



LABORATORIO 1
Vídeo Juegos, Diseño
Historia, Concepto y Diseño de Vídeo Juegos

1. Pre-Laboratorio

- Tome algún video juego que conozca e investigue o analice.
 1. Genero del juego.
 2. Resumen de las mecánicas del juego.
 3. Plataforma en las que esta disponible el juego.

2. ¿Qué es un Vídeo Juego?

Es un juego electrónico que requiere la interacción humana con una interfaz de usuario para generar *feedback* visual en dispositivo de vídeo. Los dispositivos electrónicos utilizados para jugar video juegos son llamados plataformas, ejemplos de ello son los computadores personas y las consolas de video juegos.

El dispositivo principal de entrada en un video juego se llama control de juego o *game controller*, este dispositivo suele variar por genero o plataforma, siendo por ejemplo en los juegos de computador personal mas común el uso del teclado y/o ratón, mientras que en los juegos de consolas de video juegos es mas común el uso de un mando.

La palabra *vídeo* en video juegos se refiere a un dispositivo de imágenes, sin embargo los video juegos como hoy conocemos proveen mas de una fuente de *feedback* tales como audio en su gran mayoría, vibración o *force feedback*, etc.

3. Breve Historia de los Videojuegos - Primera Generación

Cuadro 1: Historia de los Vídeo Juegos

1952	A. S. Douglas crea el primer juego electrónico documentado para su tesis doctoral, un juego de tic-tac-toe corriendo en el computador <i>EDSAC</i> de la Universidad de Cambridge [7].
1958	<i>Tenis for Two</i> de William Higinbotham en un osciloscopio [3].
1962	<i>Spacewar!</i> de Steven Russell en el computador <i>PDP-1</i> [11].
1967	Ralph Baer y sus compañeros de trabajo crean la primera consola de video juegos que funciona sobre televisión estándar, la llaman <i>Brown Box</i> [4].
1971	Nolan Bushnell crea la primera arcade llamada Computer Space [10].
1972	Sale en venta la primera consola de video juegos para el hogar llamada Magnavox Odyssey, se venden aproximadamente 330.000 unidades [9].
1972	Pong publicado por Atari y creado por Nolan Bushnell se convierte en el primer juego arcade exitoso [2].
1974	Sale Gran Trak 10 primer juego arcade de carreras [1].
1974	Sale Maze Wars considerado el primer shooter en primera persona [6].
1977	Atari saca a la venta la consola <i>Video Game Computer System</i> (<i>Atari 2600 o VCS</i>).

4. Desarrollo de Vídeo Juegos

Los video juegos son usualmente creados en grupos de desarrollo conformados por varias personas con distintos roles, una persona puede tener la capacidad de manejar uno o mas roles, algunos roles solo son necesarios durante ciertas etapas del juego (ejemplo testers) mientras que otros son necesarios durante todo el proceso de desarrollo (ejemplo programadores). Los roles necesarios para la creación de un video juego pueden variar según el objetivo y alcance del video juego pero usualmente son los siguientes [8, p. 149]:

4.1. Roles

Diseñador

Diseña las mecánicas de juego, reglas, limitantes, objetivos, estructura y alcance del video juego. Básicamente son los visionarios del juego, en proyectos grandes usualmente este trabajo es dividido en varios roles como diseñadores de mecanicas de juego, diseñadores de interfaces, diseñadores de aventuras (quests), escritores, etc.

Artista

Encargado de producir todo el arte utilizado en el juego, el trabajo del artista puede ser orientado a 3D o 2D. Artistas 2D suelen producir arte conceptual, sprites, texturas, e interfaces de usuario. Artistas 3D suelen producir modelos o mallados, animación, ambientes 3D y cinematográficas.

Programador

Crea la base de código del video juego, esto incluye:

- Físicas: Programación del motor de físicas, simulaciones físicas, colisiones, movimiento de objetos, etc.
- Inteligencia Artificial: Programación de objetos o agentes interactivos utilizando técnicas de inteligencia artificial para juegos como scripting, planificación, decisiones basadas en reglas, etc.
- Gráficos: Programación del contenido gráfico con importantes consideraciones en memoria y performance, la producción del motor gráfico, la integración de modelos, texturas y demás contenido que debe funcionar junto con el motor de físicas y motor de juego.
- Sonido: Integración de musica, dialogo y efectos de sonido en distintos sitios y situaciones.

- Mecánicas de Juego: Implementación de varias reglas, objetivos y respuestas del juego.
- Scripting: Desarrollo y manteniendo de un sistema de comando en alto nivel para la interacción con varios elementos del juego.
- Interfaces de Usuario Desarrollo de elementos de interfaz, menús y sistemas de *feedback*.
- Procesamiento de Input: Establece correlación de distintas acciones, eventos y sistemas de respuestas con variados dispositivos de entrada.
- Networking: Administración de data recibida de forma local o a través de internet.
- Herramientas de juego: Producción de herramientas de desarrollo para el video juego, especialmente para diseñadores y scripting.

Diseñador de niveles

Crea los niveles, misiones y retos del video juego utilizando programas específicos. Los diseñadores de niveles trabajan con versiones completas e incompletas del juego e interactúan directamente con editores de niveles usualmente desarrollados por los programadores del video juego, esto para eliminar la necesidad de los diseñadores tener que interactuar directamente con el código del video juego.

Ingeniero de Sonido

Se encarga de los efectos de sonido y su debido posicionamiento en tiempo y espacio sea 2D o 3D.

Testers

El control de calidad y *QA* o *quality assurance* (aseguramiento de calidad) es llevado por los testers, estos se encargan de analizar un video juego y documentar todo defecto de software encontrado además de analizar si el juego cumple o no con el diseño propuesto.

5. Lenguajes y Herramientas

5.1. HTML5 y Javascript

Phaser

Descripción: *Phaser is a fun, free and fast 2D game framework for making HTML5 games for desktop and mobile web browsers, supporting Canvas and WebGL rendering.*

URL: <https://phaser.io/>

PlayCanvas

Descripción: *PlayCanvas is the world's easiest to use WebGL Game Engine. It's free, it's open source and it's backed by amazing developer tools.*

URL: <https://playcanvas.com/>

Unity3D

Descripción: *Unity is a flexible and powerful development platform for creating multiplatform 3D and 2D games and interactive experiences. It's a complete ecosystem for anyone who aims to build a business on creating high-end content and connecting to their most loyal and enthusiastic players and customers.*

URL: <https://unity3d.com/unity>

COCOS2D-JS

Descripción: *Cocos2d-X is a suite of open-source, cross-platform, game-development tools used by thousands of developers all over the world.*

URL: <http://www.cocos2d-x.org/>

5.2. Java

Slick2D

Descripción: *PSlick2D is an easy to use set of tools and utilites wrapped around LWJGL OpenGL bindings to make 2D Java game development easier.*

URL: <http://slick.ninjacave.com/>

jMonkeyEngine

Descripción: *It's a free, open source game engine, made especially for Java game developers who want to create 3D games using modern technology. The software is programmed entirely in Java, intended for wide accessibility and quick deployment.*

URL: <http://jmonkeyengine.org/>

5.3. C++

COCOS2D-X

Descripción: *Cocos2d-X is a suite of open-source, cross-platform, game-development tools used by thousands of developers all over the world.*

URL: <http://www.cocos2d-x.org/>

Torque2D

Descripción: *Torque 2D is an extremely powerful, flexible, and fast open source engine dedicated to 2D game development.*

URL: <https://www.garagegames.com/products/torque-2d/features>

Godot Engine

Descripción: *Godot is an advanced, feature packed, multi-platform 2D and 3D game engine. It provides a huge set of common tools, so you can just focus on making your game without reinventing the wheel.*

URL: <http://www.godotengine.org/wp/>

5.4. C#

Paradox

Descripción: *Paradox is a versatile and engaging game engine, that will empower you to make stunning games that better fit your vision!*

URL: <http://paradox3d.net/>

Monogame

Descripción: *MonoGame is an Open Source implementation of the Microsoft XNA 4 Framework. Our goal is to allow XNA developers on Xbox 360, Windows & Windows Phone to port their games to the iOS, Android, Mac OS X, Linux and Windows 8 Metro. PlayStation Mobile, Raspberry PI, and PlayStation 4 platforms are currently in progress.*

URL: <http://www.monogame.net/>

Wave Engine

Descripción: *Component Based Game Engine architecture, 2D and 3D physics engines, beautiful visuals effects, cross-platform support, advanced layout system and much more.*

URL: <http://waveengine.net/>

Unity3D

Descripción: *Unity is a flexible and powerful development platform for creating multiplatform 3D and 2D games and interactive experiences. It's a complete ecosystem for anyone who aims to build a business on creating high-end content and connecting to their most loyal and enthusiastic players and customers.*

URL: <https://unity3d.com/unity>

5.5. Python

Pygame

Descripción: *Pygame is a cross-platform library designed to make it easy to write multimedia software, such as games, in Python. Pygame requires the Python language and SDL multimedia library. It can also make use of several other popular libraries.*

URL: <http://www.pygame.org/>

6. Diseño de Vídeo Juegos

El diseño de un juego generalmente inicia con una idea, esta usualmente es una modificación o re-implementación de algún concepto ya existente. Esta idea de juego puede caer entre uno o distintos géneros de juego, es usual la mezcla de géneros en el diseño de un video juego. El diseñador usualmente produce una propuesta de juego documentada en la que se incluye concepto, mecánicas de juego, características del juego, escenario o ambiente del juego e historia y requerimientos para jugar el juego (en cuanto a capacidad cognitiva), este documento también suele incluir los requerimientos para el desarrollo del juego, las personas encargadas y roles asignados para el desarrollo ademas de una de costos [5, p. 101].

7. Actividad

El alumno debe realizar un documento de propuesta de juego, el documento contiene información extensa y concisa sobre el proyecto de juego.

- Primeramente debe conceptualizar un video juego y sus mecánicas.
 1. Escriba un resumen de su "idea" de juego en al menos un párrafo.
- Luego debe realizar el '*Documento de Propuesta de Juego*' donde se extiende y explica en detalle la base conceptual del juego (es común que algunas de las cosas escritas acá cambien durante el desarrollo pero la ideal es mantener el concepto original). El documento debe contener lo siguiente:
 1. **Concepto**, describa la visión del juego en una o dos oraciones.
 2. **Genero**, describa con una sencilla oración donde encaja el juego dentro de un genero definido.
 3. **Gameplay (Mecánicas de Juego)**, en esta sección resuma que hace el jugador mientras esta jugando, también debe indicar como el juego forma parte del genero nombrado y que características lo diferencian de la base conceptual de este genero.
 4. **Features (Características o Distintivos)**, aca debe listar cuales son los principales puntos de atractivo de su juego y que significan estos puntos para el juego.
 5. **Setting (Ambiente o Escenario)**, resuma en pocos párrafos que hace el mundo de su juego y sus ocupantes interesantes.
 6. **Historia**, en caso de tener historia en el juego, haga una pequeña sinopsis.

7. **Target Market (Mercado de Destino)**, explique para quienes esta siendo desarrollada su idea de juego (nichos, géneros de juego específicos, adultos, niños, público general, fans de algún deporte, etc) y las razones por la que escoge a este público como destino.
- **Análisis Competitivo o de Mercado**, investigue juegos similares al menos en género de juego, sus números en venta y popularidad, como su juego saldrá mejor o peor que la competencia y que soluciones tiene para competir ¹.
 - **Escoger un framework, motor o conjunto de bibliotecas para el desarrollo de juego en PC**, debe escoger un framework, motor de juego o conjunto de bibliotecas para el desarrollo de su juego, es de importancia para evitar complicaciones en el desarrollo que lo que escoga incluya o sea fácil de integrar con bibliotecas o frameworks que provean ²:
1. Lógica de juego directamente o a través de scripting.
 2. Editor o manejo de niveles o escenas.
 3. Editor o manejo de objetos de escena.
 4. Manejo de inputs, dispositivos de entrada.
 5. Fácil manejo y organización de assets (arte, texturas, audio, modelos, etc)
 6. Detección de colisiones.
 7. Simulaciones físicas y sistemas de partículas.
 8. Inclusión de animaciones.
 9. Networking

Referencias

- [1] Gran Trak 10. http://www.arcade-museum.com/game_detail.php?game_id=7992.
- [2] ALCORN, A. Atari History, Timeline 1972-1984, November-30-1972. <https://www.atari.com/history/1972-1984-0>.
- [3] ANDERSON, J. Tennis for Two, The story of an early computer game. <http://www.pong-story.com/1958.htm>.
- [4] BAER, R. H. The Brown Box, 1967–68. http://americanhistory.si.edu/collections/search/object/nmah_1301997.
- [5] BETHKE, E. *Game Development and Production*. Wordware Game Developer's Library, 2003.

¹Es usual incluir el resultado y resumen de esta investigación en el 'Documento de Propuesta de Juego'

²Es usual agregar en el 'Documento de Propuesta de Juego' plataforma de destino, herramientas que se van usar y justificación de estas elecciones

- [6] COLLEY, S. Stories from the Maze War 30 Year Retrospective. <http://www.digibarn.com/history/04-VCF7-MazeWar/stories/colley.html>.
- [7] DOUGLAS, A. Noughts and Crosses - The oldest graphical computer game. <http://www.pong-story.com/1952.htm>.
- [8] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.
- [9] LOWOOD, H. Magnavox Odyssey, First home video game console. <http://www.pong-story.com/odyssey.htm>.
- [10] LOWOOD, H. Videogames in Computer Space: The Complex History of Pong. <http://lmc.gatech.edu/~bogost/courses/spring10/lcc8823/lowood.pdf>, 2010.
- [11] MARKOFF, J. Alan Kotok, 64, a Pioneer In Computer Video Games. *The New York Times* (2006).



LABORATORIO 2

Vídeo Juegos, Programación, Diseño
Cámaras Virtuales

1. Pre-Laboratorio

- Investigar:
 1. Vídeo juegos en primera persona.
 2. Vídeo juegos en tercera persona.
 3. Side-scrollers.
 4. Perspectiva.
 5. Proyección Ortogonal.
 6. Géneros de vídeo juegos.
- Investigar segun su framework, biblioteca o motor de juego escogido.
 1. Implementación de cámaras en el juego.
 2. Manejo de escenas.
 3. Scripting o programación de la lógica de juego.

2. Definición

En los vídeo juegos se utilizan cámaras virtuales para mostrar un mundo 3D o 2D en algún dispositivo visual. Un sistema virtual de cámaras se encarga de controlar una o mas cámaras en un escenario de juego. Usualmente la cámara en muchos frameworks y bibliotecas para desarrollo de vídeo juegos no es mas que otro objeto en escena [10].

3. Conceptos

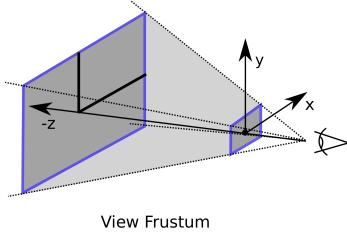


Figura 1: *Viewing Frustum* de una cámara virtual donde se muestra el far y near clipping plane [6].

Algunos conceptos necesarios para entender el uso de cámaras en la gran mayoría de los frameworks, bibliotecas y motores para el desarrollo de video juegos [9].

3.1. Tipos de Proyección

Existen dos tipos proyección en la mayoría de los frameworks y bibliotecas para el desarrollo de video juegos estos son proyección perspectiva y proyección ortogonal.

3.1.1. Perspectiva

Es la forma natural en que el ojo humano percibe una escena, en la proyección perspectiva los objetos distantes se ven mas pequeños que los objetos cercanos dando profundidad a distintos objetos en una escena. Este tipo de proyección es utilizada usualmente en juegos 3D.

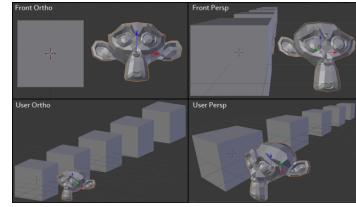


Figura 2: Proyección ortogonal y perspectiva.

3.1.2. Ortogonal

En una proyección perspectiva un objeto lejano es mas pequeño que un objeto cercano, en proyección ortogonal se ignora este efecto, eliminando la profundidad de escena. Este tipo de perspectiva es utilizada en muchos juegos 2D.

3.2. Field Of View (FoV) o Campo de Vista

Es el ancho del angulo de vista de la cámara, indica la extension de lo puede ver la cámara en cualquier momento, usualmente se mide en ángulos, este angulo puede ser el field of view vertical o el field of view horizontal dependiendo del framework, biblioteca o motor de juego usado (ver 3) [1].

3.3. Clipping Planes o Planos de Clipping

En computación se manejan términos discretos por lo tanto una cámara no puede ver hacia el infinito, para esto están el far y near clipping planes los cuales indican donde termina el renderizado de escena y donde empieza según la posición de la cámara respectivamente (ver 1 y 3).

3.3.1. Near Clipping Plane o Plano de Clipping Cercano

Es donde empieza a dibujarse los objetos de escena en display, los objetos antes de este punto son ignorados por el motor gráfico.

3.3.2. Far Clipping Plane o Plano de Clipping Lejano

Es donde termina de dibujarse los objetos de escena en display, los objetos mas lejanos a este punto son ignorados por el motor gráfico.

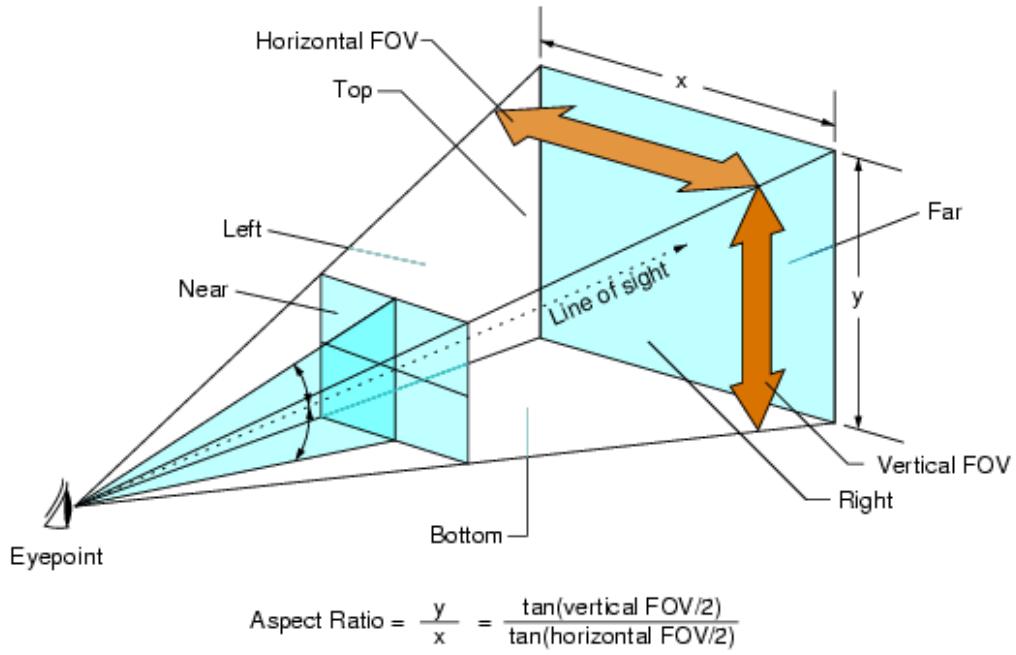


Figura 3: Parámetros de una Cámara Virtual

4. Tipos de Sistemas de Cámara en Vídeo Juegos

Existen principalmente tres tipos de sistemas de cámara en video juegos. En un sistema *fixed* la cámara no cambia sus parámetros originales, un sistema *tracking* sigue algún objeto en juego, y un sistema *interactive* la cámara es parcialmente autónoma y cambia sus parámetros según distintas situaciones. Para implementar distintos sistemas de cámara los desarrolladores de video juegos usan técnicas como programación con restricciones o inteligencia artificial.

4.1. Estáticas (*Fixed*)



Figura 4: Ejemplo de cámaras estáticas en *Resident Evil 2* [5] [2].

En este tipo de cámara las propiedades de la cámara como su posición, orientación y campo visual (*field of view*) son colocadas durante el desarrollo del juego y estas no cambian durante el *gameplay*. Algunos ejemplos de juegos con este tipo de cámara son los primeros Resident Evil, y Alone In The Dark, usualmente es utilizada para crear tensión [3][5].

4.2. Seguidoras (*Tracking*)

Este tipo de cámara sigue a algún objeto en el juego usualmente el personaje principal u otro objeto de considerable importancia. Este sistema presenta varios problemas sobretodo en ambientes tridimensionales y tercera persona donde la cámara podría quedar detrás de una objeto que oculta totalmente la vista o no deja ver algún objeto de interés al jugador [10]. Su uso es muy común en los primeros juegos 3D en tercera persona como *Crash Bandicoot* [7] o *Tomb Raider* [4, p. 39], los juegos primera persona también utilizan una cámara seguidora, a diferencia de en tercera persona donde la cámara esta detrás del personaje en un juego primera persona la cámara esta como visión del personaje principal, en los juegos 2D esta cámara esta presente en todos los juegos tipo side-scroller.

4.3. Interactivas (*Interactive*)

Las cámaras interactivas son cámaras que cambian su posición, campo visual u orientación según distintas situaciones, lugares o intereses del juego y sus desarrolladores, usualmente estas cámaras poseen alguna forma de inteligencia artificial. En su mayoría las cámaras interactivas son cámaras tracking mejoradas, estas siguen al personaje como lo hace una cámara tracking pero su posición y orientación puede cambiar si estas se ven obstruidas por algún objeto o cambian sus parámetros para mostrar objetos de interés claramente evitando de esta forma las desventajas principales de una cámara tracking. Algunos ejemplos de juegos con este tipo de cámara son *Super Mario 64*, *Super Mario Sunshine*, *The Legend of Zelda: The Wind Waker*.



Figura 5: En *Super Mario 64* la cámara rota de forma inteligente para mostrar el camino [8].

5. Actividad

- Durante esta actividad se creara la base del proyecto y se implementara inicialmente la cámara según el diseño de su juego. Los objetivos de la actividad son:
 1. Crear una pequeña escena con *placeholders*¹.
 2. Implementar la cámara principal de juego según el diseño del juego.
 3. Debe crear una estructura de datos (ejemplo una clase) que defina a los actores principales en su mundo de juego, incluyendo el jugador principal. Esto depende directamente del diseño del juego.
 4. Debe asociar esta estructura a un *placeholder* que representa cada actor, incluyendo el jugador principal.
 5. Agrege en la lógica de juego (esto se realiza usualmente a través de scripts u otro medio que provea las herramientas que eligió) comportamientos a uno o

¹Un *placeholder* no es mas que un procurador de un objeto, de manera que el *placeholder* es remplazado en un futuro por el objeto real.

varios actores en escena, este comportamiento debe ser observable por la cámara (ejemplo moverse constantemente)

Referencias

- [1] Feng Zhu School of Design: FoV in Video Games. <http://artsygamer.com/fov-in-games/>, 2014.
- [2] CAPCOM. Resident Evil 2. <http://www.giantbomb.com/resident-evil-2/3030-9418/>, 1998.
- [3] CASAMASSINA, M. Resident Evil Review. <http://uk.ign.com/articles/2002/04/26/resident-evil-3?page=3>, 2002.
- [4] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.
- [5] GIANTBOMB COMMUNITY. Fixed Camera: Concept. <http://www.giantbomb.com/fixed-camera/3015-1715/>, 2014.
- [6] MICROSOFT. What Is a View Frustum? <https://msdn.microsoft.com/en-us/library/ff634570.aspx>, 2014.
- [7] NAUGHTY DOG SOFTWARE. Crash Bandicoot. <http://www.ign.com/games/crash-bandicoot/ps-603>, 1996.
- [8] NINTENDO. Super Mario 64. <http://www.ign.com/games/super-mario-64/n64-606>, 1996.
- [9] UNITY TECHNOLOGIES. Unity Manual: Camera. <http://docs.unity3d.com/Manual/class-Camera.html>, 2014.
- [10] ROLLINGS ANDREW, E. A. *Fundamentals of Game Design*. Prentice Hall, 2006.



LABORATORIO 3

Vídeo Juegos, Programación

Interacción y Feedback con Dispositivos de Entrada

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Feedback.
 2. Háptica o Haptics.
- Investigue al menos 3 dispositivos de entrada utilizados en los video juegos.

2. Introducción



Figura 1: La combinación de teclas WASD es comúnmente utilizadas para controlar el movimiento de un personaje.

Los video juegos son un medio interactivo que requiere del input del usuario, por esto los juegos cuentan con distintos tipos de dispositivos de entrada que proporcionan *feedback* visual al realizar alguna acción con dicho dispositivo. El dispositivo mas conocido es el control de video juego o mando, utilizado comúnmente en consolas, en otras plataformas el dispositivo principal puede variar como en el caso de los juegos de computador personal el uso del mouse y teclado es mas común, los dispositivos de entrada no suelen ser exclusivos de la plataformas por tanto existen mandos para el computador personal e incluso teclados para las consolas [2, p. 395].

3. Feedback

La interacción básica entre un jugador y un juego es sencilla, si el jugador hace el algo entonces el juego hace también algo en forma de respuesta. Esto se le llama *feedback* y es lo que diferencia a un juego de cualquier otra medio de entretenimiento como películas o música [1, p. 18].

Cada entrada al juego debe tener una respuesta discernible. Esta entrada puede tener muchas formas dependiendo de la plataforma y los dispositivos de entrada, la respuesta usual es de tipo visual, auditiva e incluso táctil en caso de estar utilizando un control de juego con vibración.

4. Actividad

Continuando con la programación de la lógica de juego en la actividad del laboratorio dos usted ahora debe seguir programando la base del juego y lógica principal de todos los actores en su mundo de juego.

- Debe cumplir los siguientes objetivos.
 1. Agregar variadas acciones relacionadas con la mecánica de juego al jugador principal, estas acciones deben ahora ser realizadas utilizando teclado y mouse.
 2. Programar lógica de actores en el mundo de juego, expandir la lógica de los actores que interactúan con el jugador para completar su idea en cuanto mecánicas de juego.
 3. Agregar serie de acciones relacionadas con el estado general del juego (pausa, salir, reiniciar), estas deben ser ejecutadas a través del teclado.

Referencias

[1] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.

[2] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.



LABORATORIO 4

Vídeo Juegos, Programación

Sprites

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Texturas (en cuanto a computación gráfica).
 2. Planos o quads.
 3. Rasterización de triángulos.
 4. Matrices de proyección, vista y modelo (projection matrix, view matrix, model matrix)
 5. Espacio de cámara, de mundo, de objeto y de pantalla (camera space, world space, object space, screen space)

2. Definición



En computación gráfica un *sprite* es una imagen 2D o animación que es utilizada en una escena.

Los *sprites* fueron originalmente inventados como un método rápido de composición entre varias imágenes en video juegos 2D utilizando hardware especializado. A medida que el performance de los computadores mejoró esta optimización se convirtió innecesario y el término *sprites* hoy día se refiere específicamente a imágenes 2D que son integradas o utilizadas en una escena [2].

Figura 1: Sprite de una unidad en Battle for Wesnoth [1]

Ahora usualmente *sprite* se refiere a imágenes 2D parcialmente transparentes que son proyectadas a un plano en una escena 3D o 2D.

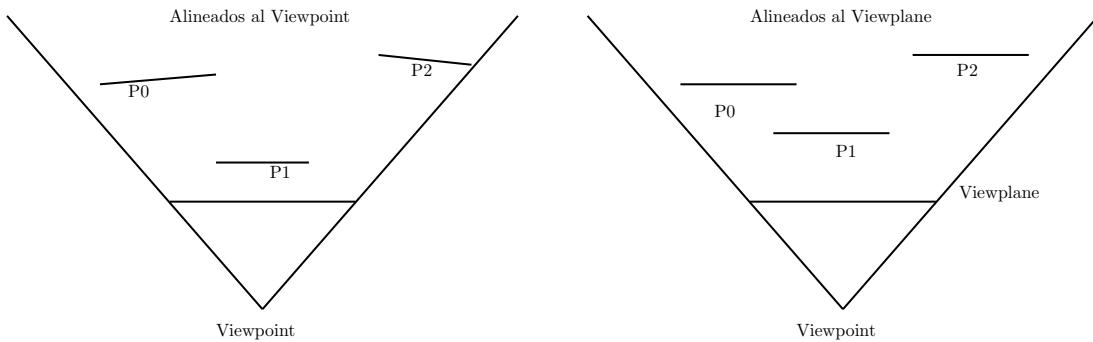


Figura 2: Tecnicas para alinear planos sprites

3. Actividad

Continuando con la programación de las mecánicas del juego entre la actividad del laboratorio 1 y 2.

Al final la actividad el juego debe estar en un estado "jugable" con una sola partida que engloba las mecánicas básicas del juego, sin embargo no presentable para un producto final refinado (faltando pues niveles, interfaz, colisiones, físicas, arte y modelos, música, etc).

- Debe tener por realizada una partida donde se pueda experimentar la idea global de las mecánicas de juego que usted ideo donde muestre:
 1. Objetivos.
 2. Mecánicas de juego.
 3. Reto o dificultad en la forma en la que se logran los objetivos.
 4. Movimiento ya sea del personaje principal, enemigos, u objetos en escena.

Referencias

- [1] The Battle for Wesnoth. <http://www.wesnoth.org/>.
- [2] COLLINS, S. ACM SIGGRAPH: Game Graphics During the 8-bit Computer Era. <http://www.siggraph.org/publications/newsletter/v32n2/contributions/collins.html>, 1998.



LABORATORIO 5

Vídeo Juegos, Programación, Físicas

Detección de Colisiones

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Arquitectura Sistemas-Componente-Entidad (Entity Component Systems o ECS), ver como esta puede ser utilizada para el desarrollo de video juegos en general.
 2. Octrees y quadtrees, arboles BSP y sus usos.
- Explique al menos dos casos para los que se podría utilizar detección de colisiones en un video juegos.
- Investigue para su herramienta de trabajo escogida como incorporar colisiones, si este ya posee detección de colisiones o debe ser incorporado a través de bibliotecas.

2. Definición

Consiste en el problema computacional de detectar la intersection entre dos o mas objetos. Ademas de detectar si dos objetos interceptan un sistema de colisión de objetos puede reportar tiempo y punto de impacto [3].

En una simulación física de colisiones se busca imitar la colisión entre uno o mas objetos de la forma mas precisa posible siguiendo las características físicas de los objetos en cuestión y las distintas propiedades de los materiales que los componen utilizando cuerpos rígidos (*rigid-body physics*) o cuerpos blandos (*soft-body physics*) [4, p. 340].

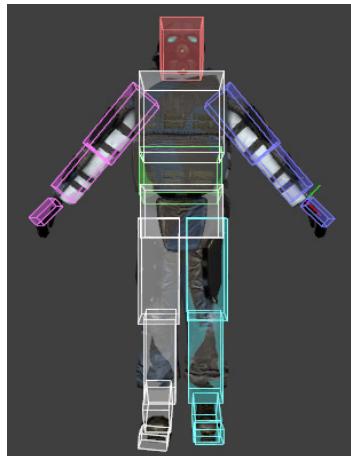
Mientras en un video juego se busca simular colisiones de una forma aceptable, en tiempo real y robusta.

3. Intersecciones Objeto / Objeto

Real Time Rendering: Intersections Object / Object <http://www.realtimerendering.com/intersections.html>. Este sitio presenta referencias a soluciones de intersecciones objeto / objeto, la mayoría de estas soluciones se encuentran en la serie *Graphics Gems*. Muchos frameworks y bibliotecas para el desarrollo de video juegos (o software general) ya tienen estas soluciones implementadas e incluso optimizadas [1].

4. Tecnicas y Optimizaciones Comunes

4.1. Partición Espacial (Spatial Partitioning)



Existen una variedad de algoritmos utilizados para dividir un espacio virtual entre los mas comunes están los octrees, quadtrees y arboles BSP (binary space partitioning). Esta partición espacial se realiza para evitar determinar intersecciones entre objetos que no pertenecen a una misma área determinada por el algoritmo de partición espacial, de esta forma se disminuye el numero de comparaciones aumentando el desempeño o *performance* [5].

4.2. Bounding Boxes o Bounding Volumes

Figura 1: Hitboxes de un *Combine* de *Half Life 2* [6] [7].

Usualmente rectángulos en 2D o cubos en 3D pero también es posible usar otras figuras como esferas, círculos, elipses, paralelogramos, envolventes convexas entre otras. Son usadas como un inicial test de colisión de tal forma que solo se determina la intersección con objetos dentro del bounding volume [2].

4.3. Hit Boxes

Al igual que un Bounding Box estos suelen ser cubos o rectángulos pero también pueden ser otras figuras, el uso de elipses o elipsoides es común. Para objetos animados es usual que los hit boxes estén enlazados a las partes que se mueven de tal forma que los hit boxes también se muevan y rotan en correlación con el personaje u objeto al que pertenecen. Los hit boxes son utilizados para detectar colisiones unidireccionales como un objeto en intersección con un bala u otro proyectil. Son poco precisos para colisiones con *feedback* como chocar contra una pared ya que la posición y orientación de los hit boxes cambia constantemente, este tipo de colisiones son mejor manejadas por bounding boxes alineadas a un eje [6].

5. Actividad

En esta actividad debe incorporar distintas acciones y eventos que hagan uso de detección de colisiones.

- Debe incorporar detección de colisiones para las siguientes condiciones.
 1. Jugador principal contra actores con movimiento.
 2. Jugador principal contra objetos inamovibles.
 3. Jugador principal contra objetos en movimiento.
 4. Actor con movimiento contra objetos inamovibles.
 5. Actor con movimiento contra objetos en movimiento.
 6. Actor con movimiento contra actor con movimiento.
- Debe crear un sistema de historial de eventos donde cada colisión e información de la misma es guardada. Considere como datos necesarios, tiempo de colisión e información de ambos objetos en colisión. Debe agregar mas datos del evento de colisión que vea de utilidad para su mecánica y diseño de juego.

Referencias

- [1] BROWN, E. Object/Object Intersection. <http://www.realtimerendering.com/intersections.html>, 2011.
- [2] CALDWELL, D. R. Unlocking the Mysteries of the Bounding Box. <http://www.stonybrook.edu/libmap/coordinates/seriesa/no2/a2.htm>, 2005.
- [3] ERICSON, C. *Real-time Collision Detection*. Elsevier, 2005.
- [4] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.
- [5] NYSTROM, B. Spatial Partition - Game Programming Patterns / Optimization Patterns. <http://www.stonybrook.edu/libmap/coordinates/seriesa/no2/a2.htm>, 2014.
- [6] VALVE. Developers Community Wiki: Hitbox. <https://developer.valvesoftware.com/wiki/Hitbox>.
- [7] VALVE SOFTWARE. Half Life 2. http://store.steampowered.com/app/220/?snr=1_5_9__300, 2004.



LABORATORIO 6

Vídeo Juegos, Diseño, Programación
Audio, Sonido y Música

1. Pre-Laboratorio

- Investigar lo siguiente:
 1. Derechos de autor y licencias.
 2. OPL u Open Content License.
- De algún juego que conozca, analice:
 1. ¿La música cambia durante algún menú o interfaz? En caso de hacerlo o no, a que cree que se deba esto.
 2. ¿La música durante el gameplay de alguna forma se siente que 'encaja' con el ambiente del juego?
 3. ¿Qué efectos de sonido son comunes en el juego al usar o interaccionar con objetos? (eje. disparar un arma, caminar o chocar con algo) ¿Cuál cree que sea el propósito de estos sonidos?

2. Introducción

Los primeros video juegos no poseían audio hoy día los video juegos son un medio audio-visual. El audio en los video juegos puede ser separado en tres categorías: efectos de sonido, música y voces [1][3].

Los efectos de sonido son producidos modificando *samples* de audio o replicando y grabando dichos efectos de sonido con objetos reales. Los efectos de sonido son importantes para proporcionar inmersión.

La música puede ser electrónica o sintetizada, o producida con instrumentos en vivo. Existen varios contextos en donde la música es presentada en el juego por ejemplo la música ambiental que busca reforzar el estado de animo o apariencia del juego, música específica para menús y créditos, música para batallas, persecuciones o cualquier otro evento rápido y tenso [2, p. 188].

Las voces son utilizadas para reforzar las interacciones entre personajes y sus personalidades.

3. Actividad

Durante esta actividad se empieza a usar uso del sistema de historial de eventos realizado en la actividad del laboratorio 5 y el manejo de eventos / triggers según su herramienta de trabajo escogida.

- Debe incorporar una respuesta auditiva a todos los eventos realizados en la actividad del laboratorio 5. El audio puede o no estar relacionado con la acción realizada. En próximos laboratorios se incorporara audio en general para propósitos artísticos y de inmersión por lo tanto si debe existir relación.

Referencias

- [1] BETHKE, E. *Game Development and Production*. Wordware Game Developer's Library, 2003.
- [2] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.
- [3] VALVE. Developers Community Wiki: Sound and Music. <https://developer.valvesoftware.com/wiki/Audio>.



LABORATORIO 7

Vídeo Juegos, Programación, Diseño
Event Logging y Manejo de Usuarios

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Software tracing, event logging y sus diferencias.
 2. Software Multi-usuario
- De algún juego que conozca analice.
 1. ¿En alguna forma este juego guarda información de su progreso?
 2. Si la pregunta 1. es positiva, ¿de qué forma es utilizada esta información?
- Investigue o nombre algún juego que conozca que permita el manejo de varios usuarios.
¿De qué forma el juego maneja múltiples usuarios?

2. Event Logging

Los video juegos actuales son piezas de software de gran complejidad por lo tanto mantener un historial de información durante la ejecución del programa puede ser de inmensa ayuda para distintos propósitos. Esta información es usualmente utilizada para debugging pero dependiendo del tipo de información guardada esta puede ser usada para una gran cantidad de propósitos [2].

3. Manejo de Usuarios

Los video juegos usualmente puede ser jugados por mas de un usuario incluso si este video juego es de un solo jugador, con cada usuario teniendo su propia sesión de juego, esta sesión de juego puede guardar cosas tan sencillas como el nombre de usuario para propósitos de personalización o datos mas complejos como actual posición en el transcurso del juego, puntuación, logros, objetivos actuales, etcétera, todo esto dependiente del diseño del juego.



Figura 1: En los arcade juegan multiples usuarios [1].

4. Actividad

En esta actividad debe expandir sobre el básico historial de eventos en actividades anteriores, creando finalmente un sistema de event logging completo para su juego con el que pueda de manera sencilla extraer de este información necesaria para guardar el progreso del jugador y permitir la existencia de varios usuarios.

- Debe usted agregar toda clase de información de eventos y/o acciones realizadas en su juego en sistema de event logging que vea necesario para guardar información del progreso del jugador y su desempeño en el juego, considere también según su diseño de juego eventos y/o acciones que crea necesario guardar información de estas.
- Utilizando el progreso guardado del jugador se debe poder guardar y cargar una partida con el teclado o alguna interfaz básica si lo desea (hasta ahora no ha sido requerimiento incorporar interfaces al juego)

Referencias

- [1] Arcade Database. <http://gamesdbase.com/system-arcade.aspx>.
- [2] NAISH, C. Software Tracing, The Catch 22. <http://www.debuggingexperts.com/software-tracing-catch-22>, 2010.



LABORATORIO 8

Vídeo Juegos, Programación, Diseño

Menús e Interfaz de Juego

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Widget o Control con respecto a interfaces gráficas de usuario.
 2. Retained Mode e Immediate Mode con respecto a APIs de interfaces de usuario.
- De algún juego que conozca analice.
 1. Menú o Pantalla de inicio, observe el diseño de la misma.
 2. Interfaz de usuario durante el juego o HUD, observe el diseño de la misma.
 3. ¿Es el diseño de las interfaces concorde con el ambiente del juego?
 4. ¿Considera las interfaces de este juego sencillas, sustanciales en información y agradables a la vista?
- Según su framework, biblioteca o motor escogido debe investigar como integrar interfaces de juego. Se recomienda practicar ejemplos.

2. Introducción

Crear elegantes, atractivas pero a las vez funcionales interfaces es uno de los trabajos mas vitales durante el desarrollo de un video juego. Acá se decide como el juego debe verse en pantalla, como la información del juego se presenta al jugador y como el jugador usa el control de juego o el mouse y teclado para realizar distintas tareas [6, p. 26].

Los video juegos durante su ejecución presentan variados sistemas de interfaces, menús y pantallas de información al usuario, en la mayoría de los casos los juegos presentan como mínimo una pantalla de inicio con la opción de empezar el juego y salir de él mismo, otro menú muy común es el menú de opciones en el cual se presentan una variedad de personalizaciones sobre ciertos parámetros del juego [3]. Es usual que en plataformas como el computador personal se presenten mayor número de opciones en el menú de opciones tales como opciones gráficas y de performance debido a la variada cantidad de configuraciones de hardware, en otras plataformas como las consolas es común que solo se presenten modificaciones sobre el audio del juego, dificultad o re-configuración de los controles de juego.

3. Menús e Interfaces Comunes



Figura 1: Menú de Inicio del juego *DOOM II* presenta inicio de juego, opciones, guardar y cargar partidas y salida del juego [4].

Las necesidades de interfaces, pantallas y menús cambian según los requerimientos de diseño de cada juego, sin embargo se presentan acá los componentes de interfaz y menús encontradas en la mayoría de los video juegos actuales.

3.1. Menú de Inicio o Menú Principal (Start Menu o Main Menu)

Es la primera interfaz que encuentra el usuario al iniciar el juego, presenta por lo menos la opción de iniciar el juego y salir del mismo [6, p. 28], también se suelen incluir opciones como ir al *Menú de Opciones*, ir al *Menú de Partidas*, ver créditos entre otras [1].

3.2. Menú de Opciones (Options Menu)

Es un menú donde se presenta distintas opciones para modificar el juego, usualmente como mínimo se presentan controles de volumen/audio y dificultad, el menú de opciones

suele dividirse en distintas ramas según el tipo de configuración de tal forma se puede tener un menú de opciones para el audio, otro menú de opciones para los controles, otro menú de opciones para la configuración gráfica, menú de opciones para *gameplay* y así muchos mas dependiendo del diseño, alcance y plataforma del video juego. Suele presentarse a través del *Menú de Inicio* pero en algunos juego también puede accederse directamente desde el *gameplay* a través de algún menú de pausa o salida [3].

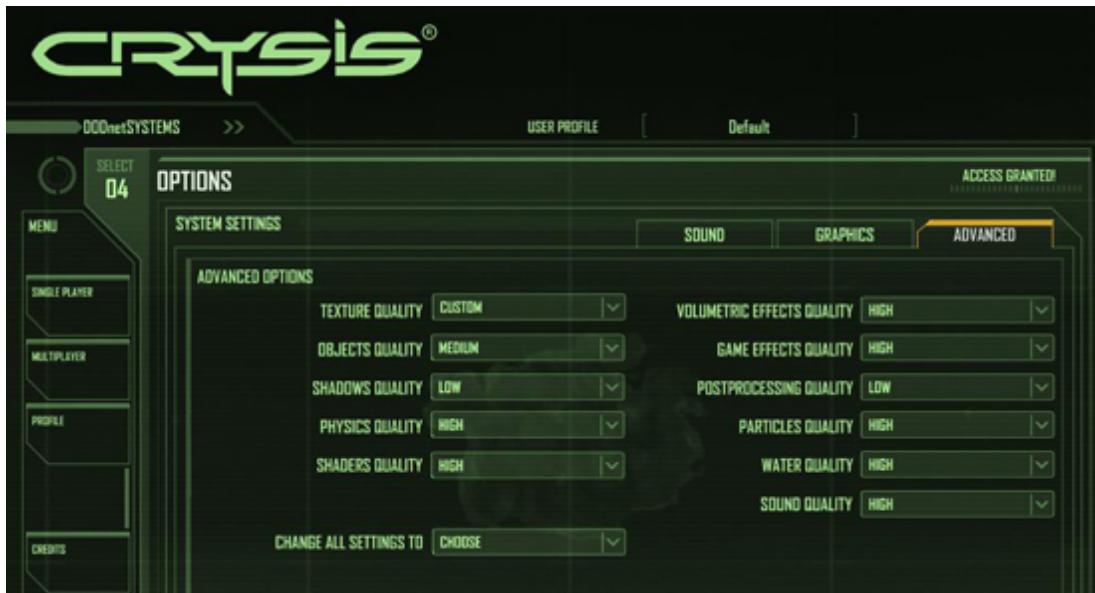


Figura 2: Menú de Opciones del juego *Crysis* presenta configuración de audio y gráficas [2].

3.3. Menú de Partidas (*Save Games*)

Un menú donde se presentan las distintas sesiones de juego guardadas usualmente se presenta información como tiempo de juego, lugar de juego entre otros. Suele presentarse a través del *Menú de Inicio* pero en algunos juego también puede accederse directamente desde el *gameplay* a través de algún menú de pausa o salida.

3.4. HUD (*Heads-Up Display*)

Interfaz que muestra información del juego en todo momento durante el juego, generalmente se presenta la información de forma concisa y corta utilizando iconos, barras

y números. El HUD muestra distinta información, diseño y integración el juego según el tipo de juego y su diseño, usualmente se muestran cosas como numero de vida o cantidad, puntuación, mini-mapa o radar, munición, etc todo esto dependiendo del video juego [7].



Figura 3: HUD del juego *Quake Live* vida, munición y defensa en el panel inferior, armas disponibles en el panel izquierdo y mas [5].

4. Actividad

Durante esta actividad debe agregar distintas piezas de interfaz a su juego, puede utilizar *placeholders* para el diseño visual de su interfaz.

- Debe agregar un menú principal, el menú principal debe tener las siguientes acciones / opciones como mínimo:
 1. Menú Principal → Empezar Juego.
 2. Menú Principal → Cargar Partida.
 3. Menú Principal → Salir del Juego.

- Debe agregar un menú de pausa y/o un menú de opciones / configuración, debe tener como mínimo las siguientes opciones:
 1. Ir a Menú Principal
 2. Salir del Juego
 3. En el menú de opciones al menos debe existir la opción de controlar volumen.
Durante el proceso de desarrollo en laboratorios futuros se deberán agregar mas opciones según el diseño de su juego como dificultad, controles y opciones gráficas.
 4. (Opcional) Cargar partidas desde el menú de pausa u opciones / configuración.
- Debe agregar un HUD o donde muestre información de el estado del jugador e información del juego, que mostrar en el HUD es dependiente de su diseño y mecánicas de juego.
- En laboratorios anteriores se trabajo una idea básica de partidas de jugador, ahora deberá agregar un Menú de Partidas, acá debe mostrar información básica de cada partida. Ademas deben existir las siguientes opciones:
 1. Cargar Partida
 2. Borrar Partida

Referencias

- [1] Video Game Design/Structure: The Main Menu. https://en.wikibooks.org/wiki/Video_Game_Design/Structure#The_Main_Menu.
- [2] CRYTEK. Crysis. <http://crysis.com/>, 2007.
- [3] GIANTBOMB COMMUNITY. Options Menu: Concept. <http://www.giantbomb.com/options-menu/3015-6644/>, 2014.
- [4] ID SOFTWARE . DOOM II: Hell on Earth. http://doomwiki.org/wiki/Doom_II, 1994.
- [5] ID SOFTWARE. Quake Live. <http://www.quakelive.com/#!home>, 2010.
- [6] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.

- [7] WILSON, G. Off With Their HUDs!: Rethinking the Heads-Up Display in Console Game Design. http://www.gamasutra.com/view/feature/2538/off_with_their_huds_rethinking_.php, 2006.



LABORATORIO 9

Vídeo Juegos, Programación, Diseño
Niveles y Diseño de Niveles

1. Pre-Laboratorio

- Investigar los siguientes conceptos:
 1. Ludificación o Gamification.
 2. Mecánicas y Dinámicas de juego.
- De algún juego que conozca analice.
 1. ¿Como se progresá en este juego?
 2. ¿Existen niveles en este juego? ¿Como se realiza la transición entre uno y otro?
 3. ¿El progreso del juego es de forma lineal o ciertas decisiones puede modificar el mundo del juego?
- Investigar según su herramienta de trabajo como incorporar niveles, si posee editor de niveles o debe integrar mas herramientas a su ambiente de trabajo. Recomendado probar ejemplos.

2. Introducción

Un nivel, mapa, área, mundo, zona o fase en un video juego es el espacio total disponible al jugador mientras este mismo completa algún objetivo concreto de el juego [1, p. 107].

En juegos con una progresión lineal los niveles son áreas que forman parte de un mundo mas grande, los juegos suelen conectar niveles con cada nivel representando una

zona o localidad en un mundo de mayor tamaño, esto suele suceder por dos razones, primero diseño del juego y segundo por razones de *performance* [2, p. 104].

Otro juegos siguen un enfoque de niveles basados en la posición del jugador en un mapa. Usualmente usado en juegos *open world* (mundo abierto), en esta clase de juegos usualmente se propone un mundo totalmente abierto a la exploración sin embargo es muy común que exista cierta progresión lineal en la historia u objetivo del juego que lleva al jugador a explorar ciertas regiones de este mundo de juego, un juego *open world* donde no existe esta idea de progresión u objetivo alguno se le llama un *sandbox* (caja de arena) puro acá el jugador simplemente está en un mundo de juego con el que puede interaccionar sin ningún objetivo claro proveído por el juego [2, p. 104].

Cada nivel usualmente tiene asociado algún objetivo diseñado para el mismo, este objetivo puede ser tan sencillo como moverse de un punto a otro, es usual que cuando el objetivo sea completado el jugador sea trasladado a otro nivel o el siguiente nivel en caso de un juego con progresión lineal, es usual también que si el jugador falla completando el objetivo este deba volver a completar el nivel o retornar a algún punto en el mismo [1, p. 111].

3. Actividad

Durante esta actividad debe incorporar más niveles a su juego aumentando su durabilidad y retos. También si su juego posee alguna historia este es el momento perfecto de incorporarla.

- Para esta actividad debe cumplir como mínimo con los siguientes objetivos.
 1. Debe crear continuidad en el juego, ya sea a través de progreso por distintos niveles en modo lineal u con objetivos basados por posición.
 2. Debe almacenar información del desempeño del jugador en un nivel o avance a través de su línea de progresión. Esta información es dependiente de su diseño de juego pero debe ser suficiente para tener alguna forma de contabilizar que tan bien juega el usuario.

Referencias

- [1] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.
- [2] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.



LABORATORIO 10

Vídeo Juegos, Programación, Diseño
Puntuaciones, Objetivos y Motivación

1. Pre-Laboratorio

- De algún juego que conozca analice.
 1. Objetivos o Misiones en el juego.
 2. ¿De qué forma el juego le motiva a continuar?
 3. ¿Existe alguna medida del progreso, desempeño o habilidad del jugador?
 4. ¿Puede resumir ciertas mecánicas del juego en ciclo reto/recompensa?

2. Introducción

La importancia de una experiencia de juego depende de el interés general que este juego pueda generar. Crear y mantener el interés de el jugador es una forma de manejar su grado de motivación. La motivación del jugador es el factor que va determinar si el jugador va continuar jugando después de cierto tiempo e incluso después de terminar el juego [5, p. 75].

En un inicio se tiene la ventaja de saber que el jugador esta motivado al comenzar el juego porque esta ya ha tomado los primeros pasos de obtener e iniciar el juego. Luego de esto es que empieza nuestro trabajo como desarrollados y diseñadores de mantener al jugador motivado y entretenido. Es usual que los primeros minutos de juego suelan definir si se capta o no al jugador [4].

3. Sistemas de Motivación

El diseño del juego debe construir un ciclo sobre las necesidades del jugador y responder a estas con una sucesión de retos y recompensas. Esta estructura en video juego esta construida alrededor de los principios de progresión, avance y logros [4].

Crear un ciclo reto/recompensa es relativamente sencillo pero puede fácilmente volverse repetitivo y restrictivo. Es tarea de los desarrolladores y diseñadores mantener al jugador motivado en el ciclo reto/recompensa, evitando los problemas principales que pueden convertir a este ciclo en un juego tedioso.

3.1. Motivación en las Recompensas

Un sistema de recompensas es base fundamental en para mantener la motivación. Básicamente consiste en que cada esfuerzo del jugador debe tener alguna forma de recompensa, esta recompensa puede tomar muchas formas y su rol principal es por supuesto que motivar al jugador.

En juegos de action-RPG como *Diablo* [2] o *Guild Wars* [1] el gameplay gira alrededor del poder que tiene el personaje manejado por el jugador, este poder va creciendo a medida que el jugador progresa en el juego, la recompensa es un personaje mucho mas poderoso que el personaje a medida que se progresa en el juego. Este progreso también abre acceso a nuevas áreas y objetos del juego.

3.2. Motivación en las Necesidades

Comúnmente utilizado en juegos de estrategia, construcción o *managing*. En juegos como *StarCraft* [3] la mecánica completa del juego consiste en la adquisición y control de los recursos en un mapa, para progresar en el juego el jugador actualmente necesita los recursos proveídos por el juego.

3.3. Motivación en el Reto

Usual en juegos competitivos como juegos de pelea o juegos de deportes, en este caso la motivación del jugador existe gracias al reto que el juego proporciona, la necesidad del jugador es cada vez ser mejor y saber mas del juego, de manera que este se sienta preparado para nuevos retos, la recompensa es la victoria ya que esta prueba el esfuerzo hecho por el jugador.

3.4. Motivacion por el Estado del Jugador

Usual en juegos como shoot em up y beat'em up estos juegos están principalmente basados en el desempeño y estado del jugador. La características del jugador son mejorables pero no de forma permanente. Todo lo que puede adquirir el jugador en estos juegos es temporal y el perder o morir en estos juegos es de gran costo, usualmente teniendo que empezar de nuevo o sin ninguna clase de mejora. El objetivo del jugador en estos juegos es mantener su capacidad lo mejor posible durante el juego tomando las mejoras disponibles a su momento para hacer el progreso lo mas sencillo posible.

4. Sistemas de Puntuación

Un sistema de puntuación es una buena forma de proveer motivación al jugador. Es parte integral en un sistema de recompensas, esto permite premiar al jugador y ademas confirmar su éxito en alguna acción o forma de reto en el juego.

El jugador es premiado con puntos y/u otorgándole un rango en el juego. Esta puntuación determina el progreso del jugador, es incluso posible agregar "bonos.^a los premios por objetivos extra o premiando la eficiencia del jugador completando alguna tarea. El jugador crea entonces una relación lógica donde el universo de juego esta organizado y estructurado en un sistema de valores, donde a mayor esfuerzo, mayor recompensa, donde el progreso en el juego facilita el esfuerzo y da acceso a mejores recompensas.

5. Actividad

Continuando principalmente con la base del historial de eventos y la medida de desempeño de laboratorios anteriores durante esta actividad se debe implementar sistemas de motivación al jugador de alguna forma premiando a este por su progreso y habilidad en el juego.

- Durante esta actividad debe cumplir con lo siguiente:
 1. De forma natural la idea de motivación ya debería estar en sus mecánicas de juego, clasifique que clase de motivación existe en su diseño de juego y refine esta idea en su juego.
 2. Cree variadas situaciones donde el jugador puede completar retos y es recompensado por esto.
 3. Observe las necesidades del jugador durante su progresión en el juego y considere que clase de objetos, actores o situaciones puede crear para aumentar el sentido de reto, motivación o simplemente mejorar la forma de progresión del jugador principal . Agregue estos objetos, actores o situaciones con su respectiva lógica al juego.
 4. Contabilice el desempeño del jugador en variadas situaciones, debe mostrar un leaderboard con las puntuaciones de cada usuario o cada nivel o partida.

Referencias

- [1] ARENA.NET. Guild Wars. <http://www.guildwars.com/en/>, 2005.
- [2] BLIZZARD ENTERTAINMENT. Diablo. <http://www.ign.com/games/diablo/pc-1890>, 1997.
- [3] BLIZZARD ENTERTAINMENT. StarCraft. <http://www.ign.com/games/starcraft/pc-2159>, 1998.
- [4] GHOZLAND, D. Design for Motivation. http://www.gamasutra.com/view/feature/1419/designing_for_motivation.php?print=1, 2007.

[5] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.



LABORATORIO 11

Vídeo Juegos, Diseño, Programación
Arte, Assets, Texturas y Modelos

1. Pre-Laboratorio

- Investigar:
 1. Game Asset Management y Control de Versiones.
 2. Licencias y derechos de autor.
- De algún juego que conozca analice.
 1. Arte del juego, investigue arte conceptual de este juego.
 2. ¿Es el arte del juego congruente con los conceptos de arte?
 3. ¿Existe cierta relación entre el ambiente y arte del juego y su genero? ¿De que forma?
- Traer material artístico para incorporar en el juego, puede ser extraído de cualquier medio que tenga disponible (propio, amigos, uso publico, etc). Esto incluye, arte visual y auditiva.

2. Introducción

Buen arte se ha convertido en una forma de juzgar los video juegos. Un inmensa mayoría compra y gana interés en algún video juego a partir de como estos se ven, esto es una reacción lógica ya que desde una tienda virtual o física o a través de videos o tomas de pantallas no se puede evaluar el gameplay del juego en cuestión [1, p. 171].

Los artistas afectan ahora una inmensa parte del diseño del juego, desde el diseño de la interfaz de juego hasta hasta el ambiente general presentado en el universo del juego y los efectos especiales. La creación de arte ha aumentando en complejidad a través de los años y el crecimiento de los video juegos como una industria, de tal forma las herramientas de creación de arte también cada vez son mas complejas.



Figura 1: *Okami* [3] es un juego con arte inspirado en sumi-e [2].

3. Actividad

- Durante esta actividad se debe agregar toda clase de assets de arte, texturas, modelos, sonido, etc, se debe remplazar todos los placeholder por sus verdaderas representaciones.
- Debe describir cual es el ambiente y estilo que posee el juego, su historia si la tiene, y justificar el uso de sus assets.
- Enlazar eventos y acciones a sonidos y/o efectos visuales que vea necesario.

Referencias

- [1] GATES, B. *Game Design (2nd Edition)*. Thomson Course Technology, 2004.
- [2] JARANSON, C. What Is Sumi-e? - Traditional East Asian Brush Painting. <http://www.sumiesociety.org/whatissumie.php>.
- [3] KAMIYA, H. Okami. <http://www.ign.com/games/okami/ps2-678618>, 2006.



LABORATORIO 12

Vídeo Juegos, Diseño, Programación
Animaciones

1. Pre-Laboratorio

- Investigar:
 1. Animación por Keyframes.
 2. Animación por Motion Capture.
 3. Animación Stop-motion
- Investigar según su herramienta de trabajo como agregar animaciones 2D, 3D. Recomendado probar ejemplos antes del laboratorio.
- Traer animaciones para integrar en su juego, estas pueden venir desde cualquier fuente que le este disponible.

2. Introducción

Animación es el proceso de crear la apariencia de movimiento y cambios de forma mostrando rápidamente una secuencia de imágenes estáticas que difieren muy poco entre unas y otras. Los animadores son artistas que se especializan en la creación de animación.

La mayoría de los juegos modernos tienen algún objeto o personaje que necesita ser animado. La excepción en esto esta en juegos de carreras, simuladores o puzzles (e incluso en estos algunos detalles podrían estar animados) [1, p. 381].

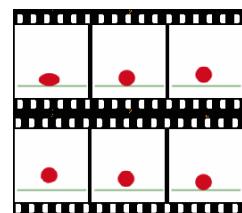


Figura 1: Cuadros de animación de una pelota rebotando.

3. Actividad

Continuando con el laboratorio anterior durante esta actividad seguimos agregando assets al juego.

- Durante esta actividad se debe agregar toda clase de assets de arte, texturas, modelos, sonido, etc, se debe remplazar todos los *placeholder* por sus verdaderas representaciones.
- Debe incluir distintas animaciones a variadas acciones y/o eventos realizados tanto por el jugador principal como por los actores en el mundo de juego.
- Debe incluir distintas animaciones a variados objetos que considere animados existentes en el mundo de juego.
- Enlazar eventos y acciones a sonidos y/o efectos visuales que vea necesario.

Referencias

- [1] BETHKE, E. *Game Development and Production*. Wordware Game Developer's Library, 2003.



LABORATORIO 13

Vídeo Juegos, Físicas, Programación
Física en Vídeo Juegos, Simulaciones y Ragdolls

1. Pre-Laboratorio

- Investigar:
 1. Las tres leyes del movimiento de Newton.
 2. Cálculos en punto flotante en simulaciones físicas y errores.
 3. Determinismo y no-determinismo en simulaciones físicas.
- De algún juego que conozca analice.
 1. ¿En algún momento son utilizadas simulaciones físicas?
 2. Si el juego posee simulaciones físicas ¿cuál cree que sea el propósito de estas?
 3. En caso de no tener simulaciones físicas ¿considera algún caso donde el juego se pueda beneficiar de agregar simulaciones físicas?
- Investigar según su herramienta de trabajo como incorporar simulaciones físicas, ragdolls, partículas y proyectiles tanto en 3D como en 2D. Recomendado practicar con ejemplos especialmente en 2D.

2. Introducción

Para agregar realismo, nuevas mecánicas o mayor calidad visual se introducen leyes físicas dentro del motor de juego, es mayormente usado en juegos tridimensionales [3, p. 325]. Estas nuevos efectos se introducen en forma de simulaciones las cuales son aproximaciones de fenómenos reales utilizando valores discretos [2]. En el ambiente de un video juego una

simulación completa y totalmente correcta podría causar complicaciones en las mecánicas de juego, progreso de la historia o hacer tediosas ciertas actividades.

3. Simulaciones Físicas



Figura 1: BeamNG un video juego simulador de vehiculos que utiliza *soft-body physics*.

Hay dos clases centrales de simulaciones físicas, simulaciones de cuerpos rígidos (*rigid-body physics*) y simulaciones de cuerpos blandos (*soft-body physics*). En una simulación de cuerpos rígidos los objetos se agrupan entre categorías basadas en como deberían interaccionar, las simulaciones de cuerpos rígidos son menos intensas en cuanto a perdida de *performance*. Las simulaciones de cuerpos blandos consisten en simular secciones individuales de cada objeto de tal forma que este se comporte de manera realista, usualmente utilizadas para simular objetos deformables como ropa o materiales destructibles [2].

4. Físicas *Ragdoll*

Es una técnica de simulación que consiste en la animación procedimental de un personaje cuando este muere (u otro estado definido por el juego para causar *ragdoll*, consiste en tratar a un objeto o personaje como una serie de objetos sólidos (huesos) conectados en distintos puntos formando un esqueleto. La simulación ocurre cuando el evento necesario para causar físicas *ragdoll* sobre un objeto o personaje sucede, en los video juegos esto pasa usualmente cuando el personaje muere [1].



Figura 2: Simulación de un personaje cayendo unas escalera, se utiliza un *ragdoll*.

5. Actividad

Durante esta actividad se busca agregar físicas al juego para mejorar la inmersión y calidad visual del juego. Este laboratorio se enfocara en el uso de simulaciones físicas con el uso de rigid-bodies (cuerpos rígidos)

- Se debe hacer uso de rigid-bodies para varios objetos en escena.
- Provocar un eventos al colisionar con un objeto rigid-body.
- Asociar alguna acción al evento de colisión con un rigid-body, como reproducir un sonido o disminuir alguna propiedad asociada al jugador principal (como los puntos de vida, etc).

Referencias

- [1] BROWN, E. Ragdoll Physics On The DS. http://www.gamasutra.com/view/feature/132309/ragdoll_physics_on_the_ds.php, 2009.
- [2] MILLINGTON, I. *Game Physics Engine Development*. Morgan Kaufmann, 2007.
- [3] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.



LABORATORIO 14

Vídeo Juegos, Físicas, Programación
Física en Vídeo Juegos, Partículas y Proyectiles

1. Pre-Laboratorio

- Investigar:
 1. Movimiento parabólico y rectilíneo y rectilíneo uniforme.
 2. Definición de partícula.
 3. Simulación N-Body y Simulación Barnes-Hut.
- De algún juego que conozca analice.
 1. ¿En este juego existe el uso de proyectiles o partículas?
 2. Si el juego posee ¿cuál cree que sea el propósito de estas partículas y/o proyectiles?
 3. En caso de no tener ¿considera algún caso donde el juego se pueda beneficiar de agregar partículas o proyectiles?

2. Introducción

Para agregar realismo, nuevas mecánicas o mayor calidad visual se introducen leyes físicas dentro del motor de juego, es mayormente usado en juegos tridimensionales [3, p. 325]. Estas nuevos efectos se introducen en forma de simulaciones las cuales son aproximaciones de fenómenos reales utilizando valores discretos [2]. En el ambiente de un video juegos una simulación completa y totalmente correcta podría causar complicaciones en las mecánicas de juego, progreso de la historia o hacer tediosas ciertas actividades.

3. Sistemas de Partículas

Es una técnica utilizada en físicas de juegos y computación gráfica en la que se usa una cantidad grande de pequeños *sprites* u otros objetos visuales para simular ciertos fenómenos como sistemas altamente caóticos, fenómenos naturales o procesos causados por reacciones químicas [4].

Algunos ejemplos de fenómenos que son replicados utilizando sistemas de partículas, es el fuego, explosiones, humo, agua en movimiento (como cascadas de agua), nubes, estrellas, galaxias, etc. Es también común su uso para efectos visuales abstractos.



Figura 1: *Powder Toy* un juego 2D que utiliza sistemas de partículas.

4. Proyectiles



Figura 2: *Gang Garrison 2* un shooter 2D basado en Team Fortress 2, distintas clases tienen distintos tipos de proyectiles.

En algunos video juegos los objetos de tipo proyectil son sometidos a simulaciones físicas o aproximaciones. Usualmente en la programación de un juego un proyectil sigue una linea recta o parabólica y en caso de colisión se inicia algún evento. Otros juegos consideran factores que afectan la trayectoria del proyectil tales como resistencia y/o dirección al viento, velocidad de proyectil (en vez de una trayectoria inmediata el proyectil posee una velocidad en espacio), gravedad, entre otros [1].

5. Actividad

Durante esta actividad se busca agregar físicas al juego para mejorar la inmersión y calidad visual del juego. Este laboratorio se enfocara en el uso de simulaciones físicas con el uso de sistemas de partículas.

- Se debe hacer uso de sistemas de partículas para varios objetos en escena.
- Provocar el uso de sistemas de partículas al colisionar un objeto contra otro objeto o proyectil. Esto se puede usar para simular una explosión o cualquier otro efecto visual.

Referencias

- [1] CHIAET, J. Getting on the Ball: How the FIFA 14 Soccer Video Game Finally Got Its Physics Right. <http://www.scientificamerican.com/article/getting-on-the-ball-how-soccer-video-game-got-physics-right/>, 2013.
- [2] MILLINGTON, I. *Game Physics Engine Development*. Morgan Kaufmann, 2007.
- [3] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.
- [4] VAN DER BURG, J. Building an Advanced Particle System. http://www.gamasutra.com/view/feature/3157/building_an_advanced_particle_.php, 2000.



LABORATORIO 15

Vídeo Juegos, Programación, Diseño
Networking, Conexión a Internet y Multijugador

1. Pre-Laboratorio

- Investigar los siguientes términos:
 1. Lag.
 2. Ping.
 3. Game State.
 4. Sincronización.
- De algún juego que conozca analice.
 1. ¿Existe en este juego la posibilidad de jugar con otros jugadores? ¿En qué consiste el modo multijugador en tal caso? ¿Cómo cree que se mantienen ambas instancias del juego sincronizadas?

2. Introducción

Los video juegos multijugador son aquellos donde dos o más personas interaccionan en el mismo juego, la forma en la que interactúan depende el objetivo y alcance del video juego, esta puede ser de forma cooperativa o competitiva [3].

Existen distintas formas de permitir la adición de múltiples jugadores una es de forma local utilizada usualmente en video juegos para computadores personales donde varios jugadores se conectan a través de una red local, otra forma es de forma remota a través del Internet.

La programación del networking en un vídeo juego consiste en administrar la data enviada y recibida a través de diferentes métodos de conexión de tal forma que el software responda apropiadamente a distintas acciones entre varios jugadores [2, p. 355].

3. Programación Networking

Los computadores se comunican unos entre otros utilizando redes de computadoras. La red física es construida a partir de cables y transmisiones radiales, pero lo de mayor importancia es la red virtual de protocolos de software construida por encima de esta red física. La responsabilidad principal del networking en los video juegos es ayudar a jugadores encontrarse unos a otros y mantener el estado de juego sincronizado entre estos. Por desgracia la información toma un tiempo para ser transmitida a través de una red, de tal forma que cuando un computador comunica su estado a otra este estado ya está desactualizado. Cuando los jugadores perciben esta tardanza se le llama *lag* [2, p. 356].

El reto del networking en los video juegos es esconder el lag de los jugadores además de proveer una infraestructura de comunicaciones confiable y segura. Comparado con otras áreas de la programación de video juegos los algoritmos en networking no son extremadamente complejos y tienden a representar una pequeña parte del código final del juego. Sin embargo debido a la existencia de eventos impredecibles, como interferencias, congestión de red, o hackers los programadores deben defender el juego en contra de estos eventos en gran parte del código fuente.

El networking también se complica aún más basado en el hecho de que este interacciona con prácticamente todo el código del juego. El juego debe sincronizar su estado con otros jugadores, el modelo de red afecta indirectamente toda pieza de código que toque el estado del juego. Es por esto que no existe solución única al networking en los video juegos, el diseño del modelo de comunicaciones debe ser específicamente creado para trabajar con las mecánicas del juego y como el juego representa su estado actual [1].

4. Actividad

■ Por hacer.

Referencias

- [1] MARK TERRANO, P. B. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond. http://www.gamasutra.com/view/feature/131503/1500_archers_on_a_288_network_.php?page=2, 2001.
- [2] MORGAN MC GUIRE, O. C. J. *Creating Games*. A K Peters, 2008.
- [3] VALVE. Developers Community Wiki: Source Multiplayer Networking. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.



LABORATORIO 16

Vídeo Juegos, Programación
Testing y Optimizaciones

1. Pre-Laboratorio

- Investigar sobre los siguientes conceptos:
 1. Quality Assurance.
 2. Software Bug.
 3. Pruebas de Funcionalidad (Functionality Testing)
 4. Pruebas de Compatibilidad (Compatibility Testing)
 5. Pruebas Beta (Beta Testing)
- Investigue sobre el proceso de pruebas y los pasos de identificación, reporte, análisis y verificación de bugs.
- Investigar sobre los siguientes conceptos:
 1. Batch Rendering.
 2. Frustum Culling.
 3. Occlusion Culling.

2. Proceso de Pruebas en Video Juegos (Game Testing)

El proceso de pruebas es parte del desarrollo de todo video juego. Los video juegos son piezas de software de gran complejidad y como toda pieza de software estos pueden poseer cientos de bugs el proceso de pruebas consiste en identificar, reportar, analizar y

verificar distintos bugs este proceso es realizado por distintos grupos de prueba con variados conocimientos de las mecánicas internas del juego durante y al final del desarrollo del juego. En esta sección se explican distintos enfoques para realizar el proceso de pruebas.

2.1. Unit Texting / White Box Testing (Pruebas por Unidades / Pruebas de Caja Blanca)

Esta es la forma mas sencilla para realizar pruebas sobre cualquier pieza de software. Consiste en que una vez finalizado el software, se debe escribir una serie de pruebas en distintas situaciones del estado del programa presentando todo un rango de valores de entradas validos y no validos para luego escribir por cada caso cual produce algún resultado inesperado o correcto [1, p. 155]. Esta clase de pruebas son usualmente realizadas por personas con conocimiento interno de las mecánicas del juego e incluso manejo del código fuente.

2.2. Black Box Testing (Pruebas Caja Negra)

Este tipo de proceso de prueba es comúnmente usado en los últimos pasos del proceso de desarrollo del video juego o incluso cuando este ya se considera terminado; se realizan sobre un grupo pequeño de personas y estas no deben tener conocimiento alguno del código fuente o la lógica interna de todas las mecánicas del juego. Consiste en simplemente presentar la pieza de software a un grupo de prueba para que estos prueben el programa y reporten algún problema que hayan encontrado durante su experiencia. Es común presentar al grupo de pruebas una lista de tareas que hacer durante el juego y un formato estricto de reporte para los problemas, inconvenientes o bugs que estos encuentren.

2.3. Beta Testing (Pruebas Beta)

Beta Testing es muy parecido a Black Box Testing pero estas pruebas son realizadas sobre personas ya interesadas en el juego y sobre un grupo mucho mayor y no existe ninguna lista tareas para el grupo seleccionado. Los desarrollados deben proveer un sistema de reporte de bugs estricto e intentar corregir todos los bugs encontrados durante el proceso de pruebas. El proceso de pruebas beta puede ser cerrado (closed beta) o abierto (open beta),

la diferencia consiste en que mientras open beta es abierto a cualquier persona que quiera participar closed beta esta limitado a un numero especifico de personas.

3. Proceso de Optimizaciones

Como toda pieza de software los video juegos también están propensos a sufrir de bajo rendimiento en algunos de sus componentes e incluso todo el software en total. El proceso de optimizacion a nivel interno es totalmente dependiente del framwork o motor utilizado, muchos motores como por ejemplo Unity3D ya poseen algoritmos de optimización comunes como occlusion culling [8], frustum culling [5] y batch rendering [6] mientras que en otros estos algoritmos deben ser implementados.

Sin embargo existen algunas técnicas comunes que pueden ayudar a mejorar el rendimiento de cualquier juego, estas son explicadas en esta sección.

3.1. Texturas Atlas

Una textura atlas es una imagen que contiene variadas sub-imágenes donde cada una de estas sub-imágenes es la textura de alguna parte de un modelo 2D o 3D. Estas sub-texturas son renderizadas modificando las coordenadas de texturas en el mapa uv del objeto en la textura atlas, en otras palabras diciéndole en qué parte de la imagen se encuentra la textura del objeto. Esta técnica es particularmente eficiente para casos donde se usan muchas pequeñas texturas que son usadas constantemente, esto reduce de manera significante cambiar el estado del render (render state) cada vez que se tiene que asociar una nueva textura ya que ahora se hace una sola vez, la desventaja es que ahora existe la posibilidad de tener texturas (guardadas en la textura atlas) que tal vez no estemos utilizando alojadas en memoria de GPU.

3.2. Optimización de Mallados

Muchos modelos utilizadas en video juegos en su estado original posee una complejidad geométrica que puede ser excesiva. La optimización de mallados busca reducir la cantidad de polígonos en un mallado preservando la calidad del mallado original. Existe

una variedad de software, frameworks y motores de juego gratis o pagos que hacen esto, algunos populares son Simplygon [4] y Polygon Cruncher [3].

3.3. Niveles de detalle

No tiene sentido mantener un modelo de extrema calidad cuando este es apenas visible en pantalla, para evitar esto se usan niveles de detalles. Consiste en tener a partir del mayado original (máximo detalle) un numero n de niveles de detalles e intercambiar el mallado del objeto por el mallado correspondiente a el nivel de detalle del objeto según algún factor como por ejemplo distancia del objeto contra la cámara principal. Algunos motores de juego como Unity3D [7] o jMonkeyEngine [2] ya realizan esto internamente en otros esto debe ser programado manualmente, para crear los distintos modelos de cada nivel de detalle se utilizan los mismos frameworks expuestos en *Optimización de Mallados*.

3.4. Lightmapping

No tiene sentido utilizar iluminación dinámica en una escena estática para evitar esto podemos utilizar lightmaps, un light map no es mas que una textura que almacena la iluminación de un objeto. Los lightmaps puede ser generados previamente y almacenados en imágenes para facil acceso o generados al cargar una escena (calculando la iluminación una sola vez en un frame) y almacenados en una textura para ser aplicada inmediatamente. Una ventaja de los lightmaps es que al estos no ser generados por cada frame sino una sola vez entonces podemos incluir efectos de gran complejidad como iluminación global que serian muy lentos de computar en tiempo real.

3.5. Compresión de Texturas y Mipmaps

Utilizar texturas comprimidas reduce el tamaño de las texturas resultando en tiempos de carga mucho mas rápidos y un menor uso de la memoria, el uso de texturas mas pequeñas puede incluso aumentar la rendimiento del renderizado en general.

Una mipmap es una secuencia de texturas pre-calculadas y optimizadas donde cada textura es una representación de la imagen original con una resolución progresivamente

menor. Todo motor de juego, framework o biblioteca debe proveer una opción para activar mipmaps en una textura.

4. Actividad

Por hacer.

Referencias

- [1] BETHKE, E. *Game Development and Production*. Wordware Game Developer's Library, 2003.
- [2] JMONKEYENGINE. Level of Detail (LOD) Optimization. http://wiki.jmonkeyengine.org/doku.php/jme3:advanced:level_of_detail.
- [3] MOOTTOOLS SOFTWARE. Polygon Cruncher SDK. <http://www.mootools.com/plugins/us/3dlibrary/>.
- [4] MOOTTOOLS SOFTWARE. Simplygon: Automagic 3D Optimization. <https://www.simplygon.com/>.
- [5] UNITY TECHNOLOGIES. Unity Manual: Camera. <http://docs.unity3d.com/Manual/class-Camera.html>, 2014.
- [6] UNITY TECHNOLOGIES. Unity Manual: Draw Call Batching. <http://docs.unity3d.com/Manual/DrawCallBatching.html>, 2014.
- [7] UNITY TECHNOLOGIES. Unity Manual: Level of Detail. <http://docs.unity3d.com/Manual/LevelOfDetail.html>, 2014.
- [8] UNITY TECHNOLOGIES. Unity Manual: Occlusion Culling. <http://docs.unity3d.com/Manual/OcclusionCulling.html>, 2014.