plantilla de Scripts).

4.3.5. Plantilla de Objetos

Para simplificar la legibilidad y la lectura de este proyecto, la plantilla de Objetos se adjunta al trabajo como un Apartado (Ver Apartado 2: Plantilla de Objetos).

4.4. Bocetos y Prototipos

Una etapa creativa fundamental en el desarrollo de cualquier videojuego, es la creación de los bocetos y prototipos en forma gráfica que permitan definir la idea principal y otros aspectos importantes como son los personajes, las interfaces de usuario, las pantallas, etc.

Para simplificar la legibilidad y la lectura de este proyecto, los bocetos y prototipos se adjuntan al trabajo como un Apartado (Ver Apartado 3: Bocetos y Prototipos).

4.5. Las Escenas

En la etapa de codificación se realizó la implementación de tres escenas:

- **"Escena1"**
  Corresponde a las pantallas iniciales del juego que aparecen con un efecto fade-in y desaparecen con un fade-out.

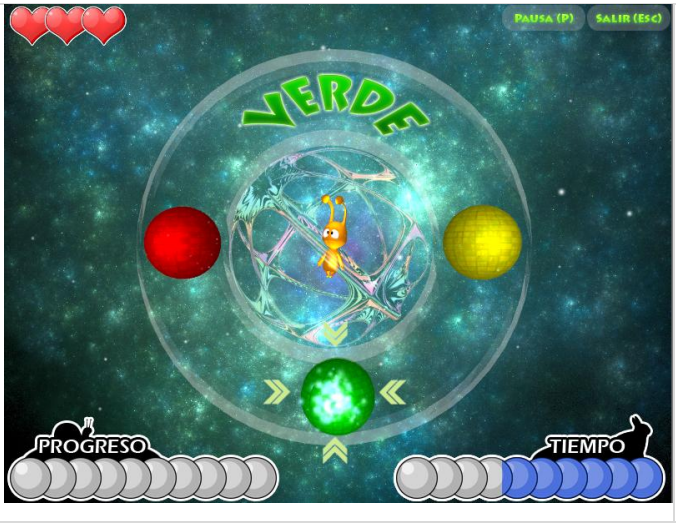
- **"Escena2"**
  Corresponde al menú principal del juego, desde donde se puede acceder a jugar, ver las instrucciones, o salir. Por el momento no se ha codificado la pantalla de "Cómo jugar" (instrucciones).

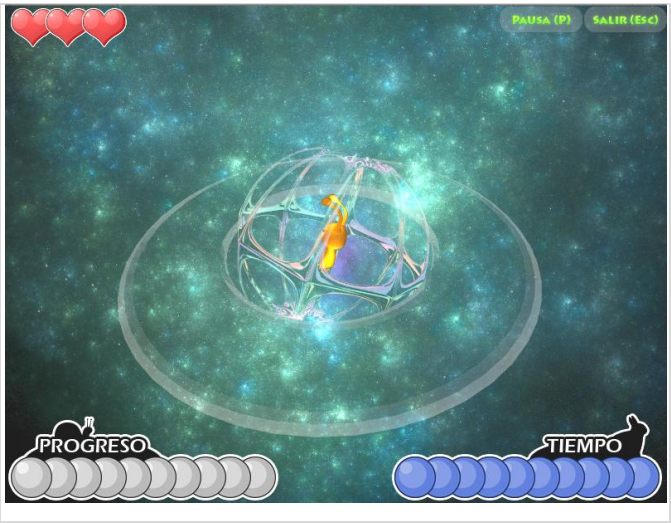
- **"seleccionarNivel"**
  Corresponde a la pantalla desde donde el jugador puede optar por los diferentes niveles de dificultad del videojuego. Por el momento sólo se ha codificado el nivel fácil.

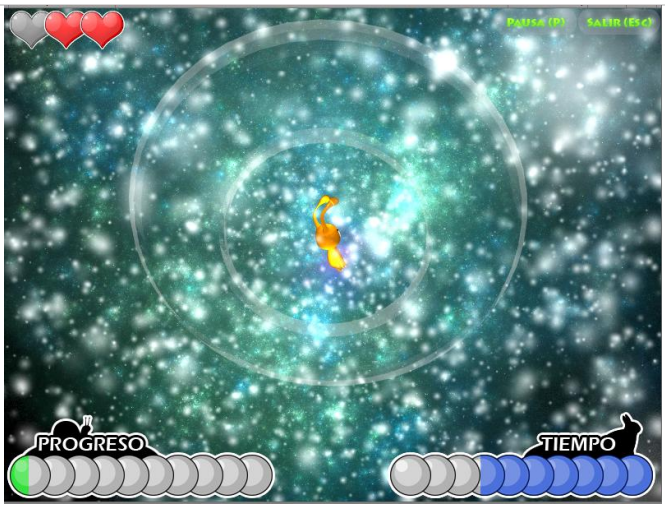
- **"nivelFácil"**
  Corresponde al primer nivel del videojuego. Actualmente es el único nivel jugable y codificado al 100%.

4.6. Capturas de pantalla

En esta sección se presentan algunas imágenes o capturas de pantallas del videojuego en ejecución (gameplay).

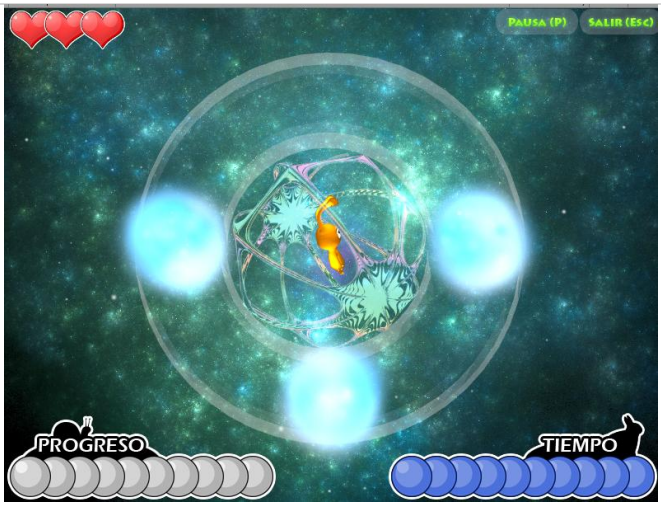







PAUSA (P)
SALIR (Esc)
JUEGO PAUSADO
REANUDAR (P)
PROGRESO
TIEMPO


PAUSA (P)
SALIR (Esc)
JUEGO TERMINADO
NO HAS LOGRADO LIBERAR A WAKU
¡INTÉNTALO DE NUEVO!
CONTINUAR
PROGRESO
TIEMPO


PAUSA (P)
SALIR (Esc)
PROGRESO
TIEMPO


PAUSA (P)
SALIR (Esc)
AMARILLO
¿SALIR?
SÍ (S)
NO (N)
PROGRESO
TIEMPO


COMENZAR JUEGO
COMO JUGAR
SALIR


MUY DIFÍCIL
FÁCIL
DIFICULTAD
FÁCIL:
DE 5 A 6 AÑOS
DIFÍCIL
MEDIO

4.7. Ejecutables y Multimedia

Para concluír la presentación de **Waku Dice**, se ha realizado un video casero que muestra algunas etapas del desarrollo del juego, el mismo se encuentra en la ruta **/multimedia/Waku dice** del DVD adjunto con esta tesina. Además, en la ruta **/Waku dice** del mismo DVD, está a disposición el videojuego (se puede ejecutar abriendo el archivo "**Waku Dice.exe")**.

En los capítulos anteriores se comentó que Unity tiene una ventaja que podría ser explotada en el marco de esta tesina, o para futuros proyectos. Ya que permite publicar los juegos en un navegador web, a través de un plugin especial creado para tal propósito. Dicha funcionalidad fue explorada a modo de experimentación, los archivos pueden encontrarse en la ruta **Waku dice - Web Player\WebPlayer** del DVD adjunto con la tesina.

III. CONCLUSIONES

1- Utilizando Unity como tecnología principal de desarrollo, con el apoyo de otras tecnologías como Blender para los gráficos 3D, Inkscape y/o GIMP para gráficos 2D, Audacity para el sonido y a través de la aplicación sistemática y organizada de una metodología de desarrollo, se logró cumplir con los objetivos de la tesina, los cuales especifican:

- Analizar y poner a prueba un Game Engine de última generación.
- Listar y proponer otras aplicaciones externas que sirvan de apoyo al desarrollo de videojuegos.
- Desarrollar un juego de computadora sencillo con las tecnologías propuestas y analizadas: Game Engine y Aplicaciones externas.

El videojuego creado: **Waku Dice**, está lejos de ser un videojuego profesional y distante de ser estética y funcionalmente aceptable para darle un motivo comercial. No obstante, es demostrable que a diferencia del pasado, en la actualidad es factible adquirir tecnologías de vanguardia, como lo es un Game Engine moderno, para desarrollar videojuegos experimentales, independientes o de bajo costo.

2- El proceso de aprendizaje de Unity para cualquier individuo depende en gran medida de la documentación existente, que cada día es mayor, y de las horas dedicadas al esfuerzo por comprender esta fantástica tecnología de desarrollo.

En las plantillas de código dispuestas en el Apartado 1 de esta tesina, se puede encontrar una base de Scripts rica en conocimientos y recursos, que se publica libremente a disposición de todas aquellas personas que quieran utilizarla en el futuro.

3- Una característica curiosa pero muy llamativa de Unity, es que permite publicar los juegos en un navegador web, a través de un plugin especial creado para tal propósito. Dicha funcionalidad fue explorada a modo de experimentación, pero sería interesante incursionar más sobre esta funcionalidad,

por lo que se propone a futuros entusiastas del tema considerar esta característica y expandir las fronteras de este trabajo.

4- Las implicancias de trabajar con un Motor de Videojuegos de última generación, trasciende al objetivo de crear juegos, ya que esta tecnología no se limita a ello únicamente, sino que también puede servir en otras disciplinas o ámbitos como son la arquitectura, la educación, la investigación, la multimedia en general y la producción de otros tipos de software.

5- Una metodología de desarrollo de videojuegos no es más que un proceso de desarrollo de software orientada a la creación de juegos. Esto significa que se pueden utilizar todas las herramientas y buenas prácticas de la Ingeniería de Software para crear videojuegos consistentes y escalables. Y que al terminar el desarrollo de un juego en particular, queden los procesos establecidos para poder ser reutilizados en otros proyectos de videojuegos.

6- Debido a que la industria de los videojuegos en la República Argentina es muy reciente y se encuentra en pleno crecimiento, existe muy poca información sobre e tema. Realizar un trabajo de análisi teórico-práctico orientado hacia las tecnologías de videojuegos, permite dar el punta pie inicial para que nuevos investigadores y personas interesadas en el tema utilicen los recursos y herramientas puestos a disposición en este trabajo para progresar y desarrollar nuevos avances en el ámbito. Inclusive, progresar con nuevas investigaciones, abriendo los horizontes y presentando nuevas metodologías que ayuden a formalizar los procesos de desarrollo de los videojuegos.

IV. BIBLIOGRAFÍA

Libros:

Eberly, David H. *3D Game Engine Architecture*. Primera edición. San Francisco: Elsevier, 2005.

Kent, Steven L. *The Ultimate History Of Video Games*. Primera edición. New York: Three Rivers Press, 2001.

Pressman, Roger S. *Ingeniería del Software*. *Un enfoque práctico*. Quinta edición. Madrid: McGRAW-HIL, 2002.

Michael, David; Chen, Sande. *Serious Games: Games That Educate, Train, and Inform*. Primera edición. Boston: Thomson Course Techology PTR.

Sitios y páginas de internet:

Adobe. *Funciones de Adobe Audition CS5*. http://www.adobe.com/es/products/audition/features.html (último acceso: Julio de 2011).

Adobe. *Funciones de Adobe Illustrator CS5*. http://www.adobe.com/es/products/illustrator/features.html (último acceso: Julio de 2011).

Adobe. *Funciones de Adobe Photoshop CS5*. http://www.adobe.com/es/products/photoshop/features.html (último acceso: Julio de 2011).

Adobe. *Página oficial de Adobe Illustrator*. http://www.adobe.com/es/products/illustrator.html.

Adobe. *Página oficial de Adobe Photoshop*. http://www.adobe.com/es/products/photoshop.html.

Adobe. *Sitio oficial de Adobe Audition*. http://www.adobe.com/es/products/audition.html.

Audacity. *Sitio oficial de Audacity*. http://audacity.sourceforge.net/.

Autodesk. *Sitio oficial de Autodesk 3ds Max*. http://usa.autodesk.com/3ds-max/.

Blender. *Funciones de Blender*. http://www.blender.org/features-gallery/features/ (último acceso: Julio de 2011).

Blender. *Sitio oficial de Blender*. http://www.blender.org/.

Epic Games. *Sitio oficial*. http://www.epicgames.com/.

Gamasutra. *Gamasutra's Best of 2009: Top 5 Game Companies*. http://www.gamasutra.com/view/news/26243/Gamasutras_Best_Of_2009_Top_5_Game_Companies.php (último acceso: Junio de 2011).

GIMP. *Sitio oficial*. http://www.gimp.org/.

GNU. *La definición de Software Libre*. http://www.gnu.org/philosophy/free-sw.es.html (último acceso: Julio de 2011).

GNU. *Licencia pública general de GNU*. http://www.gnu.org/copyleft/gpl.html.

GNU. *Sitio oficial*. http://www.gnu.org/home.es.html.

Inkscape. *Sitio oficial de Inkscape*. http://inkscape.org/index.php?lang=es.

Mod-DB. *Lista de Game Engines*. http://www.moddb.com/engines (último acceso: Junio de 2011).

Gerardo Abraham Morales Urrutia, Claudia Esther Nava López, Luis Felipe Fernández Martínez y Mirsha Aarón Rey Corral. «Instituto de Ingeniería y Tecnología (Universidad Autónoma de Ciudad Juárez).» *Procesos de desarrollo para Videojuegos*. Enero-Abril de 2010. http://www2.uacj.mx/IIT/CULCYT/enero-abril2010/7%20Art%204.pdf.

Pons Alfonso, José Vicente. *¿Qué son los "Serious Games"?.* http://www.exelweiss.com/blog/356/serious-games-juegos-serios/ (último acceso: Junio de 2011).

The Wall Street Journal. *The Winners, Category by Category. From Computing Systems to Wireless, the Most Innovative Technologies.* http://online.wsj.com/article/SB10001424052748703904304575497473735761294.html (último acceso: Junio de 2011).

UDK. *Sitio oficial*. http://www.udk.com.

Unity Technologies. *Sobre la compañía*. http://unity3d.com/company/.

Unity Technologies. *Sitio oficial de Unity3D*. http://www.unity3d.com.

V. APARTADOS

APARTADO 1: Plantilla de Scripts

ID	NOMBRE
#1	ControladorMouse

DESCRIPCIÓN

Este es un objeto CONTROLADOR, y lo que hace es controlar los momentos en que el mouse está activado o desactivado, según si el jugador está jugando o no. Este Script habilita o deshabilita a el objeto GUITexture "Mouse_Cursor" y a su Script "MouseCursor" que son los encargados de pintar el cursor del mouse (la "manito") en la pantalla. También habilita y deshabilita el mouse por defecto con la función "Screen.showCursor = false/true;"

CÓDIGO

```
//obtenemos el GUITexture que vamos a deshabilitar o habilitar (cursor del mouse)
var myGUI: GUITexture;
var Script;

function Update () {

	//obtenemos el objeto cámara para acceder a sus variables
	var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
	var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

	// cuando el Player está jugando, activamos el mouse
	if (ScriptCamara.EnJuego==1) {

		Screen.showCursor = false;   //hacemos invisible el mouse por defecto
		Screen.lockCursor = false; //desbloqueamos el mouse por defecto
		Script=myGUI.GetComponent("MouseCursor"); //obtenemos el script a habilitar
		myGUI.enabled=true; //activamos el GUITexture que coloca la manito en la pantalla
		Script.enabled=true; //activamos el script que tiene ese GUITexture
```

```
        }

        // cuando el Player no está jugando, desactivamos el mouse
        if (ScriptCamara.EnJuego==0) {

                Screen.showCursor = false;   //hacemos invisible el mouse por defecto
                Screen.lockCursor = true; //bloqueamos el mouse por defecto
                Script=myGUI.GetComponent("MouseCursor"); //obtenemos el script a deshabilitar
                myGUI.enabled=false; //desactivamos el GUITexture que coloca la manito en la pantalla
                Script.enabled=false; //desactivamos la manito (cursor)
        }
}
```

ID	NOMBRE
#2	MouseCursor

DESCRIPCIÓN

Permite colocar un sprite 2D como cursor de mouse

CÓDIGO

```
var normalCursor : Texture2D;        // The texture for when the cursor isn't near a screen edge
var leftCursor : Texture2D;          // The texture for the cursor when it's at the left edge
var rightCursor : Texture2D;         // Ditto, right edge
var upCursor : Texture2D;            // Top edge
var downCursor : Texture2D;          // ...And bottom edge
var nativeRatio = 1.3333333333333;   // Aspect ratio of the monitor used to set up GUI elements
private var lastPosition : Vector3;  // Where the mouse position was last
var normalAlpha = .5;                // Normal alpha value of the cursor ... .5 is full
var fadeTo = .2;                     // The alpha value the cursor fades to if not moved
var fadeRate = .22;                  // The rate at which the cursor fades
private var cursorIsFading = true;   // Whether we should fade the cursor
private var fadeValue : float;

// Scale the cursor so it should look right in any aspect ratio, and turn off the OS mouse pointer
function Start() {
   // Slightly weird but necessary way of forcing float evaluation
```

```
    var currentRatio : float = (0.0 + Screen.width) / Screen.height;
    transform.localScale.x *= nativeRatio / currentRatio;
    Screen.showCursor = false;
    fadeValue = normalAlpha;
    lastPosition = Input.mousePosition;
}

function Update() {

    var mousePos = Input.mousePosition;
    // If the mouse has moved since the last update
    if (mousePos != lastPosition) {
        lastPosition = mousePos;
        // Get mouse X and Y position as a percentage of screen width and height
        MoveMouse(mousePos.x/Screen.width, mousePos.y/Screen.height);
    }
    // Fade the alpha of the cursor
    if (cursorIsFading) {
        guiTexture.color.a = fadeValue;
        fadeValue -= fadeRate * Time.deltaTime;
        if (fadeValue < fadeTo) {
            fadeValue = fadeTo;
            cursorIsFading = false;
        }
    }
}

function MoveMouse(mousePosX : float, mousePosY : float) {
    // Make the cursor solid, and set fading to start in case mouse movement stops
    guiTexture.color.a = fadeValue = normalAlpha;
    guiTexture.texture = normalCursor;
    cursorIsFading = true;

    // If the mouse is on a screen edge, first make sure the cursor doesn't go off the screen, then give it the
appropriate cursor
```

```
    if (mousePosX < .005) {
        mousePosX = .005;
        guiTexture.texture = leftCursor;
    }
    if (mousePosX > .995) {
        mousePosX = .995;
        guiTexture.texture = rightCursor;
    }
    if (mousePosY < .005) {
        mousePosY = .005;
        guiTexture.texture = downCursor;
    }
    if (mousePosY > .995) {
        mousePosY = .995;
        guiTexture.texture = upCursor;
    }

    transform.position.x = mousePosX;
    transform.position.y = mousePosY;
}
```

ID	NOMBRE
#3	ParticleSpiralEffect

DESCRIPCIÓN

Permite realizar un efecto de partículas en espiral 3D

CÓDIGO

```
using UnityEngine;
using System.Collections;

public struct SpiralSettings {
    public int numArms;        // number of spiral arms
    public int numPPA;             // number of particles per arm
    public float partSep;         // separation between particles
    public float turnDist;    // distance between spiral turns
    public float vertDist;      // vertical turn distance
```

```
    public float originOffset;  // size of hole in middle of spiral
    public float turnSpeed;  // speed that spiral rotates.
    public float fade;       // fade particles along the arms
    public float size;       // change particle size along arms
}

public class ParticleSpiralEffect : MonoBehaviour
{
    public Transform particleEffect;
    public int numberOfArms = 1;
    public int particlesPerArm = 200;
    public float particleSeparation = 0.05f;
    public float turnDistance = 0.5f;
    public float verticalTurnDistance = 0;
    public float originOffset = 0.0f;
    public float turnSpeed = 0;
    public float fadeValue = 0;
    public float sizeValue = 0;
    public int numberOfSpawns = 9999999;
    public float spawnRate = 5.0f;

    // These constants define the min and max values used for the randomizeEffect method.
    public const int Min_numArms = 1;
    public const int Max_numArms = 20;
    public const int Min_numPPA = 1;
    public const int Max_numPPA = 800;
    public const float Min_partSep = -1.0f;
    public const float Max_partSep = 1.0f;
    public const float Min_turnDist = -1.0f;
    public const float Max_turnDist = 1.0f;
    public const float Min_vertDist = -1.0f;
    public const float Max_vertDist = 1.0f;
    public const float Min_originOffset = -10.0f;
    public const float Max_originOffset = 10.0f;
    public const float Min_turnSpeed = -200.0f;
```

```
    public const float Max_turnSpeed = 200.0f;
    public const float Min_fade = 0.0f;
    public const float Max_fade = 1.0f;
    public const float Min_size = -0.1f;
    public const float Max_size = 0.1f;

    // Time at which the last spawn occurred.  Defaults to a "smallish" number
    // so the first effect appears more or less immediately.
    private float timeOfLastSpawn = -1000.0f;

    // Count of effects spawned so far.
    private int spawnCount = 0;

    // Total number of particles.
    private int totParticles;

    // The settings for the effect as set when the effect is first created,
    // i.e. the default settings.
    private SpiralSettings defaultSettings;

    public void Start() {
        // Record the current Inspector settings as the defaults.
        defaultSettings = getSettings();
    }

   void SpawnEffect () {
        // Instantiate the effect prefab.
        Transform effectObject = Instantiate(particleEffect, this.transform.position, this.transform.rotation) as
Transform;

        // Parent the new effect to this script's transform.
        effectObject.parent = this.gameObject.transform;

        // Get the particle emitter from the new effect object.
        ParticleEmitter emitter = effectObject.GetComponent<ParticleEmitter>();
```

```
// Make sure autodestruct is on so that dead particles systems get destroyed.
ParticleAnimator animator = emitter.transform.GetComponent<ParticleAnimator>();
if (animator != null)
    animator.autodestruct = true;

// Generate the particles.
emitter.Emit(numberOfArms * particlesPerArm);

// Extract the particles from the created emitter.
Particle[] p = emitter.particles;

// The rotation angle, in radians, between the arms of the spiral.
float armSpacing = (2 * Mathf.PI) / numberOfArms;

// Loop thru the arms...
for (int j=0; j<numberOfArms; j++) {

        float r = 0;
        float theta = 0;
        float armRotation = j*armSpacing;

    // Loop thru the particles for this arm and place them.
    for (int i = 0;  i < particlesPerArm;  i++) {

        // This particle's index.
        int pX = j * particlesPerArm + i;

        // This is the equation for our spiral in polar coords....
        r = originOffset + turnDistance * theta;

        // All particles are positioned wrt the local transform origin.
        Vector3 newPos = effectObject.localPosition;
        //Vector3 newPos = effectObject.position;
```

```
            // Convert to Cartesian coords...
            newPos.x = newPos.x + r * Mathf.Cos(theta);
            newPos.z = newPos.z + r * Mathf.Sin(theta);

            // Rotate the particle about the origin to our desired position for this arm.
            float x = newPos.x * Mathf.Cos(armRotation) + newPos.z * Mathf.Sin(armRotation);
            float z = -newPos.x * Mathf.Sin(armRotation) + newPos.z * Mathf.Cos(armRotation);
            newPos.x = x;
            newPos.z = z;

            // Introduce the desired vertical offset.
            newPos.y = newPos.y + i * verticalTurnDistance;

            // Update the particle position.
            if (emitter.useWorldSpace)
               newPos = transform.TransformPoint(newPos);
            p[pX].position = newPos;

            // Bump to our next point on the curve.
            theta += particleSeparation;

            // Set particle lifetime to account for specified fade.
            p[pX].energy = p[pX].energy - i * fadeValue;

            // Set particle size to account for specified variation.
            p[pX].size = p[pX].size - i * sizeValue;

         }
      }
      // Update the actual particles.
      emitter.particles = p;
   }

   void Update() {
      // Spin the entire particle effect.
```

```
    this.transform.Rotate(this.transform.up * Time.deltaTime * (-turnSpeed), Space.World);
}


void LateUpdate()
{
    // Check to see if it's time to spawn a new particle system.
    float timeSinceLastSpawn = Time.time - timeOfLastSpawn;
    if (timeSinceLastSpawn >= spawnRate && spawnCount < numberOfSpawns) {
        SpawnEffect();
        timeOfLastSpawn = Time.time;
        spawnCount++;
    }
}

// Return the current settings for the effect.
public SpiralSettings getSettings() {

    SpiralSettings result;

    result.numArms = numberOfArms;
    result.numPPA = particlesPerArm;
    result.partSep = particleSeparation;
    result.turnDist = turnDistance;
    result.vertDist = verticalTurnDistance;
    result.originOffset = originOffset;
    result.turnSpeed = turnSpeed;
    result.fade = fadeValue;
    result.size = sizeValue;

    return result;
}

// Reset the effect to use the specified settings.
// Except for the killCurrent option, this will only effect the
```

```
// appearance of future spawns.
public SpiralSettings resetEffect(bool killCurrent, SpiralSettings settings) {

    // If requested, destroy all current spawns of this effect.
    if (killCurrent) {
        killCurrentEffects();
    }

    // Assign the new settings and then spawn a new effect with these settings.
    numberOfArms = settings.numArms;
    particlesPerArm = settings.numPPA;
    particleSeparation = settings.partSep;
    turnDistance = settings.turnDist;
    verticalTurnDistance = settings.vertDist;
    originOffset = settings.originOffset;
    turnSpeed = settings.turnSpeed;
    fadeValue = settings.fade;
    sizeValue = settings.size;

    SpawnEffect();
    timeOfLastSpawn = Time.time;
    spawnCount++;

    return getSettings();
}

// Reset the particle effect to its Inspector established defaults.
public SpiralSettings resetEffectToDefaults(bool killCurrent) {

    return resetEffect(killCurrent, defaultSettings);
}

// Randomize the settings and return the new values.
public SpiralSettings randomizeEffect(bool killCurrent) {
```

```
    // If requested, destroy all current spawns of this effect.
    if (killCurrent) {
        killCurrentEffects();
    }

    // Assign the new random settings and then spawn a new effect with these settings.
    numberOfArms = Random.Range(Min_numArms, Max_numArms+1);
    particlesPerArm =   Random.Range(Min_numPPA, Max_numPPA+1);
    particleSeparation =  Random.Range(Min_partSep, Max_partSep);
    turnDistance = Random.Range(Min_turnDist, Max_turnDist);
    verticalTurnDistance = Random.Range(Min_vertDist, Max_vertDist);
    originOffset = Random.Range(Min_originOffset, Max_originOffset);
    turnSpeed = Random.Range(Min_turnSpeed, Max_turnSpeed);
    fadeValue = Random.Range(Min_fade, Max_fade);
    sizeValue = Random.Range(Min_size, Max_size);

    SpawnEffect();
    timeOfLastSpawn = Time.time;
    spawnCount++;

    return getSettings();
}

// Kill all current spawns of the effect.
private void killCurrentEffects() {

    // Loop thru the particle emitter children of this object.  Each one is a particle effect system
    // we want to destroy.
    ParticleEmitter[] emitters = this.transform.GetComponentsInChildren<ParticleEmitter>();
    foreach (ParticleEmitter emitter in emitters) {
        Debug.Log("resetEffect killing: " + emitter.name);
        // Make sure autodestruct is on.
        ParticleAnimator animator = emitter.transform.GetComponent<ParticleAnimator>();
        if (animator != null)
            animator.autodestruct = true;
```

```
        // Now loop thru the particles and set their energies to a small number.  The effect will
        // subsequently autodestruct.  I originally tried setting the energy to zero, but in that
        // case they did *not* autodestruct.
        // I originally tried simply doing a Destroy on the emitter, but got threatening runtime messages.
        Particle[] p  = emitter.particles;
        for (int i=0; i < p.Length;  i++) {
           p[i].energy = 0.1f;
        }
        emitter.particles = p;
     }
  }
}
```

ID	NOMBRE
#4	Script_BackgroundCam
DESCRIPCIÓN	
Permite mover el Fondo del juego para crear un efecto de que se está “navegando” por el espacio	
CÓDIGO	

```
var angulo:int;
private var x;
private var y;

function Start () {

        moverfondo();

}



function moverfondo(){
        //Haremos que el fondo se mueva casi circularmente

        var fondo : GameObject = GameObject.Find("Background/BackgroundImage");
        var posicionX=fondo.transform.position.x; // obtenemos la posicion X del background
        var posicionY=fondo.transform.position.y; // obtenemos la posicion Y del background
```

```
/*---------------------------------------------------------------------------------------------
EL MOVIMIENTO CIRCULAR SE DEFINE COMO:

y = r * seno(angulo)   y     x = r * coseno (angulo)

En donde r es el RADIO y si definimos un angulo que vaya creciendo con el tiempo
entonces tendremos un movimiento circular uniforme
---------------------------------------------------------------------------------------------*/
        while (angulo>=0) {
                angulo=angulo + 1; //angulo que crece con el tiempo

                y = 0.002 * Mathf.Sin(angulo);
                x = 0.002 * Mathf.Cos(angulo); //el /5 hace que el movimiento no sea uniforme
                //Debug.Log("ANGULOOOOO " + angulo);
                //Debug.Log("x = " + x + "     y = " + y);



                fondo.transform.position.x = posicionX + x; //aplicamos el movimiento
                fondo.transform.position.y = posicionY + y; //aplicamos el movimiento
                yield WaitForSeconds(0.04); //este tiempo es importante
        }
}
```

ID	NOMBRE
#5	Script_JuegoTerminado
DESCRIPCIÓN	
Coloca una pantalla de “Juego Terminado” cuando el jugador pierde el nivel (Game Over)	
CÓDIGO	

```
var fadeDuration:float=0.4;
private var timeLeft:float=0.5;

function Update () {
```

```
        //accedemos al objeto  "Main Camera" y a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);



        if (ScriptCamara.varGUI_JuegoTerminado==1) {
                        fade(true);
                } else {
                        fade(false); }
}

function fade(direction:boolean){
  var alpha;
  if (direction){
    if (guiTexture.color.a < 0.5){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=0.5-(alpha/2);
    } else {
      timeLeft = fadeDuration;
    }
  } else {
    if (guiTexture.color.a > 0){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=alpha/2;
    } else {
      timeLeft = fadeDuration;
    }
  }
}
```

ID	NOMBRE
#6	Script_palabra_amarillo
DESCRIPCIÓN	

Coloca la palabra “amarillo” cuando el jugador pasa sobre la esfera amarilla

CÓDIGO

```
var fadeDuration:float=0.4;
private var timeLeft:float=0.5;

function Update () {

        //accedemos al objeto  "Main Camera" y a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

        if (ScriptCamara.palabra_amarillo==true) {
                        fade(true);
                } else {
                        fade(false); }
}

function fade(direction:boolean){
  var alpha;
  if (direction){
    if (guiTexture.color.a < 0.5){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=0.5-(alpha/2);
    } else {
      timeLeft = fadeDuration;
    }
  } else {
    if (guiTexture.color.a > 0){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=alpha/2;
    } else {
      timeLeft = fadeDuration;
    }
```

```
  }
}
```

ID	NOMBRE
#7	Script_palabra_rojo

DESCRIPCIÓN

Coloca la palabra “rojo” cuando el jugador pasa sobre la esfera roja

CÓDIGO

```
var fadeDuration:float=0.4;
private var timeLeft:float=0.5;

function Update () {

        //accedemos al objeto  "Main Camera" y a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);



        if (ScriptCamara.palabra_rojo==true) {
                        fade(true);
                } else {
                        fade(false); }
}

function fade(direction:boolean){
  var alpha;
  if (direction){
    if (guiTexture.color.a < 0.5){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=0.5-(alpha/2);
    } else {
      timeLeft = fadeDuration;
    }
  } else {
```

```
    if (guiTexture.color.a > 0){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=alpha/2;
    } else {
      timeLeft = fadeDuration;
    }
  }
}
```

ID	NOMBRE
#8	Script_palabra_verde

DESCRIPCIÓN

Coloca la palabra “verde” cuando el jugador pasa sobre la esfera verde

CÓDIGO

```
        var fadeDuration:float=0.4;
private var timeLeft:float=0.5;

function Update () {

        //accedemos al objeto  "Main Camera" y a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

        if (ScriptCamara.palabra_verde==true) {
                        fade(true);
                } else {
                        fade(false); }
}

function fade(direction:boolean){
  var alpha;
  if (direction){
    if (guiTexture.color.a < 0.5){
      timeLeft = timeLeft - Time.deltaTime;
```

```
        alpha = (timeLeft/fadeDuration);
        guiTexture.color.a=0.5-(alpha/2);
      } else {
        timeLeft = fadeDuration;
      }
    } else {
      if (guiTexture.color.a > 0){
        timeLeft = timeLeft - Time.deltaTime;
        alpha = (timeLeft/fadeDuration);
        guiTexture.color.a=alpha/2;
      } else {
        timeLeft = fadeDuration;
      }
    }
}
```

ID	NOMBRE
#9	ScriptBarraProgreso

DESCRIPCIÓN

Coloca la barra de progreso que indica la cantidad de partidas ganadas en el nivel

CÓDIGO

```
var barDisplay : float = 0;
var pos : Vector2 = new Vector2(283,54);
var size : Vector2 = new Vector2(283,54);
var progressBarEmpty : Texture2D;
var progressBarFull : Texture2D;

function OnGUI()
{

  // draw the background:
    GUI.BeginGroup (new Rect (pos.x, pos.y, size.x, size.y));
        GUI.DrawTexture (Rect (0,0, size.x, size.y),progressBarEmpty);
```

```
        // draw the filled-in part:
        GUI.BeginGroup (new Rect (0, 0, size.x * barDisplay, size.y));
            GUI.DrawTexture (Rect (0,0, size.x, size.y),progressBarFull);
        GUI.EndGroup ();

    GUI.EndGroup ();
}

function Update()
{
}
```

ID	NOMBRE
#10	ScriptBolaAmarilla

DESCRIPCIÓN

Este script controla la esfera amarilla (objeto “Amarillo”) y su interacción con el jugador. Le da una rotación constante, chequea si el jugador hace click sobre la esfera, realiza diferentes efectos (partículas y flechas), etc.

CÓDIGO

```
//OnMouseDown es llamado se hace click sobre la esfera
function OnMouseDown(){

        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
        var PartInteraccionAmarillo : GameObject =
GameObject.Find("ParticulasAnim/PartInteraccionAmarillo"); //efecto particula

        Debug.Log("El jugador pulsó el color amarillo!");

        //yield WaitForSeconds (0.1);
        ScriptCamara.sonidoAmarillo.Play(); //reproducimos un sonido

        //hacemos una animacion de particulas (tipo explosión)
        PartInteraccionAmarillo.particleEmitter.emit = true;
        yield WaitForSeconds (0.1);
```

```
        PartInteraccionAmarillo.particleEmitter.emit = false;

        // llamamos a la función que guardará el color en el arreglo del jugador
        ScriptCamara.EsferaPulsada("Amarillo");
}

//cuando el mouse está sobre la esfera, mostraremos unas flechitas
function OnMouseOver(){
        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
        var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
        ScriptGUI_Flechas.flechas_activadas="activadas_amarillo";
        ScriptCamara.palabra_amarillo=true; //mostramos la palabra AMARILLO
}

//cuando el mouse NO está sobre la esfera, no mostramos nada
function OnMouseExit(){
        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
        var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
        ScriptCamara.palabra_amarillo=false; // sacamos la palabra AMARILLO
        ScriptGUI_Flechas.flechas_activadas="desactivadas";
}


function Update(){
        //aplicamos una rotacion a la esfera
        transform.Rotate(0,Time.deltaTime*10,0,Space.World);

}
```

ID	NOMBRE
#11	ScriptBolaRoja

DESCRIPCIÓN

Este script controla la esfera roja (objeto “Rojo”) y su interacción con el jugador. Le da una rotación constante, chequea si el jugador hace click sobre la esfera, realiza diferentes efectos (partículas y flechas), etc.

CÓDIGO

```
//OnMouseDown es llamado se hace click sobre la esfera
function OnMouseDown(){

	//obtenemos el objeto cámara para acceder a sus variables
	var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
	var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
	var PartInteraccionRojo : GameObject = GameObject.Find("ParticulasAnim/PartInteraccionRojo");
//efecto particula

	Debug.Log("El jugador pulsó el color Rojo!");

	ScriptCamara.sonidoRojo.Play(); //reproducimos un sonido

	//hacemos una animacion de particulas (tipo explosión)
	PartInteraccionRojo.particleEmitter.emit = true;
	yield WaitForSeconds (0.1);
	PartInteraccionRojo.particleEmitter.emit = false;

	// llamamos a la función que guardará el color en el arreglo del jugador
	ScriptCamara.EsferaPulsada("Rojo");
}

//cuando el mouse está sobre la esfera, mostraremos unas flechitas
function OnMouseOver(){
	//obtenemos el objeto cámara para acceder a sus variables
	var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
	var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
	var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
	var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
```

```
        ScriptGUI_Flechas.flechas_activadas="activadas_rojo";
        ScriptCamara.palabra_rojo=true; //mostramos la palabra ROJO


}

//cuando el mouse NO está sobre la esfera, no mostramos nada
function OnMouseExit(){
        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
        var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
        ScriptCamara.palabra_rojo=false; // sacamos la palabra ROJO
        ScriptGUI_Flechas.flechas_activadas="desactivadas";
}



function Update(){
        //aplicamos una rotacion a la esfera
        transform.Rotate(0,Time.deltaTime*10,0,Space.World);
}
```

ID	NOMBRE
#12	ScriptBolaVerde

DESCRIPCIÓN

Este script controla la esfera verde (objeto “Verde”) y su interacción con el jugador. Le da una rotación constante, chequea si el jugador hace click sobre la esfera, realiza diferentes efectos (partículas y flechas), etc.

CÓDIGO

```
//OnMouseDown es llamado se hace click sobre la esfera
function OnMouseDown(){


        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
```

```
	var PartInteraccionVerde : GameObject = GameObject.Find("ParticulasAnim/PartInteraccionVerde");
//efecto particula

	Debug.Log("El jugador pulsó el color Verde!");

	ScriptCamara.sonidoVerde.Play(); //reproducimos un sonido

	//hacemos una animacion de particulas (tipo explosión)
	PartInteraccionVerde.particleEmitter.emit = true;
	yield WaitForSeconds (0.1);
	PartInteraccionVerde.particleEmitter.emit = false;

	// llamamos a la función que guardará el color en el arreglo del jugador
	ScriptCamara.EsferaPulsada("Verde");
}


//cuando el mouse está sobre la esfera, mostraremos unas flechitas
function OnMouseOver(){
	//obtenemos el objeto cámara para acceder a sus variables
	var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
	var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
	var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
	var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
	ScriptGUI_Flechas.flechas_activadas="activadas_verde";
	ScriptCamara.palabra_verde=true; //mostramos la palabra VERDE

}

//cuando el mouse NO está sobre la esfera, no mostramos nada
function OnMouseExit(){
	//obtenemos el objeto cámara para acceder a sus variables
	var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
	var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);
	var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
```

```
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);
        ScriptCamara.palabra_verde=false; // sacamos la palabra VERDE
        ScriptGUI_Flechas.flechas_activadas="desactivadas";
}

function Update () {
        //aplicamos una rotacion a la esfera
        transform.Rotate(0,Time.deltaTime*10,0,Space.World);
}
```

ID	NOMBRE
#13	ScriptCam

DESCRIPCIÓN

Este es el script principal que maneja a todo el juego, desde aquí se llama a las acciones principales y se mantiene la estructura o bucle de juego; se da el comportamiento y lógica general.

CÓDIGO

```
private var AnimActivada:int; //cuando es 1 desactivamos al jugador, cuando es 0 se puede jugar
private var varGUI_Atencion:int; //determinará el momento en el que se coloca el objeto GUI
private var varGUI_JuegoGanado:int; //determinará el momento en el que se coloca el objeto GUI
private var varGUI_JuegoTerminado: int; // determinará el momento en el que se coloca el objeto GUI
private var varGUI_TuTurno:int; //determinará el momento en el que se coloca el objeto GUI
private var varGUI_Fallaste:int; //determinará el momento en el que se coloca el objeto GUI
private var varGUI_BienHecho:int; //determinará el momento en el que se coloca el objeto GUI
private var CantPartidas: int = 0; //variable que llevará la cantidad de partidas ganadas
private var EstadoPartida: int; // 0 si sigue en carrera, 1 si la perdió
private var Interaccion: int; // 0 dice que el jugador no ha tenido interacción con las esferas, 1 es cuando el jugador clickeó alguna esfera
private var CantInteracciones: int; // nos indica la cantidad de clicks que el jugador hace en las esferas
private var varGUI_JuegoPausado:int; //es para activar o desactivar el cartel de pausa
private var varGUI_SalirDelJuego:int; // es para activar o desactivar el cartel de "esta seguro que desea salir?"
private var EstadoPausado_y_PorSalir:int; // es cuando el jugador esta en la pantalla de "está seguro que desea salir??"
private var palabra_amarillo:boolean; ////determinará el momento en el que se coloca el objeto GUI
private var palabra_verde:boolean; ////determinará el momento en el que se coloca el objeto GUI
private var palabra_rojo:boolean; ////determinará el momento en el que se coloca el objeto GUI
```

```
var sonidoVerde: AudioSource; //sonido de interaccion sobre la esfera verde
var sonidoRojo: AudioSource; //sonido de interaccion sobre la esfera roja
var sonidoAmarillo: AudioSource; //sonido de interaccion sobre la esfera amarilla
var sonidoComienzaSecuencia: AudioSource; //sonido para el cartelito "atención, con mienza la secuencia!"
var sonidoBienHecho: AudioSource; //sonido para el cartelito "Bien hecho!"
var sonidoFallaste: AudioSource; //sonido para el cartelito "fallaste"
var musicaJuegoGanado: AudioSource; //musica que se escucha cuando se gana el juego
var musicaJuego: AudioSource; //musica que se escucha cuando se inicia el juego
var musicaJuegoTerminado: AudioSource; //música para el Game Over


// La siguiente variable determina:
// 0 cuando está realizando algun tipo de animación o está en Pausa
// 1 cuando el jugador está jugando
static var EnJuego:int;

// Variable que sirve para pausar el juego ( ver funcion PauseGame() )
static var ispaused:boolean=false;

//arreglos para manejar los datos que devuelve la funcion CrearSecuencia ( ) del Script
"SecuenciaComputadora"
private var ArregloObtenido = new Array ();

// creamos un arreglo para meter la secuencia que obttuvimos en el paso anterior
private var ArregloComputadora = new Array ();

//arreglo para manejar los datos de la secuencia del jugador
private var ArregloJugador = new Array ();
private var Contador : int; //contador que servirá para almacenar datos en el ArregloJugador

static var Vidas:int;


function Start () {
```

```
        Vidas=3;
        AnimActivada=1;
        varGUI_Atencion=0;
        varGUI_JuegoTerminado=0;
        varGUI_JuegoGanado=0;
        varGUI_BienHecho=0;
        varGUI_Fallaste=0;
        Contador=0;
        varGUI_JuegoPausado=0;
        varGUI_Reloj=0;
        EstadoPausado_y_PorSalir=0;



        /* OBTENEMOS ALGUNOS OBJETOS */
        //-------------------------------------------------------------------------------------------------------------
-------

        var platillo = GameObject.Find("disco");
        var bolaRoja = gameObject.Find ("disco/Rojo");
        var bolaAmarilla = gameObject.Find ("disco/Amarillo");
        var bolaVerde = gameObject.Find ("disco/Verde");

        //Efecto de Particulas del principio (cuando aparecen las bolas)
        var PartRojo : GameObject = GameObject.Find("disco/ParticulaRojo");
        var PartVerde : GameObject = GameObject.Find("disco/ParticulaVerde");
        var PartAmarillo : GameObject = GameObject.Find("disco/ParticulaAmarillo");

        //Particulas para animar las esferas cuando se crea algun tipo de secuencia o evento
        var PartAnimBolaRoja : GameObject = GameObject.Find("ParticulasAnim/PartAnimBolaRoja");
        var PartAnimBolaVerde : GameObject = GameObject.Find("ParticulasAnim/PartAnimBolaVerde");
        var PartAnimBolaAmarilla : GameObject =
GameObject.Find("ParticulasAnim/PartAnimBolaAmarilla");

        //Particulas para hacer un efecto de explosión (llamado efecto espiral)
        var PartEfectoEspiral : GameObject =
```

```
GameObject.Find("ParticulasAnim/efectoEspiral/Particulas_efectoEspiral1");
        var bombaEspiral = PartEfectoEspiral.GetComponent("ParticleSpiralEffect");

        //obtenemos el objeto GUI_BarraProgreso para acceder a sus variables
        var ObjetoBarraProgreso : GameObject = GameObject.Find("GUI_BarraProgreso");
        var ScriptBarraProgreso = ObjetoBarraProgreso.GetComponent(ScriptBarraProgreso);

        //obtenemos el objeto GUI_Reloj para acceder a sus variables
        var ObjetoReloj : GameObject = GameObject.Find("GUI_Reloj");
        var ScriptReloj = ObjetoReloj.GetComponent(ScriptReloj);

        //obtenemos el objeto FadeInOut para acceder a sus variables
        var ObjetoFade : GameObject = GameObject.Find("FadeInOut");
        var ScriptFade = ObjetoFade.GetComponent(Fadeinout);

        //obtenemos el objeto CeldaWaku
        var CeldaWaku : GameObject = GameObject.Find("celda/CeldaWaku");

        //obtenemos el objeto cámara para acceder a sus variables
        var ObjetoFlechas : GameObject = GameObject.Find("GuiTextures/GUI_Flechas");
        var ScriptGUI_Flechas = ObjetoFlechas.GetComponent(ScriptGUI_Flechas);



        //-------------------------------------------------------------------------------------------------------------------
-------



        //Animación inicial (el player no puede jugar, esta desactivado)
        if (AnimActivada==1){

                bolaRoja.GetComponent(MeshRenderer).enabled = false;
                bolaAmarilla.GetComponent(MeshRenderer).enabled = false;
                bolaVerde.GetComponent(MeshRenderer).enabled = false;
```

```
        //animamos el platillo
        platillo.animation.Play("RotaPlatillo");

        //esperamos hasta que termine la animacion del platillo
        yield WaitForSeconds (3);

        //y hacemos una aimacion de particulas
        PartRojo.particleEmitter.emit = true;
        PartVerde.particleEmitter.emit = true;
        PartAmarillo.particleEmitter.emit = true;

        yield WaitForSeconds (2);

        //colocamos las esferas de color
        bolaRoja.GetComponent(MeshRenderer).enabled = true;
        bolaAmarilla.GetComponent(MeshRenderer).enabled = true;
        bolaVerde.GetComponent(MeshRenderer).enabled = true;

        //desactivamos la animacion de particulas
        PartRojo.particleEmitter.emit = false;
        PartVerde.particleEmitter.emit = false;
        PartAmarillo.particleEmitter.emit = false;

        //la animacion ha finalizado =)
        yield WaitForSeconds (2);

}



// si la animación inicial del nivel ha finalizado, entonces


        // En el nivel fácil deberemos ejecutar 10 partidas, la lógica sería mas o menos así:
```

```
                // 1- Ejecutar la secuencia de la computadora
                // 2- Ejecutar la secuencia del jugador (mientras se comprueba que esté acertando)
                // 3- Si el jugador acierta la secuencia, volver al paso 1 e incrementar la barra de progreso
                // pero si no acierta, mantener la barra de progreso como está, descontar una vida y volver al
paso 1
                // Extra: En todo momento se debe comprobar que el jugador tenga vidas para seguir jugando,
                // además, se debe comprobar la cantidad de partidas ganadas, si llega a 10, completará el
nivel =)

                CantPartidas=1; // comenzaremos en la partida número 1

                while (CantPartidas <= 2)   //MODIFICAR
                {

                        // si ya hicimos las 10 partidas entonces salimos del bucle
                        if (CantPartidas == 11 || Vidas <= 0){  //MODIFICAR
                                break;
                        }
                        // sino, seguimos ejecutando el nivel
                        else {

                                /* SECUENCIA DE LA COMPUTADORA */
                                        //-------------------------------------------------------------------------------
-----------------------------
                                        AnimActivada=1; //vamos a realizar la animación de la secuencia
de la computadora
                                        EnJuego=0; // El jugador espera a que se realice esta animación
para poder jugar ( mouse desactivado )



                                        //colocamos durante 2 segundos el cartelito "ATENCION.
Comienza la Secuencia"
                                        yield WaitForSeconds (1);
                                        varGUI_Atencion=1;
                                        sonidoComienzaSecuencia.Play(); //ponemos un sonido
```

```
                                        ScriptReloj.barDisplay=0; //reiniciamos el reloj
                                        yield WaitForSeconds (2);
                                        varGUI_Atencion=0;

                                        //esperamos un segundo para mostrar la secuencia
                                        yield WaitForSeconds (0.5);

                                        //accedemos a la funcion "CrearSecuencia( )" del Script
"SecuenciaComputadora"
                                        //nos devolverá un arreglo con dos valores: El Arreglo que contiene
la secuencia
                                        //y la cantidad de colores que hay (osea, la longitud del arreglo)

        ArregloObtenido=this.GetComponent(SecuenciaComputadora).CrearSecuencia();

                                        //Le metemos la secuencia obtenida al ArregloComputadora
                                        ArregloComputadora = ArregloObtenido[0];

                                        //Y redimensionamos la longitud de los arreglos
                                        ArregloComputadora.length = ArregloObtenido[1];
                                        ArregloJugador.length=ArregloObtenido[1];

                                        //Líneas de Debug
                                        Debug.Log("Secuencia ---------------------------------------------------- : "
+ ArregloComputadora);
                                        Debug.Log("Longitud del Arreglo ----------------------------------------
- : " + ArregloComputadora.length);

                                        // Ya tenemos el arreglo con la secuencia: ArregloComputadora[]
                                        // y tambien tenemos la cantidad de elementos que hay en el
arreglo: ArregloComputadora.length
                                        // por lo tanto ahora lo único que tenemos que hacer, es animar esa
secuencia para mostrársela al jugador

                                        //vamos recorriendo el arreglo y activando las animaciones  según
```

```
los colores vayamos encontrando
                                        var i;
                                        i=0;

                                        while(i <= ArregloComputadora.length-1){

                                                //obtenemos el color de esa posicion del arreglo
                                                var Color : String = ArregloComputadora[i];
                                                Debug.Log("---------------------------------------- : " + Color);

                                                // y según el color que sea, lo animamos :)
                                                if (Color=="Rojo"){
                                                        palabra_rojo=true;
                                                        PartAnimBolaRoja.particleEmitter.emit = true;
                                                        sonidoRojo.Play();

        ScriptGUI_Flechas.flechas_activadas="activadas_rojo"; //le activamos las flechas verdes alrededor de
la esfera
                                                }
                                                if (Color=="Verde"){
                                                        palabra_verde=true;
                                                        PartAnimBolaVerde.particleEmitter.emit = true;
                                                        sonidoVerde.Play();

        ScriptGUI_Flechas.flechas_activadas="activadas_verde"; //le activamos las flechas verdes alrededor
de la esfera
                                                }
                                                if (Color=="Amarillo"){
                                                        palabra_amarillo=true;
                                                        PartAnimBolaAmarilla.particleEmitter.emit =
true;
                                                        sonidoAmarillo.Play();

        ScriptGUI_Flechas.flechas_activadas="activadas_amarillo"; //le activamos las flechas verdes alrededor
de la esfera
```

```
                                        }

                                        yield WaitForSeconds (0.5);
                                        PartAnimBolaRoja.particleEmitter.emit = false;
                                        PartAnimBolaVerde.particleEmitter.emit = false;
                                        PartAnimBolaAmarilla.particleEmitter.emit = false;

                                        ScriptGUI_Flechas.flechas_activadas="desactivadas";
//desactivamos las flechas
                                        yield WaitForSeconds (1);
                                        palabra_amarillo=false;
                                        palabra_rojo=false;
                                        palabra_verde=false;
                                        i=i+1; //siguiente posición del arreglo
                                }
                                //-------------------------------------------------------------------------------
------------------------------




                                /* SECUENCIA DEL JUGADOR (ahora es el turno del jugador de
colocar la secuencia correcta) */
                                //-------------------------------------------------------------------------------
------------------------------
                                CantInteracciones=0; //reiniciamos la cantidad de clicks que hace el
jugador en las esferas para esta partida
                                Contador=0;
                                var Continua; // variable que define si se continúa jugando o si se
rompe el bucle para reiniciar la partida y descontar 1 vida



                                //colocamos durante 2 segundos el cartelito "TU TURNO !"
                                yield WaitForSeconds (1);
                                varGUI_TuTurno=1;
                                yield WaitForSeconds (1.5);
```

```
                varGUI_TuTurno=0;
                EstadoPartida=0;
                Interaccion=0;
                EnJuego=1; // El jugador comienza a jugar su partida :) (se activa el
mouse)
                AnimActivada=0; //En este punto ya no hay mas animaciones
                varGUI_Reloj=1; //Comienza a contar el tiempo!



                // Comprobamos si el Jugador Falla (hasta que se acabe el reloj), si
falló, entonces se reinicia la partida y se resta una vida
                var fallado: int;

                fallado=0;
                var x : int;
                var sonDistintos: int;
                x=0;

                // en este pequeño while comprobamos si el jugador logra acertar en
toda la secuencia
                // o si se equivoca antes de terminarla
                while (ScriptReloj.barDisplay < 1) { // comprobamos hasta que se
acabe el tiempo
                        if (CantInteracciones < ArregloJugador.length) {
                                yield WaitForSeconds (0.5);
                                if (EstadoPartida==1) { //si el jugador se equivocó
                                        fallado=1;
                                        Debug.Log("El jugador ha fallado");
                                        break; //sale de este pequeño bucle
                                }
                                ScriptReloj.barDisplay=ScriptReloj.barDisplay +
0.05; //disminuímos el tiempo del Reloj
                        } else
                                {
                                        break; // el usuario viene acertó en toda
```

```
la secuencia
                                                }
                                        }

                                        if (fallado==1) { //si el jugador se equivoó antes de terminar la
secuencia
                                                Vidas--;
                                                AnimActivada=1; //al perder una vida, comenzaremos una
animación
                                                Debug.Log("VIDAS
??????????????????????????????????????" + Vidas);
                                                EnJuego=0; //deshabilitamos el mouse
                                                //colocamos el cartelito de "Fallaste!"
                                                varGUI_Fallaste=1;
                                                sonidoFallaste.Play(); //reproduciomos un sonido
                                                yield WaitForSeconds (1.5);
                                                varGUI_Fallaste=0;
                                                continue; //reinicia la partida
                                        }



                                        //la secuencia del jugador es igual a la secuencia de la
computadora???
                                        //sonDistintos=1 significa que los arreglos son diferentes (por ende
las secuencias son distintas)
                                        if (ArregloJugador.length==3){
                                                        sonDistintos=0;
                                                        while (x <3 ) {
                                                                if(ArregloJugador[x] !=
ArregloComputadora[x]) { sonDistintos=1; }

                                                                x++;
                                                        }
                                                }
                                        if (ArregloJugador.length==2){
                                                sonDistintos=0;
```

```
                                        while (x < 2 ) {
                                                if(ArregloJugador[x] !=
ArregloComputadora[x]) { sonDistintos=1; }
                                                x++;
                                        }
                                }



                        // si el jugador no pulsó ninguna esfera de color mientras estaba
jugando, pierde la partida
                        if (Interaccion==0) {
                                Debug.Log("El jugador no ha pulsado ninguna esfera!!!!");
                                Vidas--;
                                AnimActivada=1; //al perder una vida, comenzaremos una
animación
                                Debug.Log("VIDAS
??????????????????????????????????????" + Vidas);
                                EnJuego=0;
                                //colocamos el cartelito de "Fallaste!"
                                varGUI_Fallaste=1;
                                sonidoFallaste.Play(); //reproduciomos un sonido
                                yield WaitForSeconds (1.5);
                                varGUI_Fallaste=0;
                                continue; //reinicia la partida
                        }



                        // si el jugador pulsó alguna esfera de color mientras estaba jugando
y NO acertó en toda la secuencia
                        // por ejemplo si pulsa la primer esfera correcta pero se le acaba el
tiempo y no vuelve a pulsar otra
                        if (Interaccion==1 && sonDistintos==1) {
                                Debug.Log("El jugador acerto alguna/s esfera/s pero se le
acabó el tiempo");
                                Vidas--;
```

```
AnimActivada=1; //al perder una vida, comenzaremos una
animación
Debug.Log("VIDAS
??????????????????????????????????????" + Vidas);
EnJuego=0; //deshabilitamos el mouse
//colocamos el cartelito de "Fallaste!"
varGUI_Fallaste=1;
sonidoFallaste.Play(); //reproduciomos un sonido
yield WaitForSeconds (1.5);
varGUI_Fallaste=0;
continue; //reinicia la partida
}

// si el jugador acertó en toda la secuencia
if (Interaccion==1 && sonDistintos==0) {
Debug.Log("Partida ganada
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
// aumentamos la barra de progreso
CantPartidas = CantPartidas + 1;
AnimActivada=1; //al ganar una partida, comenzaremos una
animación

ScriptBarraProgreso.barDisplay=ScriptBarraProgreso.barDisplay+0.1;
//colocamos el cartelito de " Bien hecho!"
varGUI_BienHecho=1;
sonidoBienHecho.Play(); //reproducimos un sonido
yield WaitForSeconds (1.5);
varGUI_BienHecho=0;
}

//-----------------------------------------------------------------------------
-----------------------------


}
```

```
		} // fin bucle


		// Si se le acabaron las vidas al jugador
		if (Vidas == 0) {
			Debug.Log("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^GAME OVER
^^^^^^^^^^^^^^^^^^^^^^^^^^^^");

			//cambiamos la música
			musicaJuego.Stop();
			yield WaitForSeconds (1);
			musicaJuegoTerminado.Play();

			//colocamos un cartelito de "Game over" o "Juego Terminado"
			varGUI_JuegoTerminado=1;


		} else { // Si el nivel finalizó con éxito, se realizará una animación en donde se libera a Waku
			Debug.Log("^^^^^^^^^^^^^^^^^^^WAKU LIBERADO !!! ^^^^^^^^^^^^^^^^^^^");



			//y hacemos una aimacion de particulas
			PartRojo.particleEmitter.emit = true;
			PartVerde.particleEmitter.emit = true;
			PartAmarillo.particleEmitter.emit = true;

			yield WaitForSeconds (2);

			//sacamos las esferas de color
			bolaRoja.GetComponent(MeshRenderer).enabled = false;
			bolaAmarilla.GetComponent(MeshRenderer).enabled = false;
			bolaVerde.GetComponent(MeshRenderer).enabled = false;

			//desactivamos la animacion de particulas
```

```
PartRojo.particleEmitter.emit = false;
PartVerde.particleEmitter.emit = false;
PartAmarillo.particleEmitter.emit = false;

yield WaitForSeconds (2);

//hacemos una súper explosión !
bombaEspiral.enabled=true;

// hacemos un Fade In-Out
ScriptFade.Start();

//hacemos desaparecer la celda
CeldaWaku.active=false;

//esperamos
yield WaitForSeconds (2);

//animamos el platillo
platillo.animation.Play("RotaPlatillo");

//desactivamos la explosión porque sino se va a volver a repetir
bombaEspiral.enabled=false;

//esperamos hasta que termine la animacion del platillo
yield WaitForSeconds (3);

// cambiamos la música
musicaJuego.Stop();
yield WaitForSeconds (1);
musicaJuegoGanado.Play();



// ponemos el cartelito "Felicidades, waku ha sido liberado!"
varGUI_JuegoGanado=1;
```

```
                }
}

/*-------------------------------------------------------------------------------------------------
cuando se pulsa una esfera, se debe guardar el valor del color en el ArregloJugador, para después
poder compararlo con el ArrgeloComputadora. Esta función es llamada desde cada Script adjunto
a cada bola de color (ScriptBolaRoja, ScriptBolaVerde, ScriptBolaAmarilla).
Además, aquí es donde comprobamos si el jugador acertó o se equivocó */

function EsferaPulsada(color){

        if (Contador < ArregloJugador.length) {
                if (color=="Rojo"){
                        ArregloJugador[Contador]="Rojo"; //asignamos un nuevo valor en el arreglo
                        Interaccion=1;
                        CantInteracciones++;
                        //comprobamos si el jugador acertó
                        if (ArregloJugador[Contador]==ArregloComputadora[Contador]){
                                Debug.Log("********************* El jugador Acertó **************");

                        } else {
                                Debug.Log("********************* El jugador NO Acertó
**************");
                                EstadoPartida=1;
                        }
                        Contador++;
                }

                if (color=="Verde"){
                        ArregloJugador[Contador]="Verde"; //asignamos un nuevo valor en el arreglo
                        Interaccion=1;
                        CantInteracciones++;
                        //comprobamos si el jugador acertó
                        if (ArregloJugador[Contador]==ArregloComputadora[Contador]){
```

```
                                Debug.Log("********************* El jugador Acertó **************");

                        } else {
                                Debug.Log("********************* El jugador NO Acertó
**************");
                                EstadoPartida=1;
                        }
                        Contador++;
                }

                if (color=="Amarillo"){
                        ArregloJugador[Contador]="Amarillo"; //asignamos un nuevo valor en el arreglo
                        Interaccion=1;
                        CantInteracciones++;
                        //comprobamos si el jugador acertó
                        if (ArregloJugador[Contador]==ArregloComputadora[Contador]){
                                Debug.Log("********************* El jugador Acertó **************");
                        } else {
                                Debug.Log("********************* El jugador NO Acertó
**************");
                                EstadoPartida=1;
                        }
                        Contador++;
                }


                // cuando el arreglo está lleno es porque el jugador ya realizó su secuencia, por lo tanto no
debemos dejar
                // que pueda seguir haciendo click en las esferas
                if (Contador == ArregloJugador.length) {
                        EnJuego=0; //deshabilitamos al jugador
                }
        }
//-------------------------------------------------------------------------------------------------
}
```

```
function Update () {


	if (Input.GetKeyDown(KeyCode.P)) {
		Debug.Log("LLAMANDO A LA FUNCION PAUSEGAME()");
		PauseGame();
	}

	if (Input.GetKeyDown(KeyCode.Escape)) {    // MODIFICAR
	//if (Input.GetKeyDown(KeyCode.Space)) {
		Salir();
	}

	// Cuando el jugador pulsa Escape, luego se comprueba si sale del juego (ESC) o si regresa al mismo
(N)
	if (EstadoPausado_y_PorSalir==1){

		// Si pulsa "N", vuelve al juego
		if (Input.GetKeyDown(KeyCode.N)) {
			// -------------------------- despausamos el juego ------------------
			Time.timeScale=1;
			if (AnimActivada==1){
				EnJuego=0;
				varGUI_SalirDelJuego=0; //sacamos el cartel
				AudioListener.pause = false; //ponemos el sonido de nuevo
				EstadoPausado_y_PorSalir=0;
			} else {
				EnJuego=1;
				varGUI_SalirDelJuego=0; //sacamos el cartel
				AudioListener.pause = false; //ponemos el sonido de nuevo
				EstadoPausado_y_PorSalir=0;
			}
			//-----------------------------------------------------------------
		}
```

```
                // Si pulsa Escape, sale del juego
                if (Input.GetKeyDown(KeyCode.S)) {
                        Screen.showCursor = true;   //mostramos mouse por defecto
                        Screen.lockCursor = false; //desbloqueamos el mouse por defecto
                        varGUI_SalirDelJuego=0; //sacamos el cartel
                        Time.timeScale=1;
                        Application.LoadLevel("escena2");
                }
        }
}

// Función que nos permite Pausar el juego
function PauseGame(){

        if (EstadoPausado_y_PorSalir != 1){ // si el usuario no ha pulsado anteriormente la tecla ESC (para
salir)
                if(!ispaused)
                {
                        Debug.Log("Juego Pausado. ANIM ACTIVADA ES IGUAL A " + AnimActivada);
                        varGUI_JuegoPausado=1; //ponemos cartel "Juego Pausado"
                        EnJuego=0; //le desactivamos el mouse al jugador
                        Time.timeScale=0;
                        AudioListener.pause = true;
                        ispaused=true;
                }
                else
                {
                        Debug.Log("Juego DESPausado. ANIM ACTIVADA ES IGUAL A " +
AnimActivada);
                        Time.timeScale=1;
                        if (AnimActivada==1){
                                EnJuego=0; }
                        else {
                                EnJuego=1;
```

```
                        }

                        varGUI_JuegoPausado=0; //sacamos el cartel
                        AudioListener.pause = false;
                        ispaused=false;
                }
        }
}

// Función que permite salir del juego pulsando la tecla "Esc"
function Salir (){
        // Si el juego no está en Pausa, salimos
        if (ispaused==false) {
                EstadoPausado_y_PorSalir=1; // ver la funcion Update
                varGUI_SalirDelJuego=1; //mostramos el cartel "Está seguro que deseas Salir ??? "

                // ---------- pausamos el juego ----------
                EnJuego=0;
                Time.timeScale=0;
                AudioListener.pause = true;

                //-------------------------------------------------------------
        }
}
```

ID	NOMBRE
#14	ScriptCelda
DESCRIPCIÓN	
Permite realizar una rotación a la celda de Waku según la posición del mouse en la pantalla	
CÓDIGO	

```
//preparamos unas variables para hacer la animación de rotación
public var VelocidadRotX:int;
public var VelocidadRotY:int;
public var VelocidadRotZ:int;
```

```
function Start (){
VelocidadRotX=0;
VelocidadRotY=0;
VelocidadRotZ=0;
}

function Update () {



/* ---------------------------------ANIMACION DE LA CELDA ----------------------
DIVIDIMOS LA PANTALLA EN 9 PARTES
  --------|--------|-----------
          |        |
    A7    |   A8  |  A9
  --------|--------|----------
           A4          |    A5  | A6
 ---------|--------|----------
          |        |
           A1    |   A2   | A3
 ---------|--------|-----------

Y SEGUN LA ZONA EN DONDE SE ENCUENTRE EL MOUSE SE VA  A GENERAR
UNA ANIMACION DE ROTACIÓN PARA LA CELDA */



          //A1
          if (Input.mousePosition.x <= 335 && Input.mousePosition.y <= 260 ){
                    VelocidadRotX=180;
                    VelocidadRotY=-20;
                    VelocidadRotZ=0;
          }

          //A2
          if (Input.mousePosition.x > 335 && Input.mousePosition.x < 485 && Input.mousePosition.y <= 260
){
```

```
                VelocidadRotX=140;
                VelocidadRotY=0;
                VelocidadRotZ=0;
        }

        //A3
        if (Input.mousePosition.x >= 485 && Input.mousePosition.y <= 260 ){
                VelocidadRotX=180;
                VelocidadRotY=20;
                VelocidadRotZ=0;
        }

        //A4
        if (Input.mousePosition.x <= 335 && Input.mousePosition.y > 260 && Input.mousePosition.y <375){
                VelocidadRotX=0;
                VelocidadRotY=120;
                VelocidadRotZ=120;
        }

        //A5
        if (Input.mousePosition.x > 335 && Input.mousePosition.x <485 && Input.mousePosition.y > 260
&& Input.mousePosition.y <375){
                VelocidadRotX=20;
                VelocidadRotY=20;
                VelocidadRotZ=20;
        }

        //A6
        if (Input.mousePosition.x >= 485 && Input.mousePosition.y > 260 && Input.mousePosition.y <375){
                VelocidadRotX=0;
                VelocidadRotY=-120;
                VelocidadRotZ=120;
        }

        //A7
```

```
        if (Input.mousePosition.x <=335 && Input.mousePosition.y >= 375){
                VelocidadRotX=-180;
                VelocidadRotY=-20;
                VelocidadRotZ=0;
        }

        //A8
        if (Input.mousePosition.x > 335 && Input.mousePosition.x < 485 && Input.mousePosition.y >= 375){
                VelocidadRotX=-140;
                VelocidadRotY=0;
                VelocidadRotZ=0;
        }

        //A9
        if (Input.mousePosition.x >=485 && Input.mousePosition.y >= 375){
                VelocidadRotX=-180;
                VelocidadRotY=20;
                VelocidadRotZ=0;
        }


//Debug.Log("MOUSE X " + Input.mousePosition.x  + "    MOUSE Y" + Input.mousePosition.y);

        //aplicamos la rotacion a la celda
        transform.Rotate(Time.deltaTime*VelocidadRotX,Time.deltaTime*VelocidadRotY,Time.deltaTime*
VelocidadRotZ,Space.World);
}
```

ID	NOMBRE
#15	ScriptGUI_Atencion
DESCRIPCIÓN	
Coloca la pantalla “Atención! Comienza la secuencia”	
CÓDIGO	
//obtenemos el guitexture que vamos a activar o desactivar var Gui : GUITexture ;	

```
function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
if (ScriptCamara.varGUI_Atencion==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_Atencion==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#16	ScriptGUI_BienHecho
DESCRIPCIÓN	
Coloca la pantalla “Bien Hecho!”	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar
var Gui : GUITexture ;

function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
if (ScriptCamara.varGUI_BienHecho==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_BienHecho==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#17	ScriptGUI_Fallaste
DESCRIPCIÓN	
Coloca la pantalla “Fallaste!”	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar
var Gui : GUITexture ;

function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
if (ScriptCamara.varGUI_Fallaste==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_Fallaste==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#18	ScriptGUI_Flechas
DESCRIPCIÓN	
Activa o desactiva las flechas según si el jugador pasa o no sobre una esfera	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar cuando nos subamos encima de la esfera
var GuiFlechas : GUITexture ;

//para colocar las flechitas verdes alrededor de las esferas cuando se coloca el mouse sobre ellas
static var flechas_activadas;

function Start(){
flechas_activadas="desactivadas";
```

```
}

function Update () {
	//Comprobamos si el mouse esta sobre alguna esfera, si es asi, colocamos las flechitas verdes

	if (flechas_activadas=="activadas_rojo"){
		GuiFlechas.enabled=true;
		//seteamos la posicion en la bola roja
		GuiFlechas.pixelInset = Rect (-270, -66, 172, 172);
	}

	if (flechas_activadas=="activadas_amarillo"){
		GuiFlechas.enabled=true;
		//seteamos la posicion en la bola amarilla
		GuiFlechas.pixelInset = Rect (88, -72, 172, 172);
	}

	if (flechas_activadas=="activadas_verde"){
		GuiFlechas.enabled=true;
		//seteamos la posicion en la bola amarilla
		GuiFlechas.pixelInset = Rect (-89, -251, 172, 172);
	}

	if (flechas_activadas=="desactivadas"){
		GuiFlechas.enabled=false;
	}
}
```

ID	NOMBRE
#19	ScriptGUI_JuegoGanado
DESCRIPCIÓN	
Coloca la pantalla “Felicidades, has logrado liberar a Waku!” cuando se gana el juego	
CÓDIGO	

```
var fadeDuration:float=0.4;
private var timeLeft:float=0.5;
```

```
function Update () {

        //accedemos al objeto  "Main Camera" y a sus variables
        var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
        var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);



        if (ScriptCamara.varGUI_JuegoGanado==1) {
                        fade(true);
                } else {
                        fade(false); }
}

function fade(direction:boolean){
  var alpha;
  if (direction){
    if (guiTexture.color.a < 0.5){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=0.5-(alpha/2);
    } else {
      timeLeft = fadeDuration;
    }
  } else {
    if (guiTexture.color.a > 0){
      timeLeft = timeLeft - Time.deltaTime;
      alpha = (timeLeft/fadeDuration);
      guiTexture.color.a=alpha/2;
    } else {
      timeLeft = fadeDuration;
    }
  }
}
```

ID	NOMBRE
#20	ScriptGUI_JuegoPausado
DESCRIPCIÓN	
Coloca la pantalla “Juego Pausado”	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar
var Gui : GUITexture ;

function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
if (ScriptCamara.varGUI_JuegoPausado==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_JuegoPausado==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#21	ScriptGUI_SalirDelJuego
DESCRIPCIÓN	
Coloca la pantalla “¿Salir?” cuando el jugador pulsa la tecla Esc	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar
var Gui : GUITexture ;

function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
```

```
if (ScriptCamara.varGUI_SalirDelJuego==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_SalirDelJuego==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#22	ScriptGUI_TuTurno

DESCRIPCIÓN

Coloca la pantalla “Tu Turno” cuando es el turno del jugador de repetir la secuencia

CÓDIGO

```
//obtenemos el guitexture que vamos a activar o desactivar
var Gui : GUITexture ;

function Update () {
//accedemos al objeto  "Main Camera" y a sus variables
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

// colocamos la animación en el momento adecuado
if (ScriptCamara.varGUI_TuTurno==0){
        Gui.enabled=false; }

if (ScriptCamara.varGUI_TuTurno==1) {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#23	ScriptReloj

DESCRIPCIÓN

Coloca la barra de tiempo

CÓDIGO

```
var barDisplay : float = 0;
var pos : Vector2 = new Vector2(20,40);
```

```
var size : Vector2 = new Vector2(64,34);
var progressBarEmpty : Texture2D;
var progressBarFull : Texture2D;

function OnGUI()
{

  // draw the background:
    GUI.BeginGroup (new Rect (pos.x, pos.y, size.x, size.y));
        GUI.DrawTexture (Rect (0,0, size.x, size.y),progressBarEmpty);

        // draw the filled-in part:
        GUI.BeginGroup (new Rect (0, 0, size.x * barDisplay, size.y));
            GUI.DrawTexture (Rect (0,0, size.x, size.y),progressBarFull);
        GUI.EndGroup ();

    GUI.EndGroup ();

}

function Update()
{
                    //barDisplay = Time.time * 0.1;
}
```

ID	NOMBRE
#24	ScriptVida1
DESCRIPCIÓN	
Gestiona las vidas	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar segun la cantidad de vidas que tengamos
var Gui : GUITexture ;

function Update () {
//accedemos a la variable Vidas del script "ScriptCam" que se encuentra
```

```
//en el objeto "Main Camera"
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

//activamos o desactivamos segun la cantidad de vidas
if (ScriptCamara.Vidas==0){
        Gui.enabled=false; }
else {
        Gui.enabled=true; }
}
```

ID	NOMBRE
#25	ScriptVida2
DESCRIPCIÓN	
Gestiona las vidas	
CÓDIGO	

```
//obtenemos el guitexture que vamos a activar o desactivar segun la cantidad de vidas que tengamos
var Gui : GUITexture ;

function Update () {
//accedemos a la variable Vidas del script "ScriptCam" que se encuentra
//en el objeto "Main Camera"
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

//activamos o desactivamos segun la cantidad de vidas
if (ScriptCamara.Vidas>=2){
        Gui.enabled=true; }
else {
        Gui.enabled=false; }
}
```

ID	NOMBRE
#26	ScriptVida3
DESCRIPCIÓN	

Gestiona las vidas

CÓDIGO

```
//obtenemos el guitexture que vamos a activar o desactivar segun la cantidad de vidas que tengamos
var Gui : GUITexture ;

function Update () {
//accedemos a la variable Vidas del script "ScriptCam" que se encuentra
//en el objeto "Main Camera"
var ObjetoCamara : GameObject = GameObject.Find("Main Camera");
var ScriptCamara = ObjetoCamara.GetComponent(ScriptCam);

//activamos o desactivamos segun la cantidad de vidas
if (ScriptCamara.Vidas==3){
        Gui.enabled=true; }
else {
        Gui.enabled=false; }
}
```

ID	NOMBRE
#27	SecuenciaComputadora

DESCRIPCIÓN

Este Script se encarga de generar la secuencia de la computadora. La secuencia completa se almacenará en un arreglo llamado "Arreglo", y este arreglo será devuelto por la función CrearSecuencia ( ).

CÓDIGO

```
//Declaramos algunas variables
var Arreglo = new Array (); //creo un arreglo donde guardaré la secuencia
private var NumeroDeColor: int = 0; //el numero que corresponde a un color
private var Color : String = ""; // el nombre del color que corresponde al número
private var LongitudArreglo : int;



function CrearSecuencia() { // con esta función crearemos la secuencia de la computadora

        //Generamos un número aleatorio entre 2 y 3, que va a ser el número de unidades que tendrá la
secuencia
```

```
var Unidades: int = Random.Range(2, 4);


// Si la secuencia tiene una sola unidad, entonces
if (Unidades==1){

        // la longitud del arreglo es igual a 1
        LongitudArreglo=1;
        Arreglo.length = LongitudArreglo; //redimensionamos el arreglo


        //generamos un color aleatorio
        NumeroDeColor = GenerarUnidad();

        //Lo almacenamos en un Arreglo
        Color = DevolverColor (NumeroDeColor);
        Arreglo[0]=Color;

        // líneas de debug (depuración)
        Debug.Log("longitud del arreglo: " + LongitudArreglo);
        Debug.Log("numero del color: " +  NumeroDeColor);
        Debug.Log("nombre del color: " +  Color);
        Debug.Log("posicion 0 del arreglo:" + Arreglo[0]);

}

// Si la secuencia tiene dos unidades, entonces
if (Unidades==2){

        // la longitud del arreglo es igual a 2
        LongitudArreglo=2;
        Arreglo.length = LongitudArreglo; //redimensionamos el arreglo

        //-------------------------------------------------------
```

```
            //generamos el primer color aleatorio
            NumeroDeColor = GenerarUnidad();

            //Lo almacenamos en la primera posicion del Arreglo
            Color = DevolverColor (NumeroDeColor);
            Arreglo[0]=Color;

            // líneas de debug (depuración)
            Debug.Log("longitud del arreglo: " + LongitudArreglo);
            Debug.Log("numero del color: " +  NumeroDeColor);
            Debug.Log("nombre del color: " +  Color);
            Debug.Log("posicion 0 del arreglo:" + Arreglo[0]);
            Debug.Log("------------------------------");

            //-------------------------------------------------------

            //generamos el segundo color aleatorio
            NumeroDeColor = GenerarUnidad();

            //Lo almacenamos en la segunda posicion del Arreglo
            Color = DevolverColor (NumeroDeColor);
            Arreglo[1]=Color;

            // líneas de debug (depuración)
            Debug.Log("numero del color: " +  NumeroDeColor);
            Debug.Log("nombre del color: " +  Color);
            Debug.Log("posicion 1 del arreglo:" + Arreglo[1]);


            //-------------------------------------------------------

    }

    // Si la secuencia tiene tres unidades, entonces
    if (Unidades==3){
```

```
// la longitud del arreglo es igual a 2
LongitudArreglo=3;
Arreglo.length = LongitudArreglo; //redimensionamos el arreglo

//-------------------------------------------------------
//generamos el primer color aleatorio
NumeroDeColor = GenerarUnidad();

//Lo almacenamos en la primera posicion del Arreglo
Color = DevolverColor (NumeroDeColor);
Arreglo[0]=Color;

// líneas de debug (depuración)
Debug.Log("longitud del arreglo: " + LongitudArreglo);
Debug.Log("numero del color: " +  NumeroDeColor);
Debug.Log("nombre del color: " +  Color);
Debug.Log("posicion 0 del arreglo:" + Arreglo[0]);
Debug.Log("------------------------------");

//-------------------------------------------------------

//generamos el segundo color aleatorio
NumeroDeColor = GenerarUnidad();

//Lo almacenamos en la segunda posicion del Arreglo
Color = DevolverColor (NumeroDeColor);
Arreglo[1]=Color;

// líneas de debug (depuración)
Debug.Log("numero del color: " +  NumeroDeColor);
Debug.Log("nombre del color: " +  Color);
Debug.Log("posicion 1 del arreglo:" + Arreglo[1]);
Debug.Log("------------------------------");

//-------------------------------------------------------
```

```
                //generamos el tercer color aleatorio
                NumeroDeColor = GenerarUnidad();

                //Lo almacenamos en la tercer posicion del Arreglo
                Color = DevolverColor (NumeroDeColor);
                Arreglo[2]=Color;

                // líneas de debug (depuración)
                Debug.Log("numero del color: " +  NumeroDeColor);
                Debug.Log("nombre del color: " +  Color);
                Debug.Log("posicion 2 del arreglo:" + Arreglo[2]);


                //-------------------------------------------------------

        }

return [Arreglo,LongitudArreglo]; //retornamos el arreglo
}


//Esta función genera un color aleatorio entre los posibles y lo devuelve como int
function GenerarUnidad(){

        // Para tener en cuenta: En éste nivel son 3 colores:
        // 1= Rojo
        // 2=Verde
        // 3=Amarillo

        //Generamos el color aleatorio (1,2 ó 3)
        var Color: int = Random.Range(1, 4);

        return Color;
}
```

```
//Con esta funcion devolvemos el nombre del color correspondiente al número que tiene ese color
function DevolverColor(numerodecolor:int){

var NombreColor : String ="";

	if (numerodecolor==1){
		NombreColor="Rojo";
	}

	if (numerodecolor==2){
		NombreColor="Verde";
	}

	if (numerodecolor==3){
		NombreColor="Amarillo";
	}

	return NombreColor;
}
```

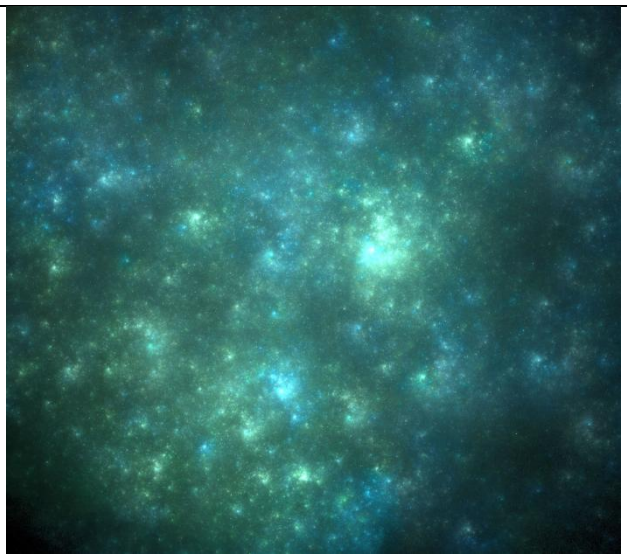
APARTADO 2: Plantilla de Objetos

A continuación podemos ver las Plantillas de Objetos que permiten detallar con exactitud cada objeto del primer nivel (nivel fácil):

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#1	Background	GameObject	Otros_Objetos	-	Background/ Background
IMAGEN					
-					
DESCRIPCIÓN					
Objeto contenedor					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#2	BackgroundCamera	Camera	Cámara	Script_BackgroundCam	Background/BackgroundCamera
IMAGEN					
-					
DESCRIPCIÓN					
Es la cámara que permite que se visualice el Background en 2D					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#3	BackgroundImage	GUITexture	GuiTexture	-	Background/BackgroundImage
IMAGEN					

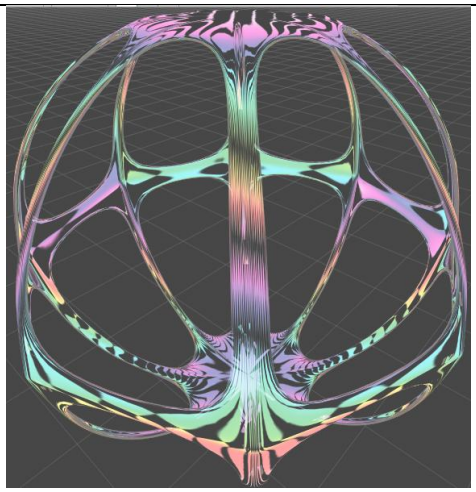


Assets\nivelFacil\Background\Fractal.jpg – 1024x747px

DESCRIPCIÓN
Fondo de estrellas que aparece en el Nivel Fácil

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#4	Celda	GameObject	Otros_Objetos	ScriptCelda	celda
IMAGEN					
-					
DESCRIPCIÓN					
Objeto contenedor. Está asociado a un Script que realiza el movimiento de rotación de la celda de Waku.					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#5	CeldaWaku	GameObject (Blender import)	Objeto3D	-	celda/CeldaWaku
IMAGEN					

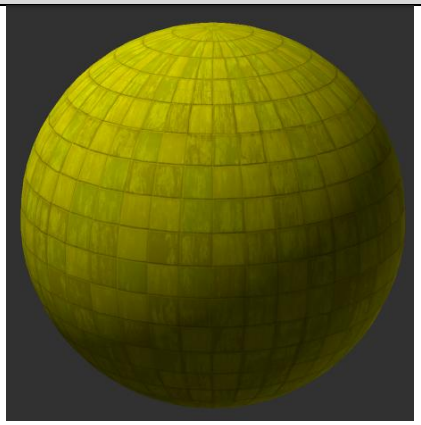


Objeto 3D importado a Unity desde Blender

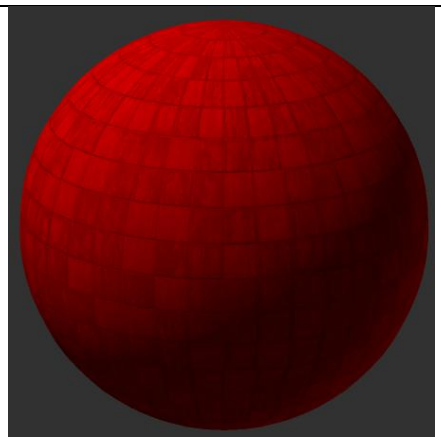
DESCRIPCIÓN
Es la celda transparente en donde se encuentra el personaje

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#6	ControladorMouse	GameObject	Otros_Objetos	ControladorMouse	celda/Celda Waku
IMAGEN					
-					
DESCRIPCIÓN					
Este es un objeto controlador, y lo que hace es controlar los momentos en que el mouse está activado o desactivado, según si el jugador está jugando o no.					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#7	disco	GameObject	Otros_Objetos	-	disco
IMAGEN					
-					
DESCRIPCIÓN					
Este es un objeto contenedor, es decir que tendrá otros objetos incluidos en él					

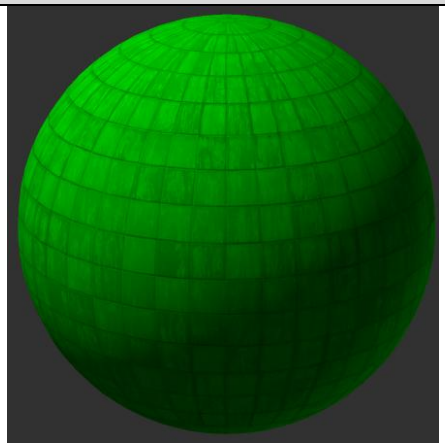
ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#8	Amarillo	Sphere	Objeto3D	ScriptBolaAmarilla	disco/Amarillo
IMAGEN					
					
DESCRIPCIÓN					
Es la esfera amarilla con la cual el jugador interactúa durante el juego					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#9	Rojo	Sphere	Objeto3D	ScriptBolaRoja	disco/Rojo
IMAGEN					



DESCRIPCIÓN
Es la esfera roja con la cual el jugador interactúa durante el juego

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#10	Verde	Sphere	Objeto3D	ScriptBolaVerde	disco/Verde
IMAGEN					



DESCRIPCIÓN
Es la esfera verde con la cual el jugador interactúa durante el juego

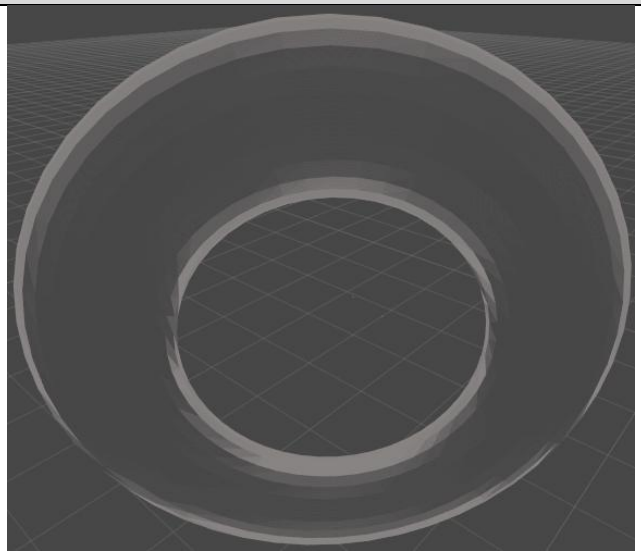
ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#11	ParticulaAmarillo	Particle System	SistemaDeParticulas	-	disco/ParticulaAmarillo

IMAGEN

DESCRIPCIÓN
Es el sistema de partículas que aparece al inicio del nivel, esto hace un efecto para que aparezcan las esferas de color.

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#12	ParticulaRojo	Particle System	SistemaDeParticulas	-	disco/ParticulaRojo
IMAGEN					
					
DESCRIPCIÓN					
Es el sistema de partículas que aparece al inicio del nivel, esto hace un efecto para que aparezcan las esferas de color.					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#13	ParticulaVerde	Particle System	SistemaDeParticulas	-	disco/ParticulaVerde
IMAGEN					
					
DESCRIPCIÓN					
Es el sistema de partículas que aparece al inicio del nivel, esto hace un efecto para que aparezcan las esferas de color.					

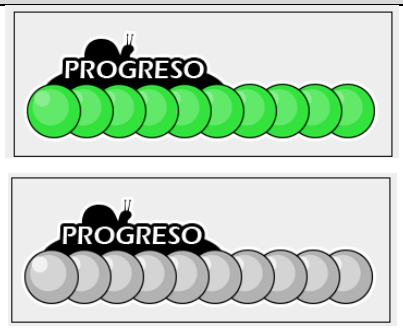
ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#14	Platillo	GameObject (Blender import)	Objeto3D	-	disco/Platillo
IMAGEN					
 Objeto 3D importado a Unity desde Blender					
DESCRIPCIÓN					
Es el platillo o disco que hace de soporte para las esferas de color					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#15	GuiTextures	GameObject	Otros_Objetos	-	GuiTextures
IMAGEN					
-					
DESCRIPCIÓN					
Es un contenedor de objetos GUITexture, como pantallas o carteles, cursores, interfaces y otras imágenes en 2D					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#16	FadeInOut	GameObject	Objeto2D	Fadeinout	GuiTextures/Fade InOut

IMAGEN
El fade-in-out es una imagen en blanco
DESCRIPCIÓN
Este objeto permite hacer un efecto de Fade-in y Fade-out en el juego

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#17	GUI_Atencion	GUITexture	Objeto2D	ScriptGUI_Atencion	GuiTextures/GUI_Atencion
IMAGEN					
					
DESCRIPCIÓN					
Pantalla que aparece cuando la computadora va a generar una secuencia					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#18	GUI_BarraProgreso	GameObject	Objeto2D	ScriptGUI_BarraProgreso	GuiTextures/GUI_Barra Progreso
IMAGEN					
					
DESCRIPCIÓN					
La barra de progreso indica la cantidad de partidas ganadas					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#19	GUI_BienHecho	GUITexture	Objeto2D	ScriptGUI_BienHecho	GuiTextures/GUI_BienHecho
IMAGEN					
					
DESCRIPCIÓN					
La pantalla que aparece cuando el jugador acertó la secuencia					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#20	GUI_Fallaste	GUITexture	Objeto2D	ScriptGUI_Fallaste	GuiTextures/GUI_Fallaste
IMAGEN					
					
DESCRIPCIÓN					
La pantalla que aparece cuando el jugador no acertó la secuencia					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#21	GUI_Flechas	GUITexture	Objeto2D	ScriptGUI_Flechas	GuiTextures/GUI_Flechas
IMAGEN					
DESCRIPCIÓN					
Las flechas que rodean una esfera cuando el mouse pasa sobre ella					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#22	GUI_JuegoGanado	GUITexture	Objeto2D	ScriptGUI_JuegoGanado	GuiTextures/GUI_Juego Ganado
IMAGEN					
-					
DESCRIPCIÓN					
Pantalla que aparece cuando el jugador ha terminado el nivel satisfactoriamente					

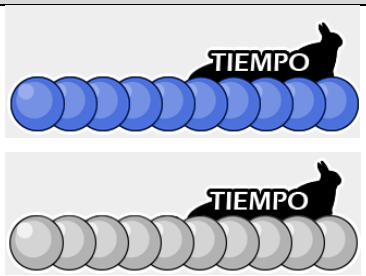
ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#23	GUI_JuegoPausado	GUITexture	Objeto2D	ScriptGUI_JuegoPausado	GuiTextures/GUI_Juego Pausado
IMAGEN					
					
DESCRIPCIÓN					
Pantalla que aparece cuando el juego está en pausa					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#24	GUI_Pausa	GUITexture	Objeto2D	-	GuiTextures/GUI_Pausa
IMAGEN					
					
DESCRIPCIÓN					
Pequeño cartel que dice “pausa” en la esquina superior derecha de la pantalla					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#25	GUI_JuegoTerminado	GUITexture	Objeto2D	Script_JuegoTerminado	GuiTextures/GUI_JuegoTerminado

IMAGEN
-
DESCRIPCIÓN
Aparece esta pantalla cuando el jugador pierde el nivel (game over)

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#26	GUI_Salir	GUITexture	Objeto2D	-	GuiTextures/GUI_Salir
IMAGEN					
					
DESCRIPCIÓN					
Pequeño cartel que dice “salir” en la esquina superior derecha de la pantalla					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#27	GUI_Reloj	GameObject	Objeto2D	ScriptReloj	GuiTextures/GUI_Reloj
IMAGEN					
					
DESCRIPCIÓN					
La barra de tiempo indica cuanto tiempo queda para finalizar la partida					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#28	GUI_SalirDelJuego	GUITexture	Objeto2D	ScriptGUI_SalirDelJuego	GuiTextures/GUI_SalirDel Juego
IMAGEN					

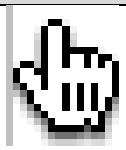
DESCRIPCIÓN
Pantalla que aparece cuando se pulsa la tecla Esc

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#29	GUI_TuTurno	GUITexture	Objeto2D	ScriptGUI_TuTurno	GuiTextures/GUI_TuTurno

IMAGEN


DESCRIPCIÓN
Pantalla que aparece cuando es el turno del jugador de realizar la secuencia

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#30	Mouse_Cursor	GUITexture	Objeto2D	MouseCursor	GuiTextures/Mouse_Cursor

IMAGEN


DESCRIPCIÓN
Cursor del mouse

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#31	palabra_amarillo	GUITexture	Objeto2D	Script_palabra_amarillo	GuiTextures/palabra_amarillo

IMAGEN

DESCRIPCIÓN
Palabra que aparece cuando hay una interacción con la esfera amarilla

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#32	palabra_rojo	GUITexture	Objeto2D	Script_palabra_rojo	GuiTextures/palabra_rojo
IMAGEN					
					
DESCRIPCIÓN					
Palabra que aparece cuando hay una interacción con la esfera roja					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#33	palabra_verde	GUITexture	Objeto2D	Script_palabra_verde	GuiTextures/palabra_verde
IMAGEN					
					
DESCRIPCIÓN					
Palabra que aparece cuando hay una interacción con la esfera verde					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#34	vida1	GUITexture	Objeto2D	ScriptVida1	GuiTextures/vida1
IMAGEN					
					

DESCRIPCIÓN
Vida activa

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#35	vida1Gris	GUITexture	Objeto2D	-	GuiTextures/vida1Gris
IMAGEN					
					
DESCRIPCIÓN					
Vida consumida					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#36	Vida2	GUITexture	Objeto2D	ScriptVida2	GuiTextures/vida2
IMAGEN					
					
DESCRIPCIÓN					
Vida activa					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#37	Vida2Gris	GUITexture	Objeto2D	-	GuiTextures/vida2Gris
IMAGEN					
					
DESCRIPCIÓN					
Vida consumida					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#38	Vida3	GUITexture	Objeto2D	ScriptVida3	GuiTextures/vida3

IMAGEN

DESCRIPCIÓN
Vida activa

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#39	Vida3Gris	GUITexture	Objeto2D	-	GuiTextures/vida3Gris
IMAGEN					
					
DESCRIPCIÓN					
Vida consumida					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#40	Luces	GameObject	Otros_Objetos	-	Luces
IMAGEN					
-					
DESCRIPCIÓN					
Objeto contenedor					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#41	Directional light	Directional light	Luz	-	Luces/Directional light
IMAGEN					
-					
DESCRIPCIÓN					
Luz direccional					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#42	Point light	GameObject	Luz	-	Luces/Point light
IMAGEN					
-					
DESCRIPCIÓN					
Punto de luz					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#43	Point light	GameObject	Luz	-	Luces/Point light
IMAGEN					
-					
DESCRIPCIÓN					
Punto de luz					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#44	Point light	GameObject	Luz	-	Luces/Point light
IMAGEN					
-					
DESCRIPCIÓN					
Punto de luz					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#45	Main Camera	GameObject	Cámara	-ScriptCam -SecuenciaComputadora	Main Camera
IMAGEN					
-					
DESCRIPCIÓN					
Cámara principal del juego					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#46	ParticulasAnim	GameObject	Otros_Objetos	-	ParticulasAnim
IMAGEN					
-					
DESCRIPCIÓN					
Objeto contenedor					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#47	efectoEspiral	GameObject	Otros_Objetos	-	ParticulasAnim/efectoEspiral
IMAGEN					
-					
DESCRIPCIÓN					
Objeto contenedor					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#48	PartEfectoEspiral1	Particle System	SistemaDeParticulas	-	ParticulasAnim/efectoEspiral/PartEfectoEspiral1
IMAGEN					
					
DESCRIPCIÓN					
Sistema de partículas: súper explosión					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#49	Particulas_efectoEspiral1	GameObject	Otros_Objetos	ParticleSpiralEffect	ParticulasAnim/efectoEspiral/Particulas_efectoEspiral1
IMAGEN					
-					
DESCRIPCIÓN					
Objeto que permite crear una súper explosión mediante un script y un sistema de partículas					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#50	PartAnimBolaAmarilla	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartAnimBolaAmarilla
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que hace un efecto sobre la esfera Amarilla					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#51	PartAnimBolaRoja	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartAnimBolaRoja
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que hace un efecto sobre la esfera Roja					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#52	PartAnimBolaVerde	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartAnimBolaVerde
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que hace un efecto sobre la esfera Verde					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#53	PartEstrellas	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartEstrellas
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que crea un efecto de estar navegando por el espacio					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#54	ParticulaCentral	Particle System	SistemaDeParticulas	-	ParticulasAnim/ParticulaCentral
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas multicolor que está en el centro de la pantalla					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#55	PartInteraccion	Particle	SistemaDeParticulas	-	ParticulasAnim/PartInteraccionAm

	Amarillo	System	ulas		arillo
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que se activa al hacer click sobre una esfera					

ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#56	PartInteraccion Rojo	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartInteraccionRojo
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que se activa al hacer click sobre una esfera					

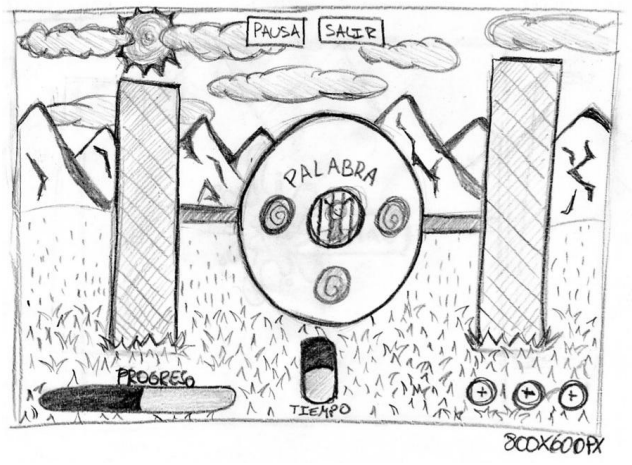
ID	NOMBRE	TIPO	CLASE	SCRIPT ASOCIADO	DIRECCIÓN
#57	PartInteraccion Verde	Particle System	SistemaDeParticulas	-	ParticulasAnim/PartInteraccionVerde
IMAGEN					
-					
DESCRIPCIÓN					
Sistema de partículas que se activa al hacer click sobre una esfera					

APARTADO 3: Bocetos y Prototipos

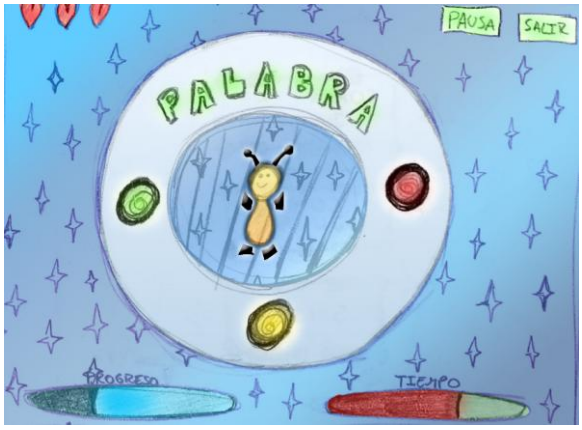
En este apartado se muestra una pequeña parte del proceso creativo de diseño, del cual surgen diferentes modelos gráficos (dibujados a mano alzada) y prototipos desarrollados en el game engine y programas de diseño.

Boceto (nivel fácil)

Boceto (nivel fácil)



Boceto coloreado (nivel fácil)



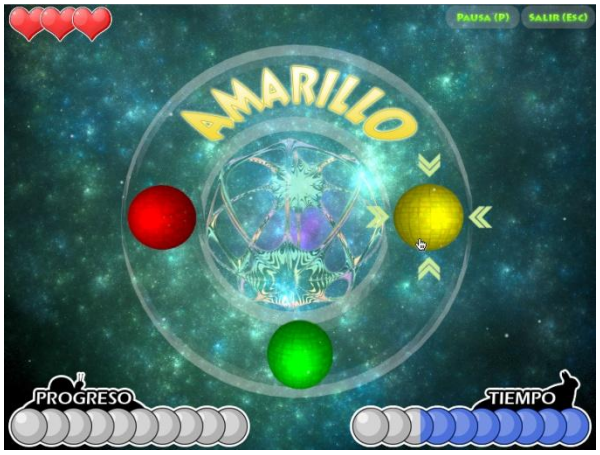
Boceto coloreado (nivel fácil)

Prototipo implementado en Unity

Variación del Prototipo implementado en Unity

Prototipo implementado en Unity

Prototipo. Modelo definitivo

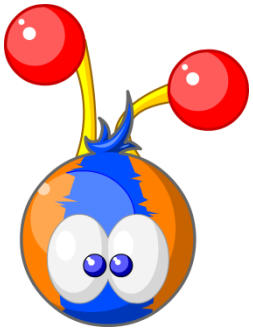
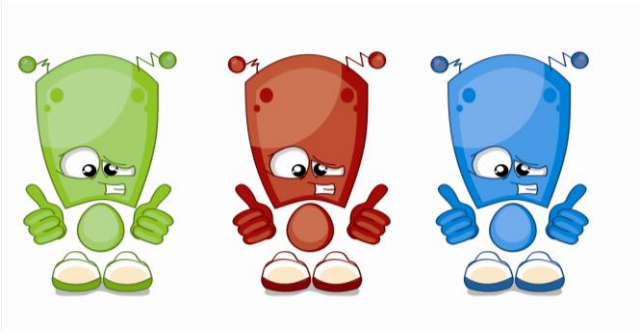


Boceto del menú principal

Prototipo del menú principal. Modelo definitivo

Boceto del logo de Games Ink

Prototipo implementado. Modelo definitivo

Prototipo inicial de personaje	Prototipo alternativo de personaje
	

Boceto de personaje	Prototipo implementado de personaje