ESCUELA POLITECNICA DEL EJÉRCITO

DPTO. DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMATICA

"EVOLUTION"
3D GAME DEVELOPMENT PROJECT

Previa a la obtención del Título de:

INGENIERO EN SISTEMAS E INFORMÁTICA

POR: **Natalia Ortiz**
Christian Viteri

SANGOLQUI, 22 de Mayo 2009

CERTIFICACION

Certifico que el presente trabajo fue realizado en su totalidad por los Srs. NATALIA VERENICE ORTIZ DUQUE Y CHRISTIAN ANDRÉS VITERI PROAÑO como requerimiento parcial a la obtención del título de INGENIEROS EN SISTEMAS E INFORMÁTICA.

Sangolquí, 22 de Mayo del 2009.

_____

Ing. Geovanny Raura.

DEDICATORIA

A mis queridos padres quienes creyeron en mí e hicieron posible este momento.

A mis hermanos quienes con su apoyo y respaldo me ayudaron a cumplir una meta más.

A todos mis amigos y compañeros que con sus vivencias y consejos aportaron en mi crecimiento y me motivaron a salir adelante.

A mi gordo bello por todo el apoyo en los momentos más difíciles, por su comprensión y paciencia.

A Dios que siempre será el guía de mi camino y la inspiración de todos mis triunfos.

Natalia Ortiz D.

DEDICATORIA

Principalmente, agradezco a Dios y a mis padres ya que con sus enseñanzas, esfuerzos, sacrificios y amor diario me han llevado al lugar donde me encuentro para lograr todas las metas que me propuse. A mis hermanos que con su apoyo me han ayudado en los momentos más difíciles a los que he enfrentado.

A mi novia y amigos que con sus palabras de aliento supieron ayudarme durante todas las etapas de mi desarrollo humano y profesional.

Agradezco de manera especial a Nexxt Generation Studios que nos brindaron su apoyo y colaboración con la elaboración del presente proyecto.

Christian Viteri

AGRADECIMIENTOS

Queremos expresar nuestro más sincero agradecimiento a todas aquellas personas que de manera directa o indirecta contribuyeron al desarrollo de este proyecto, en especial a Hugo Muñoz, Jamil Falconi y Felipe Pareja por su ayuda, disponibilidad y sobre todo por su don de amigos.

A nuestros familiares, amigos y profesores por su lealtad y ánimos.

Un agradecimiento especial a nuestro Director de Tesis Ing. Geovanny Raura así como a nuestro Co–Director Ing. Danilo Martínez que en todo momento supieron ayudarnos y guiarnos para culminar este proyecto.

A todas las personas que nos apoyaron con sus valiosos comentarios, opiniones y sugerencias.

Natalia Ortiz
Christian Viteri

RESUMEN

El presente proyecto consiste en llevar a cabo el diseño y construcción de una aplicación tridimensional interactiva denominada “Evolution”, cuyo objetivo primordial es el crear un videojuego con un enfoque educativo, manteniendo aspectos claves como la creatividad y la diversión, de tal forma que ayude a crear conciencia sobre un tema tan importante como lo es el cuidado de la Naturaleza. Evolution simula un medio ambiente en donde los habitantes de un planeta tratan de sobrevivir ante los posibles fenómenos naturales que amenazan su evolución. En este entorno el usuario es capaz de producir y controlar fenómenos naturales de acuerdo al comportamiento de la población dentro del juego donde su meta principal es lograr que los habitantes de la pequeña civilización aprendan a sobrellevar estas dificultades para poder evolucionar a la siguiente fase. Evolution es un videojuego multiconsola, desarrollado utilizando la herramienta Microsoft XNA Framework 2.0, que aplica conceptos como la teoría de juegos y patrones de comportamiento e inteligencia artificial para definir la conducta de los personajes que conforman el videojuego, y que en conjunto con las herramientas de diseño 3D Studio Max y Torque Game Engine Advanced contribuyan al diseño de este entorno tridimensional controlado.

ÍNDICE DE CONTENIDOS

LISTADO DE GRÁFICOS

LISTADO DE TABLAS

CAPITULO I

Generalidades

1.1. Introducción

El crecimiento acelerado de la población, el vertiginoso avance tecnológico, y la sociedad incentivada por la mejora de la calidad de vida, han traído como consecuencia la explotación inconsciente y acelerada de los recursos naturales, la degradación de nuestro hábitat y la destrucción de zonas protegidas únicas a nivel mundial.

En los últimos años el calentamiento global ha influido en la disminución de los glaciares, cambios climáticos, así como la devastación de zonas protegidas; acabará en poco tiempo con el único legado que se podrá dejar a nuestros hijos, su hogar, la naturaleza; es por esto que recientemente la sociedad se ha visto obligada a buscar nuevas formas de concientización frente a estas amenazas. El compromiso es plantear soluciones aplicables y sensatas ante problemas cotidianos en pro del medio ambiente.

Por otra parte, el uso y conocimiento de herramientas para la construcción de videojuegos es escaso dentro del Ecuador debido a que en el país no se ha incursionado en este campo además esta tecnología tiene un alto grado de complejidad y podría resultar muy costosa.

En base a lo expuesto se plantea la construcción de un videojuego que abarque todos los problemas antes mencionados y teniendo como objetivo final el aprendizaje de una manera original y divertida acerca del cuidado y conservación del medio ambiente.

1.2. Antecedentes

En el Ecuador el desarrollo y aplicación de tecnologías de entretenimiento es prácticamente nulo debido a la complejidad, falta de información y al elevado costo que conlleva el elaborar este tipo de proyectos.
A esto se suma el escaso avance tecnológico, haciendo aún más difícil que empresas inviertan sus recursos en la industria del juego.

Para empezar, el desarrollo de videojuegos demanda un alto nivel de conocimientos en diversas áreas pues no sólo se centra en la programación y en el uso de herramientas para creación de software, sino más bien su concepto se extiende a otras ramas como es el diseño e ingenio del equipo a cargo del proyecto.

Adicionalmente no existen centros de enseñanza capacitados para brindar el nivel adecuado que es requerido para una persona o empresa que pretenda incursionar en el tema.

Otro problema y quizás el más importante es el presupuesto que se requiere para llevar a cabo los procesos de preproducción, producción y postproducción que contemplan las fases de creación y modelado de los elementos visuales, del código y del marketing, donde en conjunto alcanzan cifras millonarias, como es el caso de los juegos más populares. Por ejemplo, la producción de Halo 3 costo casi 30millones de dólares mas otros casi 20 de publicidad, mientras que el costo Gears of War fue de 10 millones de dólares. Siendo ambos juegos de Microsoft para el Xbox 360.

Simultáneamente existe otra penosa realidad en lo que se refiere al cuidado y conservación del medio ambiente, la cual tiene como origen la indeferencia y la

despreocupación por parte de la sociedad que han causado la devastación sin medida del ecosistema ya que actualmente no existen métodos adecuados para concienciar a todos los grupos sociales y fomentar una cultura que participe activamente en pro de la naturaleza y sus derivados.

1.3. Objetivos

Objetivo General

Analizar, diseñar, desarrollar e implementar un entorno tridimensional controlado enfocado al medio ambiente, basado en XNA 2.0 y herramientas de animación 3D orientado a la programación visual.

Objetivos Específicos

- Analizar y desarrollar los aspectos importantes sobre la teoría de juegos, algoritmos, fórmulas matemáticas a ser implementadas así como el Framework XNA y los patrones de comportamiento relacionados con la inteligencia artificial a ser desarrollada para la construcción de la Aplicación.

- Diseñar gráficamente guiones, modelos y caracteres que ayuden a generar un concepto o visión del juego.

- Diseñar y construir un motor gráfico que se adapte a las necesidades del sistema.

- Definir una metodología que permita administrar de manera adecuada el manejo de tiempo y recursos.

1.4. Visión y Alcance

1.4.1 Visión

El cuidado de la naturaleza puede ser entretenido y fácil, incentivando el aprendizaje mediante el uso de videojuegos. Con el propósito de aportar soluciones viables ante problemas críticos afines a nuestro ecosistema, se pretende contribuir con un nuevo método de enseñanza que puede captar la atención de un mayor número de personas, para que se fomente el cuidado del entorno, aprovechando de manera consciente y ordenada los recursos naturales; en consecuencia logrando con ello un progreso sostenible.

Todo esto con el fin de crear una sociedad que proteja su entorno así como sus riquezas naturales mediante instrumentos tecnológicos que aseguren la prevención y control de la contaminación y un aprovechamiento sustentable de los recursos ambientales; fortaleciendo principios que satisfagan las necesidades actuales de la humanidad, sin arriesgar la capacidad de las generaciones futuras para atender sus propios compromisos que aseguren su existencia.

1.4.2 Alcance

El mundo esta sufriendo muchos cambios debido a la acción del hombre; cambios que de alguna manera u otra desequilibran la normalidad del mismo, y por supuesto nuestra vida.

Teniendo en consideración lo antes expuesto y debido a que este tema es de interés global se puede lograr una concienciación masiva mediante el uso de tecnología de punta, así como nuevos mecanismos para atraer la atención de la gente como lo es en este caso el uso de videojuegos.

Para lo cual, se plantea diseñar y construir una aplicación tridimensional interactiva denominada “Evolution”, cuyo objetivo primordial será crear un videojuego con un enfoque educativo, manteniendo aspectos claves como la creatividad y la diversión, de tal forma que ayude a crear conciencia sobre un tema tan importante como lo es el cuidado de la Naturaleza.

Evolution permitirá simular un medio ambiente en donde los habitantes de un planeta tratarán de sobrevivir ante los posibles fenómenos naturales que amenacen su evolución, teniendo como meta principal buscar la concienciación de la colectividad sobre problemas ambientales cotidianos a los cuales no se les ha dado la debida importancia. En este entorno el usuario será capaz de producir y controlar fenómenos naturales de acuerdo al comportamiento de la población dentro del juego; permitiéndole conocer e indagar acerca de los efectos positivos y negativos que el comportamiento de una sociedad provocaría en la naturaleza.

Para lograr este fin el proyecto comprende los siguientes pasos:

Conceptualizar la creación del juego mediante diagramas y diseños conceptuales.

Diseñar y crear todos los elementos visuales que conformaran al juego, es decir todos los modelos, animaciones, interfaces, etc.

Diseñar e implementar un motor gráfico que simule factores como:

- Emisión de Partículas
- Shaders y sombras
- Gravedad
- Detección de colisiones

Implementar Inteligencia Artificial para el comportamiento de los personajes que conforman el juego, así como algoritmos matemáticos que permitan crear los diferentes modelos a usarse en el juego.

Aplicar una metodología que permita elevar al máximo la productividad del equipo de desarrollo, de tal manera que permita alcanzar los objetivos trazados para el proyecto.

Someter a pruebas el sistema para conseguir un producto final libre de errores que puedan afectar su buen desempeño.

Evolution implementará el uso tanto de dispositivos (hardware) como de aplicaciones (software) de última generación las mismas que van a permitir que el proyecto tenga una mayor aceptación. En este caso particular se utilizará el framework XNA para el desarrollo y construcción del aplicativo en conjunto con las herramientas de diseño; de tal manera que el producto final pueda ser implementado en computadores así como en la consola de Video Juegos de tercera generación lanzada por Microsoft (XBox 360) para mejorar la experiencia del jugador en cuanto a la calidad gráfica así como en factores de operabilidad.

1.5. Justificación

Debido al acelerado ritmo de vida de la sociedad y al constante cambio que cada generación demanda, ésta ha centrando su atención en temas que en la actualidad se les da mayor importancia como política, economía, avances tecnológicos, etc., dejando a un lado el cuidado del medio ambiente y a pesar de que es un problema del que todos forman parte, es muy poca la gente que realmente hace algo por cambiar esta realidad.

Es por esto que se busca brindar un aporte que contribuya en la dura tarea de concienciar a la sociedad acerca del respeto y conservación del medio ambiente mediante el uso de las nuevas tecnologías de la información y comunicación (TIC's).

Siendo esta una iniciativa nueva y creativa se pretende atraer a más personas e involucrarlas en la lucha por mejorar el entorno y darles una pauta para que a futuro se sigan desarrollando ideas similares en beneficio de todos.

También es importante afrontar temas actuales y polémicos como el consumo excesivo de recursos así como la contaminación ambiental, los cuales conlleva a un análisis profundo acerca de la situación actual del medio ambiente y de la forma en que se puedan combatir las diferentes amenazas que afectan al ecosistema.

El proyecto tiene un enfoque a la aplicación de herramientas de programación visual totalmente nuevas y de libre distribución; las cuales en nuestro país no han sido explotadas debido a varios aspectos como un mercado sin competencia, poco conocimiento técnico y enseñanza académica y un elevado presupuesto económico para su desarrollo, pero que son de gran utilidad para entusiastas programadores de aplicativos visuales.

Además, un factor vital para la elaboración del proyecto es el alcance poblacional que se va a poder cubrir, ya que no solo abarcaría un grupo reducido de usuarios; por el contrario, no tiene fronteras haciendo partícipes a todos los grupos que conforman nuestra sociedad, sin distinción de género y prácticamente para cualquier edad.

Finalmente, según análisis realizados por “Entertainment Software Association”[1], compañía dedicada al estudio de todo lo que involucra la industria de videojuegos, se puede apreciar que el 83% de la población (entre 18 y 40 años) adquieren videojuegos, y de los cuales un 53% planea seguir jugando por 10 años o más, por lo tanto, el uso de juegos no se limita solo a niños y adolescentes, pues ellos conforman un cierto porcentaje, sino a la mayoría de las personas como se muestra a continuación en los gráficos 1 y 2:

Edad (Años)	Porcentaje %
Por debajo de los 18	28.20%
Entre 18 y 49	47.60%
Mayor a 50	24.20%

Tabla 1. Edad y Porcentajes de Jugadores

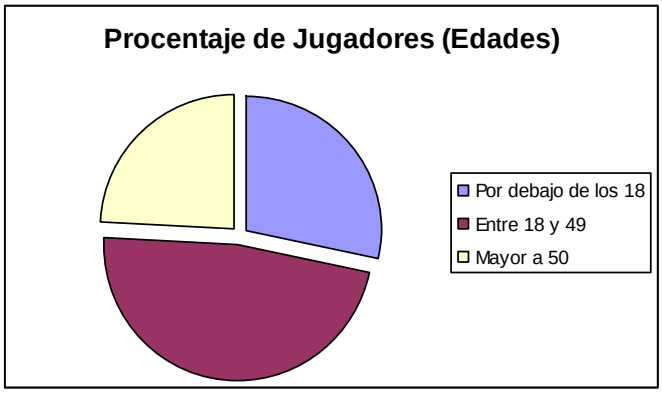


Gráfico 1. Porcentaje de Jugadores.

Genero	Porcentaje %
Femenino	38%
Masculino	62%

Tabla 2. Género y Porcentajes de Jugadores

[1] Entertainment Software Association, Disponible: http://www.theesa.com/facts/gamer_data.php [Consulta: 2008, Marzo 2]

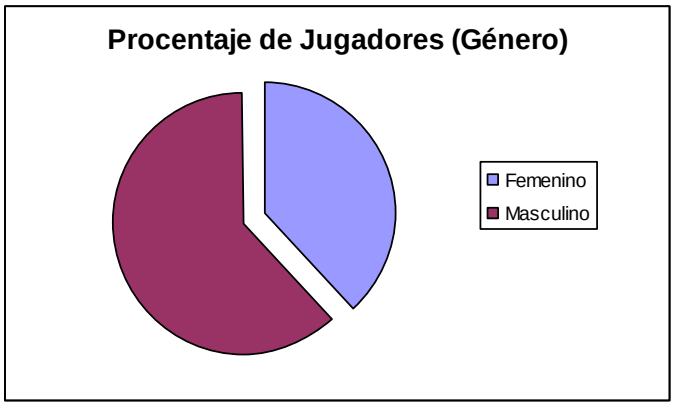


Gráfico2. Porcentaje de Jugadores por Género.

1.6. Descripción General de "Evolution"

El proyecto denominado "Evolution", es en si una aplicación visual interactiva la cual tiene como objetivo principal la concienciación de la sociedad acerca del cuidado y protección del medio ambiente, por lo cual el producto obtenido encierra estos aspectos para que el usuario tenga una noción clara del enfoque hacia la que va dirigida la aplicación.

El usuario aprenderá y de cierta manera tomará conciencia de una forma interactiva y entretenida aspectos de carácter más serio, topando temas relacionados con el medio ambiente que últimamente no se los toma muy en cuenta o que prácticamente son ignorados por la gran mayoría de personas.

Evolution está en la clase de juegos de estrategia, el cual trata acerca de una pequeña sociedad de individuos que intentan sobrevivir en un medio utilizando los recursos disponibles según sus necesidades. El jugador toma el rol de un ser supremo capaz de controlar el destino de sus vidas alterando y produciendo cambios climáticos y fenómenos naturales como son erupciones volcánicas, terremotos, inundaciones, lluvias, sequías, etc. Teniendo como objetivo primordial que los habitantes de la pequeña civilización aprendan a sobrellevar estas dificultades para poder evolucionar a la siguiente fase.

Dentro del juego se encuentran tres niveles de dificultad diferentes: el primero se remonta a la época prehistórica teniendo un nivel de dificultad básico debido a que existen menos factores que afectan el curso normal de la naturaleza. El segundo entorno se desarrolla en la época actual teniendo un nivel medio de dificultad, debido a que existen más factores que afectan la supervivencia de la civilización, como por ejemplo la polución, erosión, etc. Por último, se tiene el tercer entorno localizado en el futuro teniendo un nivel avanzado de dificultad debido a problemas y factores que traen consigo la evolución tecnológica de la sociedad.

Finalmente, la aplicación tiene como objetivo ser un videojuego multiconsola, es decir que el mismo puede ser utilizado tanto en un computador como en una consola de video juegos, en este caso particular el uso de la consola “Xbox 360” de Microsoft.

CAPITULO II

Fundamentos y Marco Teórico previo al diseño y desarrollo de “Evolution”

2.1. Prerrequisitos a Considerar para el Desarrollo de Juegos

Para empezar un proyecto que involucra el desarrollo de videojuegos es necesario contar con ciertos requisitos previos en lo que se refiere a Hardware, Software, Conocimientos, Tiempo y Personal.

- **Hardware y Software**

  El hardware y software necesarios para un proyecto de esta magnitud va de acuerdo a la complejidad y al tipo de juego. Siendo la demanda de recursos en el caso de juegos de dos dimensiones inferior a la de una en tres dimensiones, debido a que esta última conlleva el uso de herramientas que operan con recursos tecnológicos más costosos.

  A continuación se detallan los requerimientos para el proyecto Evolution tanto de hardware como de software:

  Requisitos de Hardware

  En el caso del hardware es esencial contar con tarjetas de video que soporten las demandas del juego en cuanto a efectos. En el caso de Evolution, la tarjeta debe disponer de la versión Shader 1.0, así como 64Mb de memoria como mínimo para que ésta pueda funcionar correctamente en una consola Xbox360.

  - Computadores con Tarjetas de Video:
    - Nvidia Geforce 8400 GT (512Mb)

    - Nvidia Geforce 6200 Turbo Caché TM (256Mb)
- Mínimo 1GB en memoria RAM.
- Capacidad de por lo menos 40 Gb en disco duro
- Procesador Intel Pentium 4 de 3.0Ghz. en adelante.

Requisitos de Software

En cuanto a software es necesario tener herramientas de diseño, desarrollo y postproducción. En este caso en particular para la parte de diseño se utilizarán herramientas 2D y 3D como son Photoshop CS (8.0) y AutoDesk 3D Studio Max 9. Para la parte de desarrollo se utiliza Microsoft Visual Studio .Net 2005 en conjunto con el API XNA 2.0 de Microsoft.

Adicionalmente se requiere el siguiente software:

- Torque Game Engine Advanced 1.7.2
- Microsoft DirectX SDK 9.0 c
- IBM Rational
- Microsoft Project
- Microsoft Office 2007
- Windows Vista

- **Conocimientos**

  El equipo de desarrollo debe estar en la capacidad de manejar tanto el hardware como el software mencionados anteriormente para poder empezar con el proyecto.

  Además se debe contar con materiales de capacitación adecuados como son tutoriales, libros y manuales de uso.

- **Personal**

  Se requiere personas aptas y hábiles para desempeñar las funciones de investigación, diseño y desarrollo en cada etapa del proyecto.

- **Tiempo**

  En Scrum, en la etapa de Planificación Del Sprint, se realizan estimaciones sobre los ítems del Backlog Del Producto.

  La técnica empleada para realizar la estimación en este tipo de proyectos es la Planificación De Póker. Esta permite orientar a los Miembros Del Equipo De Scrum y el Scrum Master dando una idea del tiempo que durará la realización del proyecto, para lo cual es necesario calcular los siguientes valores:

  Estimación de la Velocidad del Equipo: La velocidad del equipo se estima multiplicando la cantidad de integrantes por la cantidad de días hábiles del Sprint. Con esto se obtiene la cantidad total de días ideales disponibles para el Sprint.

Parámetros	**Valor**
Integrantes	2
Días Hábiles en el Sprint	360

Días Ideales: 720 días.

Factor de Foco: El factor de foco es un coeficiente que ajusta la estimación anterior, para poder acercarla a la realidad (teniendo en cuenta demoras, distracciones, errores, etc.).

El factor de foco suele estar entre 50% - 80%. A medida que se va avanzando los Sprint, se puede tomar el último factor de foco, o un promedio de los últimos 2-3 Sprint.

ff(1) : factor de foco del Sprint 1.
ff(1) : (ff (1-2) +ff(1-1) / 2 = 0.5
ff(2) : factor de foco del Sprint 2.
ff(2) : (ff (2-2) +ff(2-1) / 2 = 0.5

ff(3) : factor de foco del Sprint 3. ff(3) : (ff (3-2) +ff(3-1) / 2 = 1 ff(4) : factor de foco del Sprint 4. ff(4) : (ff (4-2) +ff(4-1) / 2 = 2.5
ff(n) = 1.125.

Velocidad en Etapas de Scrum	Días
Investigación	30 días
Velocidad estimada al comienzo del Sprint 1	24*1.125= 27 días
Velocidad estimada al comienzo del Sprint 2	35*1.125= 39 días.
Velocidad estimada al comienzo del Sprint 3	38*1.125= 43 días.
Velocidad estimada al comienzo del Sprint 4	25*1.125= 28 días.

Ajuste al final del Sprint: Al finalizar el Sprint se tiene la velocidad real que tuvo el equipo. Así, se tienen dos valores: la velocidad estimada al comienzo del Sprint y la velocidad real al finalizar el Sprint. Esta velocidad real al finalizar el Sprint deberá ser la velocidad estimada para el próximo Sprint.

A su vez, deberá ajustarse el factor de foco para ver su valor real. Este nuevo factor de foco será útil en el caso de que se modifique la cantidad de integrantes del equipo para el siguiente Sprint, para poder recalcular la velocidad del mismo.

El resultado final de estimación del proyecto será:

Velocidad Final en Etapas de Scrum	Días
Investigación	30 días
Velocidad Real al Final del Sprint 1	40 días
Velocidad Real al Final del Sprint 2	60 días
Velocidad Real al Final del Sprint 3	90 días
Velocidad Real al Final del Sprint 4 **Documentación**	20 días 30 días
Total	270 días.

El tiempo que se ha estimado para la realización del proyecto es de nueve meses, período que se ha establecido de acuerdo al tipo de aplicación que va a ser desarrollada y tomando en cuenta los

conocimientos tanto de investigadores y desarrolladores que van a realizar este proyecto.

En el siguiente cuadro (figura 1) se detallan las etapas del proyecto, la duración de cada una y el personal asignado a cada tarea.

Las fases del proyecto son consisten en pequeñas iteraciones de varias semanas denominadas Sprints donde se va implementando la funcionalidad que se ha decidido al comienzo de cada una de esas iteraciones.

Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
Evolution XNA 3D Game Development Project	**197 días**	**lun 02/06/08**	**mar 03/03/09**		
Investigación	21 días	lun 02/06/08	lun 30/06/08		Investigadores
Sprint1	22 días	mar 01/07/08	mié 30/07/08	2	Director_Codirector_Investigadores
Sprint2	28 días	jue 31/07/08	lun 08/09/08	3	Desarrolladores_Investigadores
Sprint3	44 días	mar 09/09/08	dom 09/11/08	4	Investigadores_Técnicos
Sprint4	67 días	lun 10/11/08	mar 10/02/09	5	Investigadores_Desarrolladores
Documentación	15 días	mié 11/02/09	mar 03/03/09	6	Investigadores_Desarrolladores

Figura 1. Asignación del las Tareas Proyecto Evolution.

Este cronograma describe las personas que serán parte activa en cada tarea del proyecto como se muestra en la figura 2.

Figura 2. Tiempo estimado de duración de las tareas.

2.2. Pasos fundamentales para el Desarrollo de Juegos

Para empezar el desarrollo de un videojuego es necesario tomar en cuenta ciertos pasos, los cuales desempeñan una función específica en cada etapa de la producción del juego.

Dichas etapas se encuentran contenidas dentro de tres fases principales siendo éstas: fase de preproducción, producción y postproducción. Las cuales se detallan a continuación:

Fase de Preproducción

Dentro de esta fase se estructuran las ideas generales del juego, así como su temática, las cuales dan una noción a los diseñadores y programadores de lo que se quiere alcanzar al final del proyecto.

En el caso de Evolution la fase de preproducción se divide a su vez en cinco etapas que son:

- **La idea del Juego**

  En esta etapa se plantea el tema y el concepto que definen las bases y la modalidad del juego. En este caso el nombre escogido es “Evolution” debido a que se da un enfoque a la evolución de la humanidad y las consecuencias que esto conlleva. El concepto se basa en la concienciación del cuidado al medio ambiente, teniendo como modalidad la educación y aprendizaje.

- **La Historia**

  En este punto se describen los objetivos, el rol de los personajes y los niveles en los que se desenvuelve el juego. En Evolution el objetivo principal es la supervivencia de una civilización la cual se enfrenta a las amenazas producidas por la naturaleza. El rol de cada habitante es aprender a defenderse de las adversidades que se le presenten, buscar su comida y mantenerse a salvo de cualquier alteración del medio en el que se desenvuelve y así pasar al siguiente nivel del

juego. Evolution se compone de tres niveles los cuales difieren en el tiempo y la dificultad, siendo estos: El primer nivel, cuya historia se sitúa en el pasado teniendo un nivel de dificultad básico; el segundo nivel situado en el presente con un nivel medio de dificultad, y por último el tercer nivel ubicado en el futuro, con un nivel avanzado de dificultad.

- **Tipo de Juego**

El videojuego será desarrollado en su totalidad en tres dimensiones, teniendo como base el tipo RTS (Real Time Strategy), haciéndolo netamente estratégico y permitiendo el control de hasta un jugador.

- **Sistema de Visualización**

Esta parte describe la forma en que el jugador visualiza los escenarios y la ejecución de sus acciones.

Evolution está diseñado para ser jugado en tercera persona debido a que se mantiene el control de todo el entorno ambientado en cada nivel.

- **Complementos del Juego**

Evolution acopla en su modo de juego el control y administración de los eventos naturales más conocidos como son: lluvias, inundaciones, terremotos, incendios, erupciones volcánicas, entre otros; mientras que la civilización de individuos subsiste por sus propios medios sin que el jugador tome el control sobre ellos, lo cual lo hace diferente del resto de juegos ubicados dentro de esta categoría.

Fase de Producción

Dentro de esta fase se dan hechos más concretos, es decir aquí empieza la etapa de diseño y desarrollo donde los especialistas se organizan para llevar a cabo las ideas que fueron expuestas en la primera fase.

- **Etapa de Diseño**

  La etapa de diseño permite plasmar gráficamente los elementos que constituyen el juego. En el caso de Evolution se ha tomado en consideración a tres subetapas que son: Diseño de Arte, Diseño de Mecánicas y Diseño de Producción.

  Diseño de Arte: En esta categoría se realizan todas las tareas artísticas del juego.

  - **Guión:** Consiste en relatar la historia del juego en base a bosquejos que ilustren de manera conceptual la funcionalidad del juego, así como sus caracteres, escenarios e interfases de usuario.

    El guión de “Evolution” muestra la funcionalidad, el manejo de menús, así como los personajes y escenarios tentativos para el juego, como se observa en las figuras 3, 4 y 5:

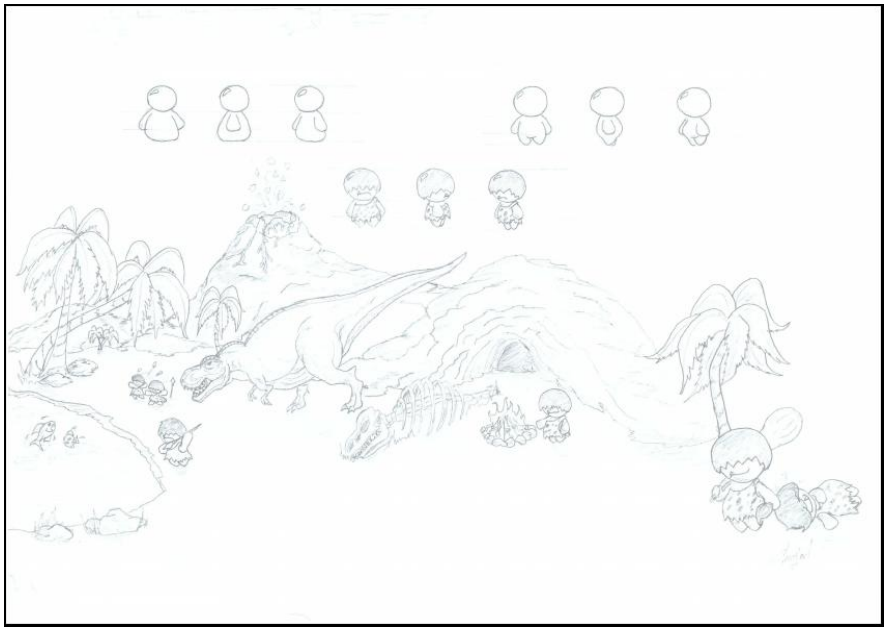


Figura 3. Story Board Personajes

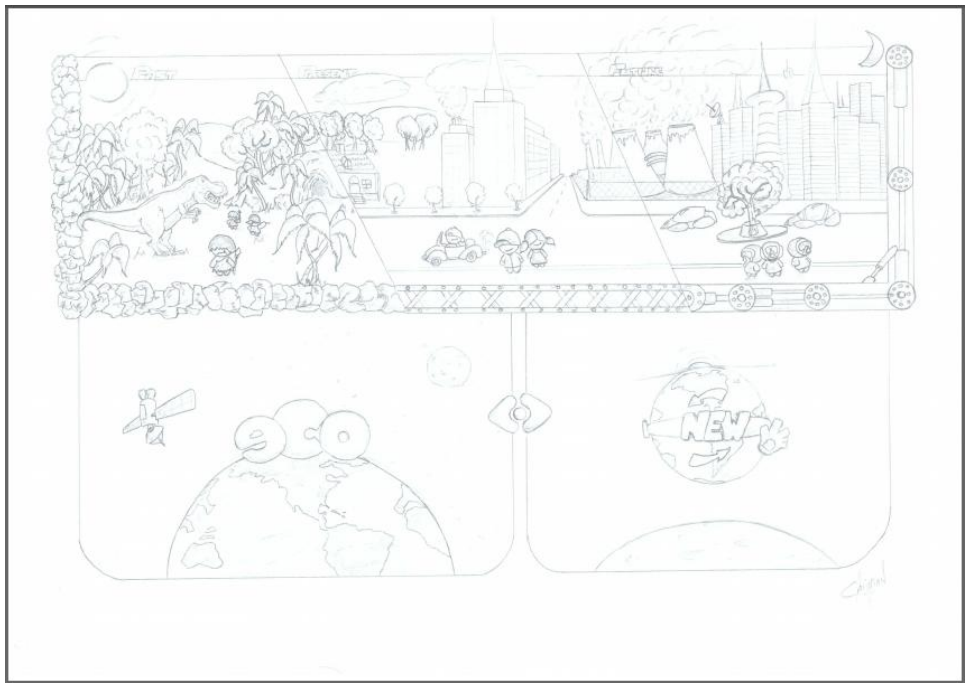


Figura 4. Story Board Diseño de Niveles

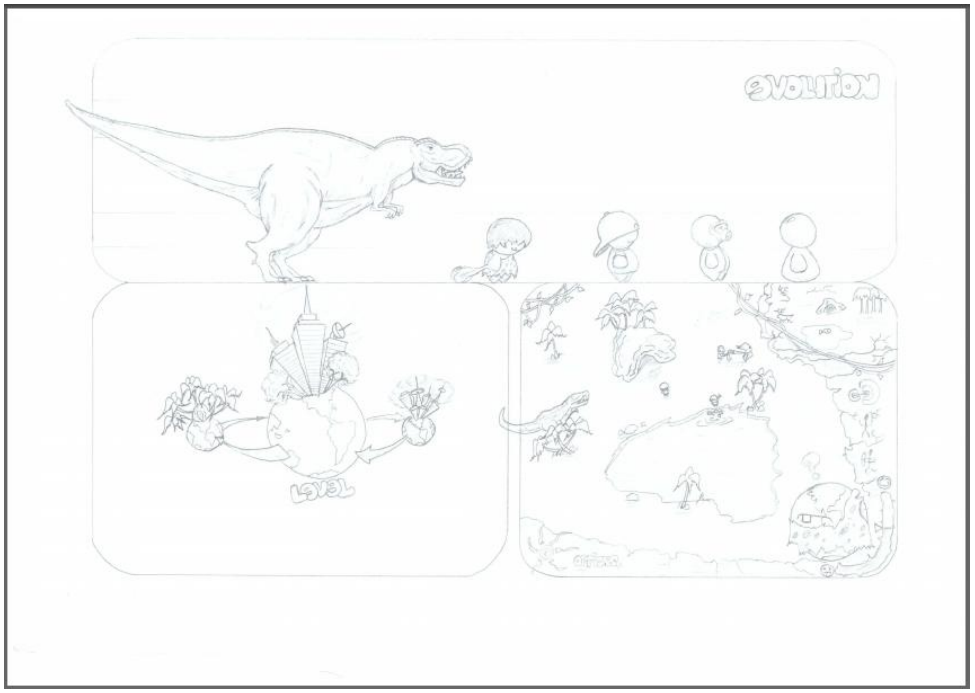


Figura 5. Story Board Diseño de Personajes y entorno

- **Sonido:** El sonido es una parte esencial a la hora de desarrollar juegos ya que este brinda al usuario la sensación de realismo captando su atención y haciéndole sentir parte del entorno con el que está interactuando.

  Los sonidos dentro de Evolution difieren según el escenario, por ejemplo en el primer nivel que se sitúa en la prehistoria los sonidos serán netamente ambientales a diferencia de los niveles posteriores donde se encontrarán sonidos propios de una ciudad moderna.

- **Interfaz:** Aquí se especifican la lógica y secuencia de menús, de tal manera que la interfaz sea agradable y fácil de manejar para el usuario.

  De la misma forma la interfaz gráfica del usuario dentro del juego debe ser comprensible y práctica distribuyendo de una manera adecuada las opciones, estadísticas, ayudas y demás funciones que el usuario requiere para poder desenvolverse apropiadamente.

  Evolution contará con menús en 2D para el menú principal y para las opciones de configuración. También contará con menús en 3D, destinados a la jugabilidad y otros para tutoriales.

- **Gráficos:** Dentro de esta categoría se establecen los parámetros tanto en modelos de 2D y 3D, los cuales parten de un modelo conceptual previamente establecido.

  Debido a que Evolution es un juego completamente en tres dimensiones, todos los modelos serán creados en base a los bocetos conceptuales, de tal manera que el resultado satisfaga lo artístico manteniendo la versatilidad en el diseño.

Todos los elementos tridimensionales están optimizados para un mayor desempeño dentro de la aplicación, ya que ellos serán estructurados con la menor cantidad de polígonos sin que esto afecte la calidad del diseño.

Diseño de Mecánicas: Describe el comportamiento de todos los entes que conforman el juego, así como las reglas que lo van a regir.
Las mecánicas y reglas planeadas para Evolution están divididas de la siguiente manera:

- **Reglas del Jugador:** Están destinadas al control de la naturaleza, teniendo en cuenta que se debe tener especial cuidado con cada acción que el jugador realice ya que el es el encargado de la supervivencia de la civilización, por lo cual si se abusa de esta característica se puede causar la muerte de todos los habitantes y por ende la pérdida del juego.

- **Comportamientos del Juego:** Cada habitante actúa de manera libre e independiente interactuando con su entorno y tratando de sobrevivir a las condiciones que se le presenten disponiendo de los recursos según sus necesidades, y conforme a su accionar ya sea positivo o negativo ocasionará que la naturaleza actúe en su contra.

Diseño de Programación: En esta categoría es necesario aplicar una metodología de desarrollo que permita cumplir con los resultados esperados. Además es necesario crear diagramas que permitan describir el funcionamiento dinámico y estático, la interacción con los usuarios y los diferentes estados que atravesará el videojuego como software.

Evolution aplicará la metodología “Scrum” debido a que cumple con las fases de desarrollo esperadas para proyectos de este tipo.

- **Etapa de Desarrollo**

  La etapa de desarrollo contempla la programación, pruebas e implementación de las versiones del juego las cuales se basan en el Diseño de Programación descrito anteriormente. Para lo cual es necesario escoger la herramienta de desarrollo que se va a utilizar.

  En este caso “Evolution” va a ser desarrollado en XNA Game Studio 2.0 que es un framework para la programación de Juegos.

  Análisis y Pruebas

  En esta etapa se deben corregir los errores de las diferentes versiones del juego, y al mismo tiempo, se debe dar énfasis al refinamiento de la jugabilidad; siendo esta la característica fundamental para brindar calidad en este tipo de aplicaciones.

  Las versiones del juego en la etapa de pruebas son:

  - **Pruebas Alpha:** Estas pruebas ayudan a detectar y corregir los errores más críticos y mejorar las condiciones de jugabilidad.

  - **Pruebas Beta:** Estas pruebas permiten obtener una versión estable con defectos irrelevantes que no afecten el correcto funcionamiento y desempeño del juego.

Fase de Post-Producción

En esta fase ya se dispone de una versión final, la cual se va a instalar en la consola XBOX 360 para su ejecución.

2.3. Descripción Parcial de la Teoría de Juegos

La teoría de juegos pertenece al área matemática y su objetivo principal es analizar el comportamiento estratégico de los jugadores que intervienen en el juego, estudiando situaciones que involucran conflictos de intereses, estrategias óptimas, trampas, así como el comportamiento previsto y observado de individuos en juegos para realizar procesos de decisión, buscando siempre un modelo de actuación impecable.

Esta teoría concuerda en algunos puntos con la teoría de la decisión, la cual estudia la toma de decisiones en casos reales, ya sea para realizar experimentos o en casos racionales donde interviene la lógica y la estadística. La teoría de decisión se diferencia de la teoría de juegos, porque en esta el 'adversario' corresponde a la realidad en lugar de otro jugador o jugadores.

Además estudia la elección de conductas adecuadas cuando los beneficios y costos de una opción no están definidos desde un principio, sino que dependen de las elecciones de otros individuos. Como ejemplo de esta definición se puede citar el juego del dilema del prisionero, el cual se centra en comprender el comportamiento humano en situaciones de cooperación.

Los juegos estudiados por esta teoría están definidos por objetos matemáticos. El juego se lo define como un conjunto de jugadores, un conjunto de movimientos o estrategias que están disponibles para cada jugador y las respectivas recompensas a cada combinación de estrategias. Además, se centra en el análisis de situaciones estratégicas, estudiando cualquier situación que los individuos puedan tomar decisiones estratégicas y en la que el resultado final depende de lo que cada uno decida hacer, puede concebirse como un Juego.

Todos los juegos tienen 3 elementos básicos:

- Jugadores: Individuos que intervienen en el juego.
- Estrategias: Especificación completa de las acciones que ejecutará un jugador en cualquier contingencia que pueda presentarse en el desarrollo del juego. Es cada uno de los cursos de acción de cada jugador.
- Ganancias o Pérdidas: Recompensas dadas a cada jugador que identifican al ganador.

Historia de la Teoría de Juegos

La teoría de los Juegos nace de estudios realizados por John Von Neumann y Oskar Morgenstern en 1944 para elaborar estrategias militares aplicando conceptos de destrucción mutua garantizada. Estos estudios establecieron una conexión entre la noción de equilibrio y el punto fijo de una función, con lo que se prueba que todo juego no cooperativo, es decir en el que cada jugador se preocupa únicamente por sus propias ganancias, admite al menos un equilibrio, mientras que un juego cooperativo analizará las estrategias óptimas para cada jugador, asumiendo que pueden establecer acuerdos entre sí acerca de las estrategias más apropiadas.

Equilibrio de Nash

Este concepto ocupa un lugar central en la teoría de juegos. John Nash desarrolló una definición de una estrategia óptima para juegos de múltiples jugadores donde una combinación de estrategias (una por jugador) que está en equilibrio de Nash puede aumentar sus ganancias por un cambio unilateral de estrategia.

El equilibrio de Nash es un conjunto de estrategias tal que ningún jugador puede mejorar su rentabilidad si cambia su estrategia, dadas las estrategias de todos los demás jugadores en el juego, es decir el escenario es estable para todos los jugadores, logrando así un equilibrio entre jugadores ya que ninguno de ellos tiene incentivos para alejarse de su estrategia actual.

El equilibrio de Nash es un equilibrio no cooperativo, donde ninguna de las firmas tiene un incentivo para modificar su decisión de producción. Es decir, la estrategia elegida por cada firma maximiza sus utilidades, dadas las estrategias de sus competidores.

Cuando existen estrategias dominantes, si cada jugador sigue su estrategia dominante se logra un equilibrio de Nash, pero, se puede lograr un equilibrio aún cuando uno de los jugadores no tenga una estrategia dominante.

Nash demostró que no todos los juegos de cooperación tienen un equilibrio, y por tanto, estableció una estructura analítica en el que todas las situaciones de conflicto y la cooperación podría ser objeto de estudio.

Por este trabajo recibió el Premio Nobel 1994 de Economía, junto con John Harsanyi y Reinhard Selten[1].

Este equilibrio permite el análisis de juegos no cooperativos y de los juegos cooperativos.

Estrategia Maximin

En el concepto de equilibrio de Nash es fundamental el supuesto de racionalidad de los agentes. Si un agente sospechara que su adversario no se comporta racionalmente, podría tener sentido que adoptara una estrategia *maximin*, esto es, aquella en la que se maximiza la ganancia mínima que puede obtenerse[2].

Tipos de Juegos

Los juegos pueden ser representados en varias formas de acuerdo a los métodos que se emplean para resolverlos. Existen cuatro clases principales de juegos:

- Juegos en forma extensiva (árbol)
- Juegos en forma estratégica (normal)
- Juegos en forma gráfica

[1] Tomado de: http://www.zonaeconomica.com/teoriadejuegos/tiposdejuego.

[2] Tomado de: Referencia http://www.zonaeconomica.com/teoriadejuegos/tiposdejuego.

- Juegos en forma coalicional o Cooperativo
- Juegos Simétricos y Asimétricos
- Juegos de Suma Cero y de Suma No Cero
- Juegos simultáneos y secuenciales
- Juegos de Información Perfecta y Completa
- Juegos de Longitud Infinita

Las tres primeras clases de juegos se analizan en la teoría de juegos no cooperativos y la cuarta corresponde a los juegos cooperativos.

- **Forma Normal del Juego:** Definida también como forma estratégica del juego, consiste en una matriz que contiene los jugadores, las estrategias empleadas y las recompensas ofrecidas. En dicha matriz cada jugador puede escoger entre fila o columna, las estrategias se especifican según el número de filas o columnas, mientras que las recompensas son internas.

  Donde: El primero número es la recompensa recibida por el jugador de las filas, el segundo número es la recompensa del jugador de las columnas, así si el jugador 1 elige arriba y el *jugador 2* elige izquierda entonces sus recompensas son 4 y 3, respectivamente como se muestra en la Tabla3[3].

	El jugador 2 elige izquierda	**El jugador 2 elige derecha**
El jugador 1 elige arriba	4, 3	-1, -1
El jugador 1 elige abajo	0, 0	3, 4

Tabla3. Juego en Forma Normal.

[3] Tomada de: Binmore, K. Teoría de Juegos, McGraw-Hill, 1994.

En un juego que tenga esta forma, los jugadores actúan de manera simultánea sin saber la elección que tome un jugador de otro.

- **Forma Extensiva del Juego:** Se da cuando los jugadores tienen información de las elecciones de otros jugadores en juego. Esta forma se aplica cuando los juegos tienen que seguir un orden; se representan en forma de árbol donde cada nodo de éste corresponde a un punto de decisión para el jugador. El jugador se especifica por un número situado junto al vértice. Las líneas que parten del vértice representan acciones posibles para el jugador y las recompensas se especifican en las terminaciones de las ramas del árbol. En el gráfico1 se observa que existen dos jugadores. El *jugador 1* mueve primero y elige *F* o *U*. El *jugador 2* ve el movimiento del *jugador 1* y elige *A* o *R*. Si el *jugador 1* elige *U* y entonces el *jugador 2* elige *A*, entonces el *jugador 1* obtiene 8 y el *jugador 2* obtiene 2 como se observa en la figura 6.

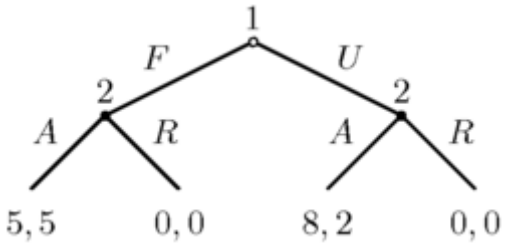


Figura 6. Representación del Juego en Forma Extensiva

- **Juegos en Forma Gráfica:** Este consiste en un conjunto de jugadores, escenarios y movimientos que cada uno puede realizar para cambiar de escenario, donde el objetivo del juego consiste en aumentar las recompensas que recibe el jugador.

- **Juegos Cooperativos:** Involucra contratos que se deben cumplir, donde cada jugador negocia el monto de su inversión,

además se permite hacer ofertas y considerar la mejor. Este juego aplica el equilibrio de Nash.

- **Juegos Simétricos:** En estos juegos las recompensas dependen únicamente de las estrategias que tome el otro jugador más no quién las juegue. La identidad de cada jugador puede cambiar sin que cambie la recompensa de cada estrategia.

- **Juegos Asimétricos:** En este tipo de juegos no existe un conjunto de estrategias general para todos los jugadores, puesto que cada jugador tiene sus propias reglas. Ejemplos de estos juegos son: ultimátum y el juego del dictador.

- **Juegos de Suma Cero y de Suma No Cero:** Los juegos de suma cero benefician a todos los jugadores, en cada combinación de estrategias la suma es siempre cero, ejemplos de este tipo de juego son: ajedrez, póker, ya que la cantidad de ganancia de un jugador es igual a la pérdida del otro. Los juegos de suma no cero se dan cuando la ganancia de un jugador no es la misma que la pérdida del otro. Ejemplo de este tipo es el dilema del prisionero.

- **Juegos Simultáneos y Secuenciales:** En los juegos simultáneos los jugadores desconocen los movimientos anteriores que realizó su adversario, este tipo de juegos es representado mediante la forma normal de un juego. En los juegos secuenciales el jugador tiene conocimiento de los movimientos anteriores realizados por su oponente. Este tipo de juego se representa por la forma extensiva de un juego.

- **Juegos de Información Perfecta:** Este tipo de juego está dado si todos los jugadores conocen los movimientos que han efectuado previamente todos los otros jugadores; sólo los juegos secuenciales pueden ser juegos de información perfecta, pues en los juegos simultáneos no todos los jugadores conocen las acciones del resto. También existen los juegos de información imperfecta en la cual se representa la ignorancia de información por parte de un jugador como se observa en la figura 7.

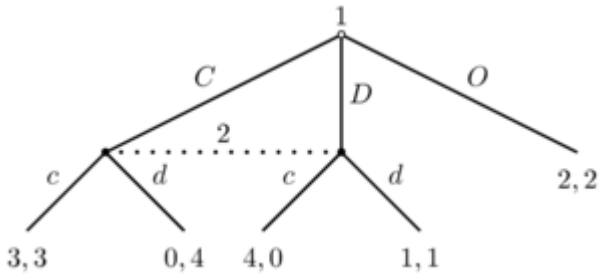


Figura 7. Representación del Juego de Información Imperfecta en donde el jugador 2 no carece de información.

- **Juegos de Información completa:** Este tipo de juego requiere que cada jugador conozca las estrategias y recompensas del resto pero no necesariamente las acciones. Un ejemplo de este juego es el ajedrez y el dilema del prisionero. Este tipo de juegos ocurren rara vez en el mundo real.

- **Juegos de Longitud Infinita (SuperJuegos):** En estos juegos no existe un número finito de movimientos, puesto que no tienen restricciones para los jugadores, y el ganador se conoce cuando todos los movimientos se hayan conocido también.

La teoría de juegos es aplicable a muchas ramas debido a su versatilidad en la resolución de problemas entre las cuales se encuentran: Economía, Matemática, Sociología, Política, Biología, Jurisprudencia. También existen ejemplos muy

conocidos como la fea y el dilema del prisionero que ayudan a comprender la naturaleza de la cooperación humana, entre otras.

Función de Utilidad

Frente a un juego, o ante un problema de decisión, se tiene un conjunto de posibilidades de actuación, cada una de las cuales presenta un resultado. Estos resultados pueden ser de dos tipos: determinista y no determinista. En el primer caso, si es determinista, se tiene un ambiente de incertidumbre y si el resultado es no determinista el ambiente será de riesgo.

Para unificar ambos ambientes se utiliza la función de utilidad, en la cual se tiene un conjunto $A=\{A_1, A_2,..., A_n\}$ de posibles estrategias o actuaciones que desembocan en resultados. La relación entre los posibles resultados, se denomina relación de preferencia-indiferencia. El jugador debe decidir entre dos resultados A y B siendo indiferente su elección. Con esto se tiene que la función de utilidad es una aplicación definida en el conjunto de posibles resultados A, con valores reales, tal que respeta la relación de preferencia entre los resultados, esto es: si el resultado A es preferido al B, entonces la utilidad U(A) es mayor que la utilidad U(B).

En ambientes probabilísticos, una actuación suele desembocar en posibles resultados: eligiendo la estrategia, X se tiene una probabilidad p_1 de obtener el resultado A_1, una probabilidad p_2 de obtener el resultado A_2... y una p_i de obtener el resultado A_i. En este caso, una vez definida la función de utilidad de los sucesos A_i, la valoración de utilidad de la adopción de la estrategia o elección X será: $p_1 \cdot U(A_1) + ... + p_i \cdot U(A_i)$, donde la suma de todas las p_i es la unidad.

De esta forma, el ambiente de certidumbre se considera como una variedad del ambiente en riesgo, para el cual ante una determinada actuación o estrategia, las probabilidades de todos los resultados posibles son cero excepto para uno de ellos, que vale uno, lo cual se conoce como distribución de probabilidad

degenerada. Con lo cual el ambiente de riesgo es general y éste engloba al de certidumbre como caso particular.

La teoría define la función de utilidad de la siguiente manera:

U = f (X1, X2, X3 , ... , Xn) (1.1)

Donde:

- "U" es el nivel de la utilidad
- "Xi" son los bienes y/o servicios que consume una determinada persona.

En la figura 8, se analiza la evolución de la utilidad a medida que aumenta el consumo del bien "X", donde el eje vertical es la utilidad total y el eje horizontal, las cantidades del bien "X".

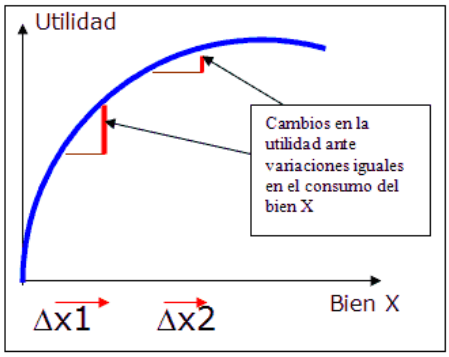


Figura 8. Teoría de la Utilidad.

2.4. Algoritmos y Fórmulas Matemáticas

Curva Plana: Es uno de los conceptos básicos e importantes de la geometría diferencial. Una curva plana es una función que tiene por dominio un segmento (intervalo) de recta de la forma [a, b] y su rango esta en $\mathbb{R}^2$. En símbolos se tiene:

$$F : [a,b] \rightarrow \mathbb{R}^2 \text{ o } F(t) = (x(t),y(t))$$

Por ejemplo, la función $f(x) = \sqrt{x}$, la cual se muestra en el gráfico 3, se transforma en una curva según la definición, si $F(t) = (t, \sqrt{t})$, con el dominio de F definido en $[0, \infty)$.

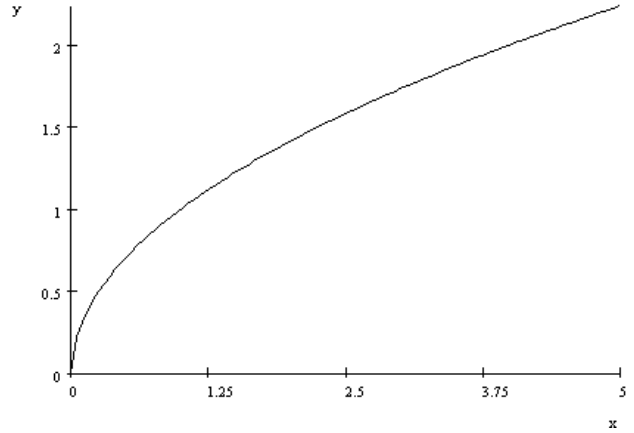


Gráfico 3. Curva de la función F(x).

La función $f(x) = 5x - 5$, se transforma en $F(t) = (t, 5t - 5)$, con dominio en $(-\infty, \infty)$, como se muestra en el gráfico 4.

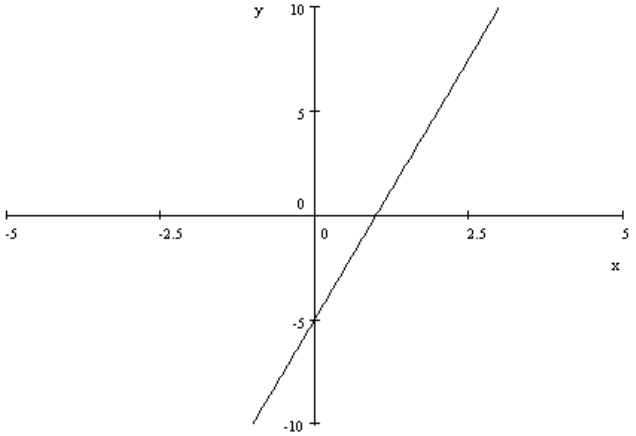


Gráfico 4. Función F(x) = 5x-5.

En el caso de ecuaciones que no son funciones también se puede llevar a cabo la parametrización, como por ejemplo en el caso de la circunferencia $x^2 + y^2 = 5$ del gráfico 5, su parametrización estaría dada por $F(t) = (5 * \cos t, 5 * \sin t)$, con $t \in [0, 2\pi]$.

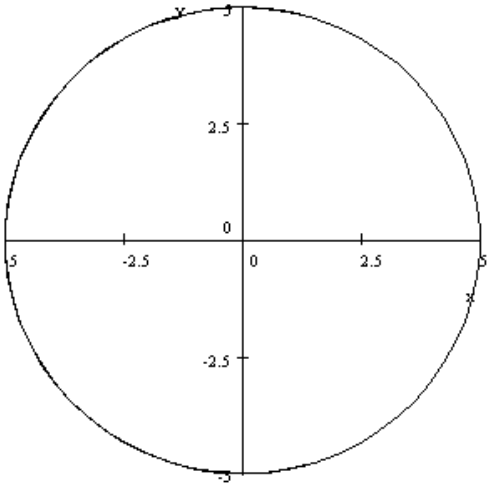


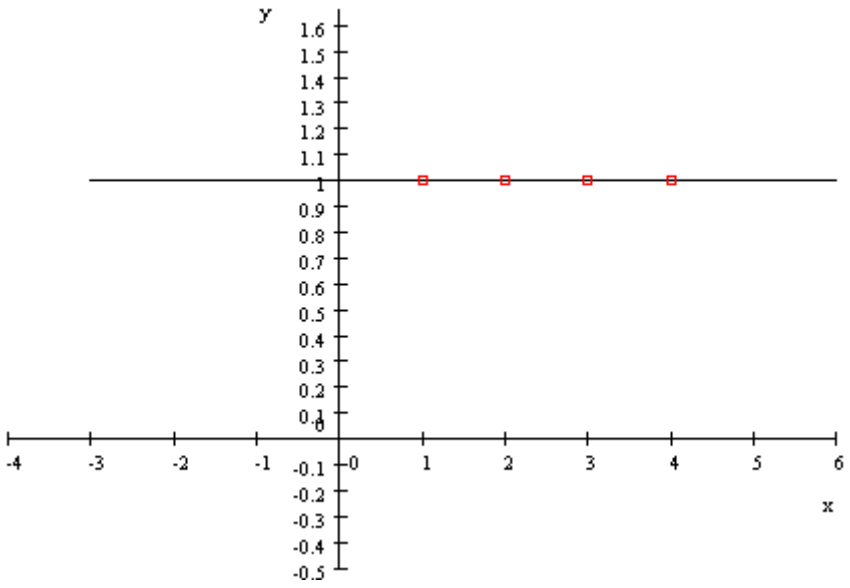
Gráfico 5. Circunferencia

Movimientos Básicos de un Punto

Las curvas se pueden analizar de manera muy sencilla y manipulaciones tales como la derivada y la integral tienen aplicaciones muy importantes, una de ellas es describir los movimientos de partículas físicas o puntos, al poder calcular su velocidad, aceleración y desplazamiento.

Desplazamiento Rectilíneo

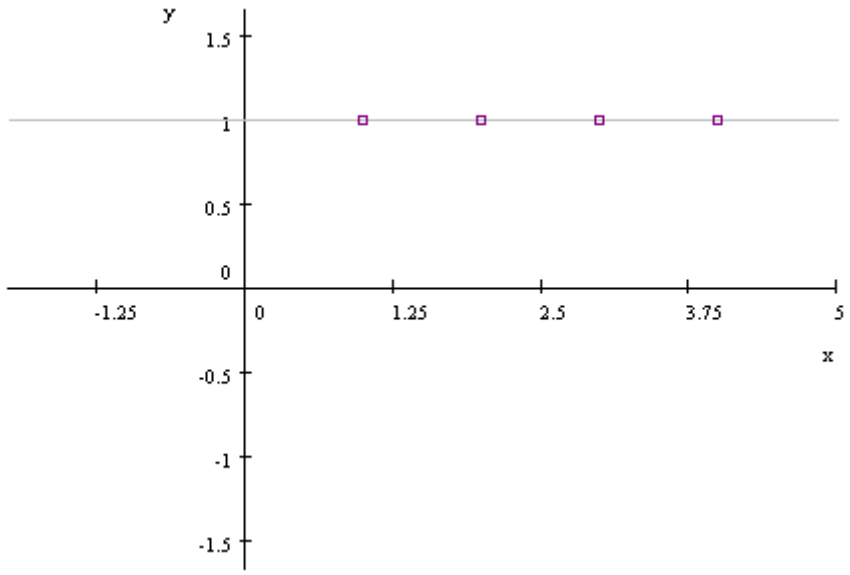
Este movimiento sucede sobre una recta. En el gráfico 6 se observa la función $F(t) = (t + 1, 1)$ donde se destacan los puntos $F(1) = \begin{bmatrix} 2 & 1 \end{bmatrix}, F(2) = \begin{bmatrix} 3 & 1 \end{bmatrix}$ $F(3) = \begin{bmatrix} 4 & 1 \end{bmatrix}$ y $F(4) = \begin{bmatrix} 5 & 1 \end{bmatrix}$, los cuales indican que el desplazamiento es hacia la derecha.



$$F(t) = (t + 1, 1)$$

Gráfico 6. Desplazamiento hacia la derecha.

Si se define una nueva función como $F^*(t) = (-t + 4, 1)$. Se puede observar en el gráfico 7 que al evaluar los mismos puntos, el desplazamiento es a la izquierda.



$$F^*(t) = (-t + 4, 1)$$

Gráfico 7. Desplazamiento hacia la izquierda.

Con estos dos ejemplos se concluye que:

- La parametrizacion de una curva puede tener 2 o mas expresiones diferentes.
- Si la parametrizacion cambia por un signo entonces esta solo cambia en el sentido en el que se recorre.

Estos cambios se dan aplicando el concepto de derivada, que permite observar el cambio de velocidad en un punto dado, y el concepto de rapidez, que es la magnitud o norma de la derivada (velocidad)

La derivada de una función $F(t) = (x(t), y(t))$ con respecto a "t" esta dada por $\frac{dF(t)}{dt} = (\frac{dx(t)}{dt}, \frac{dy(t)}{t})$.

Interpolación

Si se tiene n puntos en el plano de la forma (x_i, y_i) con $i = 1 \ldots n$, en Evolution los puntos están determinados por el contorno una figura, entonces el proceso de construir una función que cumpla con: $f(x_i) = y_i \forall i$ con $i = 1 \ldots n$ se define como Interpolación. Dicha función es denominada Interpolante.

Al conjunto de las abscisas $\{x_i \ / \ i = 1 \ldots n\}$ se le llama soporte o nodos de Interpolación.

Interpolación Lineal: Consiste en unir cada uno de los puntos del conjunto dado con una recta, la misma que está determinada por los pares de puntos consecutivos, los cuales se simbolizan por:

$$\begin{array}{cc} Soporte & Y \\ x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{array}$$

Con cada uno de los cuales se forman pares de puntos consecutivos, que son:

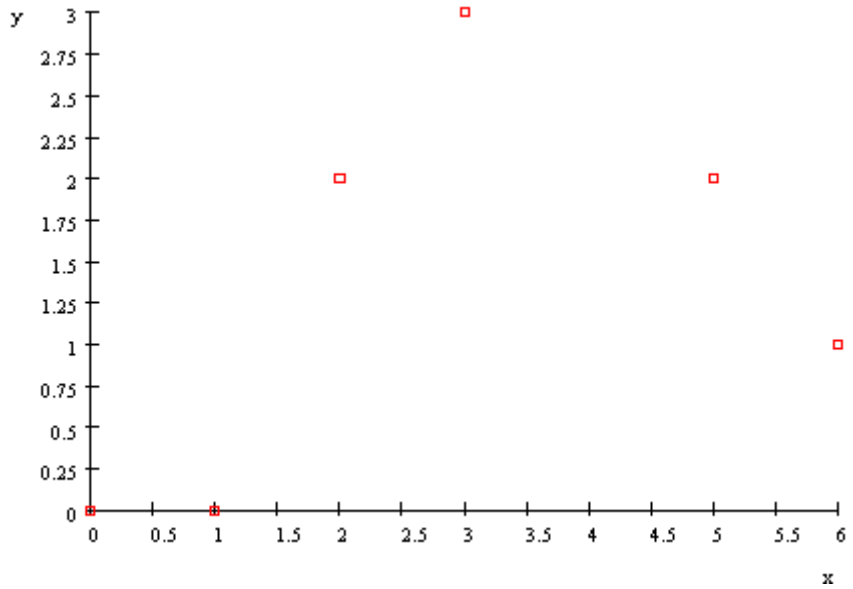
$$[(x_1, y_1), (x_2, y_2)], [(x_2, y_2), (x_3, y_3)], \cdots [(x_{n-1}, y_{n-1}), (x_n, y_n)]$$

La recta generada por cada uno de estos puntos es: $y = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}(x - x_i) + y_i$ para $i = 1 \ldots n$. Por lo tanto el Interpolante estará dado por:

$$f(x) = \begin{cases} \frac{y_2-y_1}{x_2-x_1}(x - x_1) + y_1 & \text{en } [x_1, x_2] \\ \frac{y_3-y_2}{x_3-x_2}(x - x_2) + y_2 & \text{en } [x_2, x_3] \\ \vdots & \vdots \\ \frac{y_n-y_{n-1}}{x_n-x_{n-1}}(x - x_{n-1}) + y_{n-1} & \text{en } [x_{n-1}, x_n] \end{cases}$$

Este método es muy rápido, pero tiene algunos inconvenientes, el más importantes de estos es que el interpolante que produce no es suave, es decir en los puntos $Soporte - \{x_1, x_n\}$ la función no tienen derivada, lo que indica que tiene muchos picos; esto produce funciones difíciles de analizar, y en el caso de los videojuegos, personajes muy cuadrados.

Un ejemplo, se da usando los puntos $(0,0),(1,0),(2,2),(3,3),(5,2),(6,1)$, se tiene la gráfica 8:

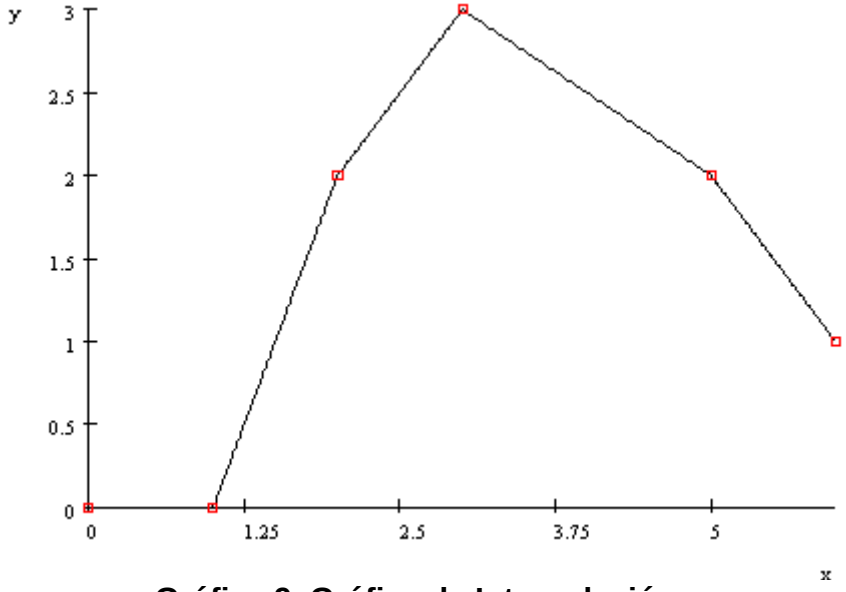


Gráfica 8. Gráfica de los puntos interpolados.

Donde el Interpolante es:

$$f(x) = \begin{cases} 0 & if \ 0 \le x < 1 \\ \frac{2}{1}(x-1) & if \ 1 \le x < 2 \\ (x-2)+2 & if \ 2 \le x < 3 \\ \frac{-1}{2}(x-3)+3 & if \ 3 \le x < 5 \\ -(x-5)+2 & if \ 5 \le x \le 6 \end{cases}$$

La grafica 9 que se presenta a continuación es la grafica del interpolante, conjuntamente con los puntos interpolados.



Gráfica 9. Gráfica de Interpolación.

2.5. Análisis de Herramientas de Diseño, Modelado y Animación 3D

Herramientas Graficas

El uso de herramientas gráficas ha revolucionado el mundo en cuanto a las posibilidades o enfoques que se les puede dar. El uso principal que se les ha dado a herramientas de esta categoría ha sido enfocado principalmente a aplicaciones interactivas o videojuegos, pero mas allá de esta aplicación; hoy en

día, estas herramientas han facilitado otras labores cotidianas que hace algún tiempo atrás se las consideraba muy difíciles o tediosas. Tal es el caso de la aplicación en la medicina, en donde doctores disponen de software el cual muestra el cuerpo humano en 3D para mayor acercamiento a la realidad. Como resultado se puede apreciar que las herramientas, principalmente 3D han aportado en gran medida a algunas áreas que necesitan de ayuda para generar prototipos como es el caso de este proyecto.

En la siguiente figura (figura9), se observa un ejemplo de diseño realizado con una aplicación 3D.

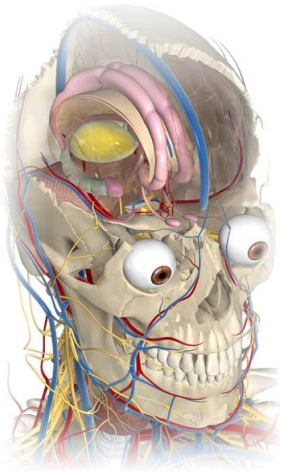


Figura 9. Aplicación 3D (Visible Body 3D)

En cuanto a Herramientas Graficas a ser involucradas dentro del proyecto se tienen tres áreas. Primero, las herramientas 2D las cuales son de gran utilidad para la creación de interfaces, edición de texturas, bocetos, etc. Segundo, se tienen las herramientas 3D para el modelado de los diseños conceptuales para su posterior animación. Por último, las herramientas de animación las cuales ayudarán a dar vida a los personajes para su posterior implementación en la aplicación. En la figura 10 se muestra el diseño de los personajes en 3D.

Figura 10. Diseño y modelado de habitantes de Evolution

Herramientas de Edición 2D

Estas herramientas van dedicadas a la preproducción del proyecto, donde el equipo trata de dar una noción de lo que va a ser el desarrollo del videojuego en cuanto a la historia y lógica del mismo, con lo cual el equipo de trabajo tiene una visión general del proyecto gracias a las ideas aportadas que son reflejadas en el Diseño conceptual del Juego. En la figura 11 se muestra un boceto que va a ser posteriormente diseñado utilizando herramientas en 3D.

Figura 11. Diseño Conceptual (T-Rex)

Una vez que las ideas del equipo son expuestas, estas se plasman en imágenes para mayor entendimiento, en este caso se presenta la figura de un Dinosaurio el cual será parte del juego. El Dinosaurio es diseñado como una imagen para

poder entender algunos aspectos como el comportamiento o la orientación que se le va a dar dentro del juego. El dibujo no muestra la versión final del personaje solo una idea general, ya que en el caso de este Juego el dinosaurio tendrá un enfoque mas cómico en base al juego resultante.

Las herramientas 2D brindan una perspectiva general de lo que se busca conseguir en base a bocetos e imágenes. El proceso también incluye otras técnicas muy conocidas y útiles como el uso de Story Boards (figura 12) que permiten visualizar una secuencia de imágenes sobre algún aspecto clave que los desarrolladores necesitan saber para poder elaborar la aplicación. Tal es el funcionamiento de la interfaz grafica, o el manejo de menús o el comportamiento de un determinado caracter dentro de la aplicación.

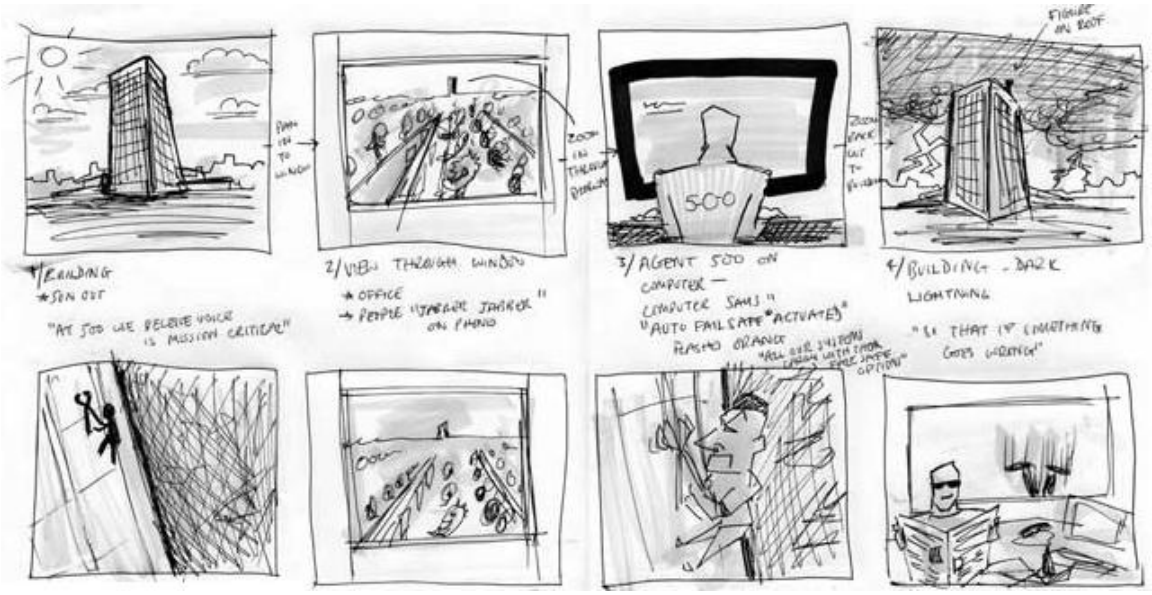


Figura 12. (Story Board)

La aplicación de herramientas 2D es bastante amplia. Otro de sus usos dentro del desarrollo de Juegos, es la creación de texturas. Este es un proceso muy importante a la hora de crear un videojuego ya que de este proceso depende la apariencia del juego. Dependiendo del enfoque, el juego puede tener un acercamiento a la realidad mediante el uso de texturas foto realista o imágenes

que son parte de la realidad. Por otro lado se le puede dar un enfoque cómico con texturas que contienen pocos detalles o utilizan colores que no coinciden con la realidad pero que son suficientemente claras como para representar el concepto de la aplicación como se muestra en la figura 13.

Figura 13. Aplicación de Texturas en imágenes 3D.

En la actualidad existe una gran variedad de programas de edición 2D los cuales van orientados a diversas necesidades como en el caso de desarrollo de videojuegos, en donde por facilidad y conveniencia se tiene 2 programas muy fundamentales.

- Macromedia Fireworks
- PhotoShop

PhotoShop, es un programa especializado en la edición de imágenes a nivel profesional, el cual es muy usado en aplicaciones 3D gracias a su precisión y exactitud. En la siguiente figura (figura 14) se muestra un ejemplo de la utilización de PhotoShop.

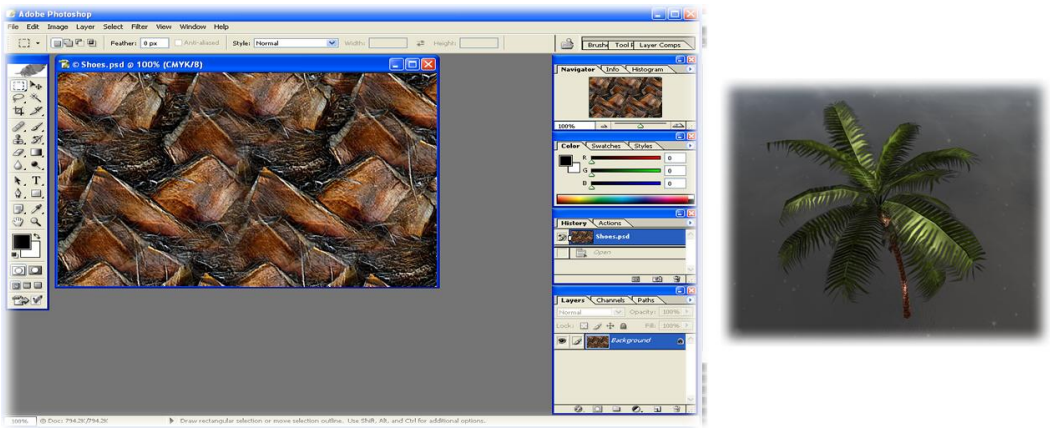


Figura 14. Herramienta de Diseño PhotoShop.

Fireworks es un programa que de la misma forma esta orientado a la edición de imágenes a un nivel intermedio el cual tiene la mayoría de funciones que se necesitan para esta aplicación en especial. En la siguiente figura (figura 15) se muestra un ejemplo de la utilización de Fireworks.

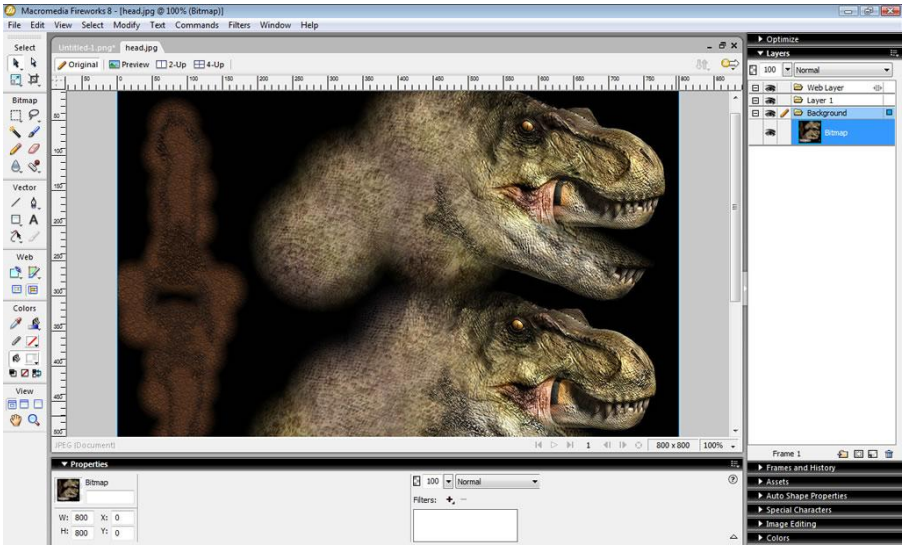


Figura 15. Herramienta de Diseño Fireworks.

Las principales aplicaciones de las herramientas dentro de la aplicación serán las siguientes:

- Diseño conceptual
- Story Boards
- Texturas

- Bump Maps
- Interfaces
- Menús

Las siguientes figuras (figura 16 y 17) muestran la textura realizada con herramientas de diseño y el modelo final en 3D.

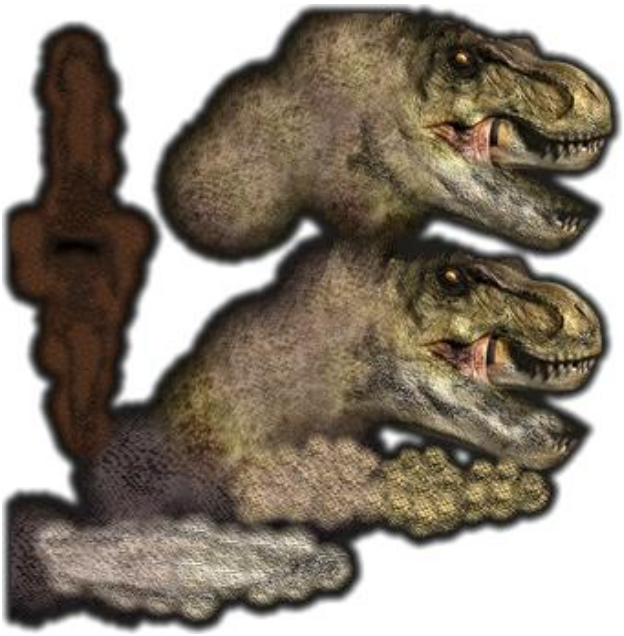


Figura 16. (Texture)

Figura 17. (3D Model)

Las herramientas 2D son útiles tanto en el proceso de preproducción como en el proceso de producción, es decir durante todos los ciclos de desarrollo.
Para ser un diseñador eficiente se necesitan herramientas para desarrollar diseños con rapidez y que permitan comunicar con total fidelidad las ideas, emociones y datos a todas las personas involucradas en la investigación, el desarrollo, el marketing y la fabricación del producto, además del proceso de construcción.

Herramientas de Modelado y Animación 3D

Estas herramientas son fundamentales en el desarrollo de una aplicación 3D, ya que son capaces de trasladar una idea previamente plasmada en papel para

luego ser llevada a la aplicación resultante. Es necesario tener la idea o el modelo conceptual, como se muestra en la figura 18, para trasladarlo a la tercera dimensión, para lo cual es necesario disponer de varios bosquejos que muestren tanto el alto, ancho y profundidad del objeto a modelar. Otros métodos comúnmente usados son el manejo de modelos reales, maquetas o esculturas las cuales brindan una mejor perspectiva del objeto al diseñador.

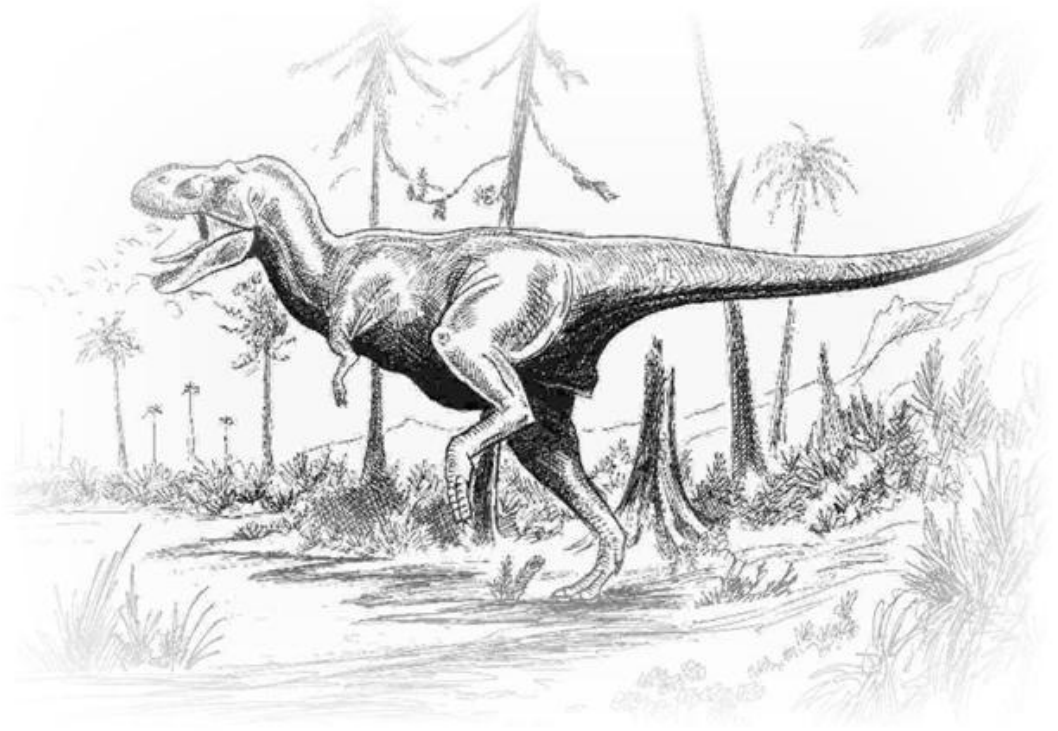


Figura 18. Boceto Dinosaurio.

Existes varias herramientas de modelado 3D las cuales tienen ciertas ventajas o desventajas para el desarrollo de videojuegos. Las Herramientas más conocidas en el mercado son:

- Autodesk 3D Studio Max
- Autodesk Maya
- XSI
- Blender

Los cuatro productos hacen prácticamente lo mismo pero de modo distinto.

Blender

Blender, es una herramienta en desarrollo, su ventaja más notoria es el hecho de ser de libre distribución, es decir que no es necesario comprar una licencia para poder crear aplicaciones con fines comerciales. Algunas deficiencias son la poca compatibilidad con los motores gráficos, escasa documentación y menos características que otros programas 3D.

Blender es open source y lo mantiene una comunidad muy leal que a logrado hacer que un proyecto considerado perdido se levante a niveles muy buenos.
La interfaz de Blender tiene una gran cantidad de elementos que lo hacen ver complejo, además incorpora un motor de 3D en tiempo real el cual permite la creación de contenido tridimensional interactivo que puede ser reproducido de forma independiente.

Softimage

Softimage ha estado en el mercado por mucho tiempo y ha cambiado de dueño muchas veces, incluso fue de Microsoft por un tiempo, pero, su versión actual el XSI fue hecha prácticamente de cero así que se puede decir que es uno de los más nuevos. También está más orientado a personajes y su última versión tiene herramientas especializadas en animación facial. Viene también con Mentalray, de hecho, fue el primero en incluirlo. No hay una preferencia por utilizarlo en alguna función en especial.
XSI, es una nueva herramienta de modelado 3D creada en su mayor parte para la creación de juegos. Sus ventajas son la buena adaptabilidad a herramientas para la elaboración de juegos y su buen manejo con modelos de pocos vértices. Las desventajas radican principalmente en el precio y falta de documentación ya que es una herramienta nueva para el mercado.

Maya

Maya, es una herramienta muy conocida en lo que se refiere a modelado 3D, sus aplicaciones son extensas, va desde la creación de videojuegos a la

creación de películas debido a su precisión. Sus ventajas son la disponibilidad de documentación, buena interfase y creación de modelos detallados. La desventaja principal es la complejidad de su manejo ya que está orientado a un nivel profesional en detalle y precisión.

Maya se caracteriza por enfocarse en el diseño de personajes principalmente, trae muchas más herramientas de modelado y texturizado y sus motores de simulación son mucho más elaborados que los de 3D Max.
Maya es la herramienta preferida por los estudios de animación y efectos para películas por su lenguaje de scripting, siendo éste muy sencillo de modificar y acondicionar a las necesidades de cada estudio.

3D MAX

3D Max, es la herramienta más popular y conocida para muchos aficionados al diseño 3D, sus aplicaciones son muy extensas, y al igual que Maya son las herramientas más usadas en todo lo relacionado a arquitectura 3D. Sus ventajas son la excelente operabilidad, facilidad de uso, extensa documentación en el mercado, así como la integración con animación, programación, y facilidad de acoplamiento con herramientas externas, tal es el caso de XNA. Su desventaja es el costo si se lo relaciona con programas que poseen similares características, pero es inferior si se relaciona con las herramientas previamente expuestas.

3D Max, es el mas "general" de los cuatro, viene con herramientas muy buenas para modelado, animación, efectos y simulaciones (tela, cabello, partículas, etc.), un motor de render muy bueno, la posibilidad de agregarle plugins de terceras compañías y permite lograr diseños utilizando pocos polígonos lo cual resulta una gran ventaja ya que para el desarrollo de videojuegos a mayor polígonos, menor rendimiento.

Direct X SDK

DirectX SDK (DirectX Software Development Kit) es un conjunto de herramientas de desarrollo, necesario para un juego y para crear aplicaciones basadas en DirectX en VisualBasic.NET, C / C + +, C #.

Algunos de los principales componentes incluidos en el DirectX SDK son los siguientes:

- DirectX cabeceras y bibliotecas.
- DirectX componentes del sistema (runtimes).
- DirectX API (Application Programming Interface), documentación

DirectX SDK permite crear excelentes juegos y aplicaciones gráficas debido a que cuenta con las últimas innovaciones en hardware para el desarrollo de juegos.

Una de las características más importantes incluidas en el SDK de DirectX es la librería D3DX, que es un conjunto de instrumentos que brinda a los desarrolladores un alto nivel de funcionalidad para crear aplicaciones, también incorpora aspectos básicos tales como, las matemáticas y las rutinas de carga de texturas, y la actualización de tecnología para la manipulación de los contenidos. Además posee una interfaz gráfica sencilla y comprensible, para todo aquel usuario que posea los conocimientos técnicos y teóricos necesarios para manipular aplicaciones que desarrollan lenguaje de programación orientado a objetos. La siguiente figura (figura 19) muestra la vista de un diseño 3D utilizando Direct X.

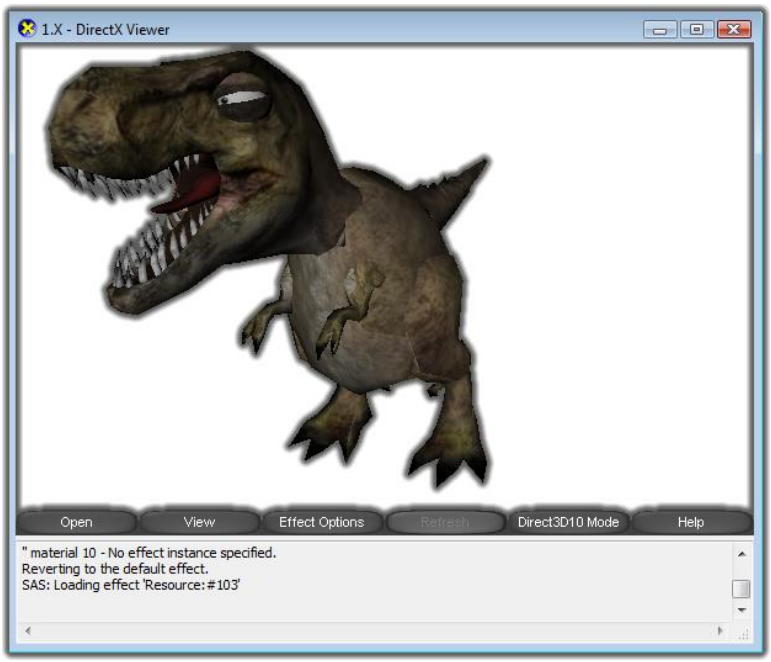


Figura 19. (DirectX Viewer)

2.6. Estudio de plataformas y herramientas de Desarrollo

Hoy en día existen muchos lenguajes de programación diseñados especialmente para crear videojuegos, sin embargo las herramientas más comunes utilizadas para desarrollar videojuegos en su mayoría son las mismas que se utilizan al desarrollar aplicaciones comunes. A continuación se presenta un listado de las aplicaciones de software más utilizadas para el desarrollo de juegos:

XNA:

Es una herramienta y un lenguaje de programación que permiten generar software conectado a .NET para Microsoft Windows, Web y una amplia gama de servicios. Debido a su sintaxis familiar, similar a la de C++, a su entorno de desarrollo integrado de gran flexibilidad y a su capacidad para crear soluciones para una gran variedad de plataformas y dispositivos.

XNA está basado en C# .NET lo cual hace que su entorno sea familiar a programadores que manejan Visual Studio.NET.

Xna tiene además las siguientes características:

- Independencia de la plataforma al crear videojuegos sin pensar en el dispositivo concreto sobre el cual se montará el juego.
- Los gráficos, sonidos y movimientos generados tienen un elevado grado de realismo.
- La herramienta de desarrollo de esta plataforma es Visual Studio, por lo que la adaptación de los programadores será más rápida.

Entre las herramientas incluidas en XNA se encuentran DirectX, las API de XAudio y XACT para sonido, HLSL (High-Level Shader Language), y PIX.

En la figura 20 se muestra un ejemplo de un juego realizado con XNA.

Figura 20. Videojuego Assault Heroes creado utilizando XNA 2.0

Leguaje C:

C es un lenguaje de programación de nivel medio ya que combina los elementos del lenguaje de alto nivel con la funcionalidad del ensamblador.

Su característica principal es ser portable, es decir, es posible adaptar los programas escritos para un tipo de computadora en otra.

Otra de sus características principales es el ser estructurado, es decir, el programa se divide en módulos (funciones) independientes entre sí.

El lenguaje C inicialmente fue creado para la programación de sistemas operativos, intérpretes, editores, ensambladores, compiladores y administradores de bases de datos. Actualmente, debido a sus características, puede ser utilizado para todo tipo de programas.

Un ejemplo de un juego realizado en lenguaje C se observa en la figura 21.

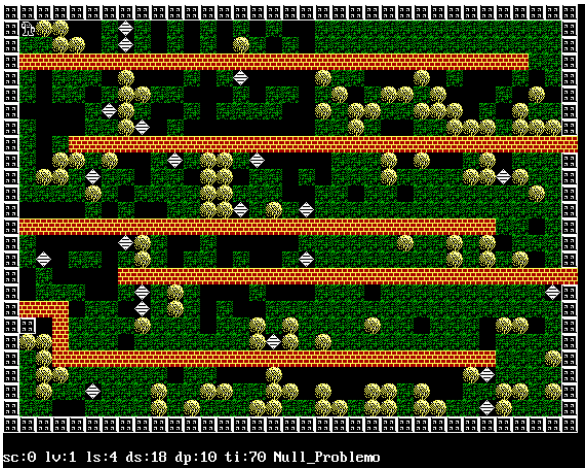


Figura 21. Videojuego creado utilizando C.

Lenguaje C++:

El lenguaje C++ se comenzó a desarrollar en 1980. Su autor fue B. Stroustrup, también de la ATT. Al comienzo era una extensión del lenguaje C que fue denominada C with classes.

Este nuevo lenguaje comenzó a ser utilizado fuera de la ATT en 1983.

El nombre C++ hace referencia al carácter del operador incremento de C (++). Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, la ATT comenzó a estandarizarlo internamente en 1987. En 1989 se formó un comité ANSI (seguido algún tiempo después por un comité ISO) para estandarizarlo a nivel americano e internacional.

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones y de videojuegos. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original.

La evolución de C++ ha continuado con la aparición de Java, un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras, que se utiliza para realizar aplicaciones en Internet.

Un ejemplo de un juego realizado en C++ se observa en la figura 22.

Figura 22. Videojuego creado utilizando C++.

Ogre: Ogre3D (Object-Oriented Graphics Rendering Engine) es un motor de videjuegos 3D escrito en C++ que permite soportar GLSL shader, bezier y un gran nivel de detalle. Esta herramienta funciona únicamente como una librería 3D, que se utiliza especialmente para el diseño de gráficos por lo cual trabaja en conjunto con otras librerías que proporcionan sonido y adaptaciones físicas. Entre sus características se puede destacar el manejo de shaders, LOD, sistema de partículas, HDR, etc. Es compatible con varios sistemas de edición en 3D como Blender o 3DSMax. Esta herramienta de software libre está disponible para los sistemas operativos Windows, Mac OS y como no Linux, dispone de una gran cantidad de herramientas y SDKs precompilados.

Su principal ventaja radica en la facilidad de uso y la amplia documentación existente para el aprendizaje de esta herramienta.

Un ejemplo de un juego realizado en Ogre se observa en la figura 23.

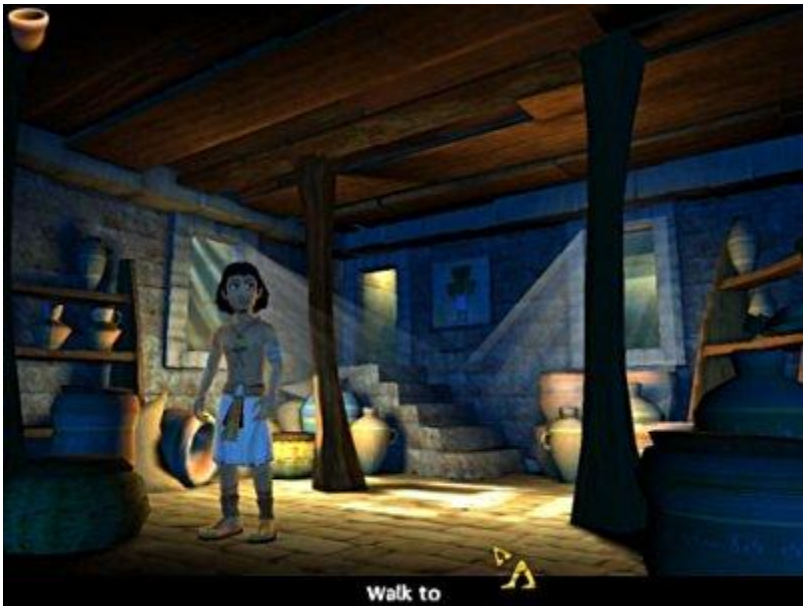


Figura 23. Videojuego creado utilizando Ogre que implementa HDR para su iluminación.

Nebula: Nebula es un potente motor de videojuegos open source en 3D que utiliza C++. Entre sus principales características están el poseer un motor de renderizado DX9, API unificada y una capa de abstracción. Esta herramienta permite el desarrollo simple en pocas líneas de código. Similar a esta

herramienta se encuentra también: DevIL, Tcl, Lua, Python, TinyXML, D3DX. Al igual que Ogre se puede usar para multiples plataformas: Win, Linux, Irix, Mac, Xbox.

Un ejemplo de un juego realizado en Nebula se observa en la figura 24.

Figura 24. Videojuego creado utilizando Nebula.

Crystal Space:

Está diseñado en C++ y su función principal es la de desarrollar ambientes 3D para cualquier tipo de juego, el cual incluye numerosas herramientas especialmente preparadas para ello.

Aparte del desarrollo 3D Cristal Space es un potentísimo motor gráfico enfocado al desarrollo de juegos y aplicaciones 3D que soporta renderizado Direct3D, OpenGL y por software.

Dentro de los posibles diseños que se puede crear con esta herramienta son superficies curvas, niebla volumétrica, luces de colores dinámicas, motor de terreno, LOD, proceso de texturas, sistema de partículas, físicas con ODE, entre otras. Las posibilidades son numerosas y los resultados increíbles, siendo su mayor ventaja la disponibilidad gratuita de esta herramienta que es uno de los más completos engines. Su inconveniente más grande es que tiene poca

documentación en forma de tutoriales por lo que tiene una curva alta de aprendizaje.

Un ejemplo de un juego realizado utilizado Crystal Space se observa en la figura 25.

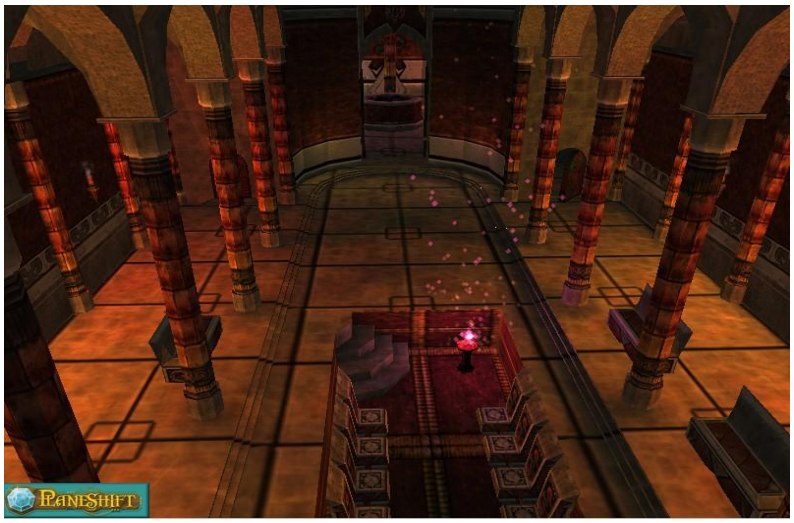


Figura 25. Videojuego creado utilizando CrystalSpace.

Darkbasic / Pro:

DarkBASIC PRO es una herramienta con la cual se puede crear todo tipo de videojuegos, desde los clásicos shooters en primera persona, a puzzles, aventuras gráficas, rol, estrategia, simuladores, simulación, coches, plataformas, etc.

DarkBASIC Profesional ofrece un compilador completamente original que lo convierte en una de las herramientas más avanzadas. Las principales características son:

- El código es compilado en ejecutables 100% código máquina.
- El depurador ofrece puntos de ruptura, modo paso a paso y observador de variables.
- Produce programas independientes que usan combinaciones de archivos externos e internos.
- Encripta todos los gráficos y archivos internos creando ejecutables más pequeños y seguros

- Los programas almacenan DLLs internamente para una distribución más sencilla de los juegos.
- Posee técnicas avanzadas de renderizado que ayudan a conseguir increíbles efectos gráficos con total facilidad.
- Permite añadir mayor realismo y belleza a los videojuegos utilizando los potentes shaders que esta herramienta ofrece, permitiendo dotar a los videojuegos con texturas de auténtico relieve o increíbles efectos de agua ultrarrealista.

Un ejemplo de un juego realizado en DarkBASIC Profesional se observa en la figura 26.

Figura 26. DOOM Videojuego creado utilizando DarkBASIC Profesional.[4]

Blitz Basic:

Blitz Basic es una poderosa herramienta de creación de videojuegos tanto en 2d como en 3D, que aplica conceptos orientados a objetos y permite la implementación de gráficos OpenGL. Con esta herramienta se puede crear propios terrenos irregulares a través de mapas de imágenes. Soporta mallas estáticas y animadas de extensión .3DS, .X, .MD2 y su nativa .B3D. Además puede manejar multicámaras, además de que puede reproducir archivos de sonido del tipo RAW, WAV,MP3 y OGG. Reproduce archivos de video .AVI o .MPEG

Blitz Basic es un compilador para el lenguaje de programación Basic diseñado para ejecutarse en múltiples sistemas operativos.

[4] Tomado de. http://www.darkbasic.es/website/descubre-darkbasic.

BlitzMax es la primera versión modular del lenguaje basic, lo que permite escribir plugins para el propio lenguaje con lo que se puede realizar configuraciones que aumentan la productividad de la herramienta.
Un ejemplo de un juego realizado en Blitz Basic se observa en la figura 27.

Figura 27. Videojuego creado utilizando Grey Alien BlitzMax Game Framework[5].

2.7. Metodología Aplicada

Es de vital importancia aplicar una buena metodología a la hora de realizar un proyecto debido a que esta define las pautas necesarias para construir un software de calidad en el plazo establecido y con todos los objetivos propuestos. Para suplir estas necesidades se crearon las Metodologías Ágiles, las cuales permiten disminuir el tiempo de desarrollo sin que esto afecte a la calidad de la aplicación.

Scrum forma parte del conjunto de Metodologías ágiles permitiendo definir un marco para la gestión de proyectos. Esta metodología creada por Hirotaka Takeuchi e Ikujiro Nonaka en el año 1986 en un estudio que documentaba una

[5] Tomado de: http://es.wikipedia.org/wiki/Blitz_BASIC.

serie de proyectos muy exitosos los cuales tenían en común el uso equipos chicos y multidisciplinarios. Posteriormente Jeff Sutherland creó el proceso Scrum para el desarrollo de software en 1993 usando este estudio como base. Para el año 1995 Ken Schwaber formalizó el proceso y lo abrió a toda la industria del software.

La construcción de un videojuego conlleva un ciclo de desarrollo diferente a la construcción de cualquier otro proyecto informático, debido a que las fases aplicadas en las metodologías de desarrollo de software habitual no son las mismas que se aplican a proyectos de este tipo porque contemplan fases diferentes como preproducción, producción y postproducción que deben ser revisadas en periodos cortos de tiempo, permitiendo un mejor control del trabajo, además cada fase debe ser concretada mediante iteraciones que añadan funcionalidad en cortos incrementos y por ser proyectos altamente innovadores, se requiere controlar de mejor manera la gestión de calidad para evitar fallos.

Es por esta razón que se cree conveniente utilizar la metodología “Scrum” ya que abarca todas las expectativas planteadas por el equipo de trabajo, teniendo como objetivo principal el construir un software de calidad. De la misma forma Scrum es usado en su mayoría para el desarrollo de este tipo de aplicaciones en lo que se refiere a la rama de los Videojuegos por lo cual resulta indispensable su uso para el correcto desempeño, control y manejo del proyecto planteado.

Además esta metodología permite gestionar de manera efectiva el tiempo y los recursos que van a ser consumidos en el sistema, de la misma forma debido a que aplica iteraciones cortas, se tiene como ventajas que el trabajo sea controlable, autogestionable, adaptable a cambios posteriores, y que en cada una de ellas se tenga un alto grado de innovación. Por otro lado los miembros del equipo tienen las mismas responsabilidades ya que no cuentan con un líder por lo cual todos pueden aportar en la toma de decisiones.

Scrum requiere para su implementación tres elementos principales los cuales son:

- Personal Involucrado
- Sprints o iteraciones
- Documentación

Personal Involucrado

La división del personal se enfoca más al ámbito funcional ya que no se requiere asignar roles por jerarquías. Dicha división puede ser planteada de la siguiente forma:

- Product Owner
- Scrum Master
- Scrum Team
- Usuario Final o Cliente

Product Owner: Es el responsable de velar por los intereses de cada uno de los participantes dentro del proyecto. También tiene la responsabilidad de asegurarse que todos los requerimientos de los diferentes grupos de usuarios del sistema sean contemplados y que el producto de un beneficio para los intereses económicos y estratégicos del negocio.

Scrum Master: Es el responsable del proceso Scrum, quien debe enseñar la metodología a cada integrante implicado en el proyecto, además determina qué tareas se deben hacer, quién debe hacerlas, cuándo y durante cuánto tiempo y el costo que cada una de ellas conlleva, transformándose en un coordinador de actividades.

Scrum Team: Los equipos auto-suficientes, auto-organizados y funcionales, tienen la responsabilidad, en cada iteración, de transformar el Product Backlog en un incremento en la funcionabilidad del producto y planificar su propio trabajo para lograrlo. El scrum team son los responsables en conjunto del éxito de cada iteración y del proyecto en su totalidad.

Usuario Final o Cliente: Son los beneficiarios finales del producto, y son quienes observando los progresos, pueden aportar ideas, sugerencias o necesidades.

Sprints o Iteraciones

Para iniciar este proceso es necesario construir un Product Backlog, siendo este el punto de partida para el inicio de las iteraciones o Sprints.

Un sprint involucra el trabajo a realizarse en un plazo de 30 a 45 días y están estrictamente limitadas a los tiempos preestablecidos.

En cada sprint se definen las siguientes tareas:

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Sprint Planning Meeting: Es una reunión de planeamiento del Sprint donde se encuentran involucrados el Product Owner y el equipo. En esta fase de la metodología se toma en cuenta al Product Backlog que fue previamente realizado ya que de este se obtienen las prioridades más altas que van a ser implementadas

La duración de la junta de planeación del sprint debe ser suficiente para realizar el análisis, evaluación, repriorización y estimación de las tareas acumuladas del producto así como el desglose y estimación de las tareas de los requerimientos aceptados por el equipo.

Daily Scrum: Es una tarea que se realiza todos los días que dure el Sprint Backlog con el equipo de desarrollo. Se trata de una reunión operativa, informal y ágil, de un máximo de 30 minutos, en la que se le hace 3 preguntas a cada integrante del equipo:

- Qué tareas se ha realizado desde la última reunión.
- Sobre qué va a trabajar en el día actual.
- Identificación de que obstáculos o riesgos impiden o pueden impedir el normal avance.

Sprint Review: Se revisa el Sprint finalizado que a este punto, se debe tener un avance que el Cliente pueda apreciar. En esta reunión, suelen asistir el Product Owner, el Scrum Master, el Scrum Team y personas que podrían estar involucradas en el proyecto. El Scrum Team es quién muestra los avances realizados en el Sprint.

Sprint Retrospective: En este paso el Product Owner revisa con el equipo los objetivos marcados inicialmente en el Sprint Backlog, se aplican cambios y ajustes si son necesarios, y se marcan los aspectos positivos y los aspectos negativos del Sprint.

Documentación

Para gestionar el proyecto y para dar seguimiento al mismo la metodología propone elaborar 3 documentos fundamentales:

- Product Backlog
- Spring Backlog
- Burn Down Chart

Product Backlog: Que consiste en listar todas las tareas, funcionalidades o requerimientos ordenadas de mayor a menor de acuerdo a su importancia.

Spring Backlog: Es una lista de tareas que provienen del Product Backlog y que son completadas mientras dura el spring.

Burn Down Chart: Este gráfico permite conocer de manera ágil y visual el progreso o no de los trabajos del proyecto.

2.8. Análisis del Framework XNA

Introducción al Framework XNA

XNA fue desarrollado por Microsoft. Este framework fue lanzado por primera vez en el año 2004 en GDC (Game Developers Conference),

XNA no es solo un framework como DirectX ya que contiene además un conjunto de herramientas y un IDE derivado de Visual Studio para facilitar la programación de videojuegos.

XNA Framework está compuesto por dos librerías y aplicaciones construidas sobre el .Net Framework 2.0 de Microsoft que proveen a los lenguajes .Net un conjunto de herramientas con el objetivo de programar videojuegos, proporcionando una conexión entre el código manejado en .NET y la librería de multimedia de Microsoft DirectX.

Antes de lanzar XNA Game Studio Express, los desarrolladores de juegos usaban tecnologías como DirectX y OpenGL. Estas API permiten obtener acceso básico al hardware, pero no son intuitivas. Debido a ello, muchas veces los desarrolladores ensamblaban componentes básicos como gráficos, audio y datos de entrada en un marco, denominado motor y luego establecían una lógica específica del juego por niveles sobre este marco. Esta abstracción a menudo es costosa y ocupa mucho tiempo de creación y utilización.

Además, la administración de la memoria y la lógica en lenguajes como C++ es muy exigente técnicamente, ya que administra todas las piezas de contenido artístico presentes en un juego. Existen también complejos requisitos técnicos y empresariales para el desarrollo de consolas de videojuegos de última generación, como Xbox 360.

XNA Game Studio Express incluye el XNA Framework, una API de desarrollo de juegos para varias plataformas, tanto para Windows como para Xbox 360. XNA Framework simplifica no sólo los gráficos, el sonido, la entrada y el almacenamiento de datos, sino también los bucles de temporización y de dibujo utilizados en todos los juegos.

XNA Game Studio permite a los desarrolladores agregar su contenido artístico (modelos 3D, texturas gráficas, audio) en su solución del IDE, de forma que se integre en objetos de tiempo de ejecución a los que se puede obtener acceso con un código.

Además, el modelo de la aplicación, establece automáticamente el dispositivo de gráficos apropiado sin ningún código complicado de enumeración de dispositivo. Como el código ya está establecido, se puede compilar y ejecutar un proyecto de XNA Game Studio Express con un bucle de temporización y representación completamente funcional que se ejecutará desde el momento de su creación.

El modelo de programación en XNA es un modelo basado en componentes y contenedores. El juego, que está representado por la clase Game, es en sí mismo el contenedor de todos los componentes representado por las clases GameComponent y DrawableGameComponent, el cual se encarga de manejarlos automáticamente, lo que evita mucho trabajo de estructuración y programación al desarrollador.

Todos los GameComponent tienen su método Update donde se encuentra la lógica del juego para ese componente. Los DrawableGameComponent además añaden el método Draw, que contiene las órdenes para renderizar el componente. Estos métodos se llaman automáticamente y pueden ser reescritos para crear componentes personalizados.

XNA es una herramienta gratuita que permite a los desarrolladores crear juegos para la plataforma Windows y para la plataforma XBox 360.

XNA Game Studio Express

El entorno para el desarrollo en XNA Game Studio Express funciona sobre Visual C# 2005 Express Edition. Este IDE integra el cargador de contenidos mediante el Content Pipeline y la creación de proyectos predefinidos.

En la figura 28 se muestra el IDE del framework XNA.

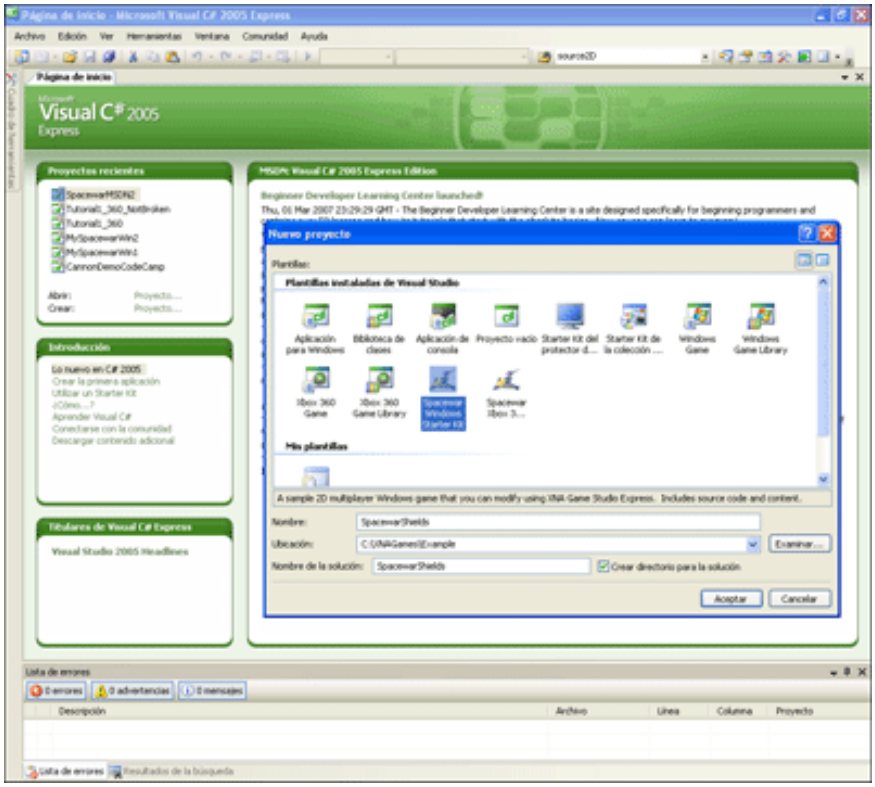


Figura 28. IDE de desarrollo Framework XNA.

2.9. Modelos de Inteligencia Artificial

Introducción a la Inteligencia Artificial

La Inteligencia Artificial comenzó como el resultado de la investigación en psicología cognitiva y lógica matemática. Se ha enfocado en la explicación del trabajo mental y construcción de algoritmos de solución a problemas de propósito general.

La inteligencia artificial es la ciencia que enfoca su estudio a lograr la comprensión de entidades inteligentes.

Es una disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura.

Existen distintos tipos de conocimiento y medios de representación del conocimiento. También se distinguen varios tipos de procesos válidos para obtener resultados racionales, que determinan el tipo de agente inteligente. De más simples a más complejos, los cinco principales tipos de procesos son:

- Ejecución de una respuesta predeterminada por cada entrada (análogas a actos reflejos en seres vivos).
- Búsqueda del estado requerido en el conjunto de los estados producidos por las acciones posibles.
- Algoritmos genéticos (Análogo al proceso de evolución de las cadenas de ADN).
- Redes neuronales artificiales (Análogo al funcionamiento físico del cerebro de animales y humanos).
- Razonamiento mediante una Lógica formal (Análogo al pensamiento abstracto humano).

También existen distintos tipos de percepciones y acciones, pueden ser obtenidas y producidas, respectivamente por sensores físicos y sensores mecánicos en máquinas, pulsos eléctricos u ópticos en computadoras, tanto como por entradas y salidas de bits de un software y su entorno software.

Inteligencia aplicada a los Videojuegos

Dentro de los videojuegos la inteligencia artificial es una de las características más importantes que se le atribuye a un motor gráfico ya que provee estímulo al juego. La inteligencia artificial de un juego puede ser compleja, ya que dentro de ella se define algunos pasos para su aplicación:

1.- Establecer NPC(Non Placer Characters): Consiste en definir la línea base del comportamiento de los NPC, es decir, de los personajes que no hacen el

papel del jugador, para lo cual se debe comenzar definiendo la actividad que va a representar un NPC dentro del juego, se debe delimitar el escenario, tomando en cuenta que el personaje no solo estará en medio del mundo del juego, sino que también éste interactuará con él. Estas características del personaje son realizadas mediante la toma de decisiones, en base a un sistema de reglas para las acciones a las que se puede someter el personaje dentro del juego.

El entorno reactivo es uno de los más importantes conceptos aplicado al desarrollo de videojuegos, el cual se basa en una relación causa-efecto; es decir, el jugador provoca algo y en consecuencia ocurre algún suceso. Un ejemplo de este tipo de entorno se lo encuentra en el juego *Sims* (Figura29), en el cual los jugadores evolucionan solos.

Figura 29. Imagen del Juego Sims.

Los grupos clásicos de NPC son los siguientes:

- Amigos: Son personas que ayudan brindando información o algún objeto. Un buen ejemplo es el mercader de muchos juegos que, en la mayoría de los casos, no se mueve del sitio o, circula por una zona muy limitada del escenario.
- Aliados: Este grupo pueden ser los personajes que acompañan durante un juego peleando a favor. Tienen muchas animaciones y, por tanto,

comportamientos, ya que tienen que hacer lo mismo que el jugador: desplazarse, atacar, estar en reposo, hablar, etc.

- Neutrales: Personajes que habitan dentro del entorno como, transeúntes principalmente, que pueden girar al ver al personaje que hace de jugador o con los cuales se puede chocar.

- Enemigos: Son grupos de NPC que atacan. Se puede hacer una subdivisión por tipos de enemigos donde podría haber cuatro clases, por ejemplo, los mercenarios, los asesinos, animales, etc.

- Jefes finales: Quizás el grupo más importante, junto con los aliados. El concepto de jefe final es un poco ambiguo pero básicamente quiere decir que son los enemigos a los que el jugador se enfrenta para pasar de nivel o misión y que requieren un gran esfuerzo.

2.- Definir grupos de comportamientos: Permite definir las situaciones de juego que se pueden dar y a las que deben reaccionar los NPC. Entre las situaciones más conocidas en los videojuegos se tiene:

- Situación de Ataque: Se tiene enemigos en pantalla y se debe definir la forma de ataque (por grupos o de forma individual, si los enemigos rodean o atacan a distancia, etc.).

- Situación de patrulla: Los enemigos defienden una zona del escenario y se tiene que entrar en ella. Los enemigos patrullarán de una forma concreta y por una zona bien limitada.

- Situación de alerta: Esta situación se da cuando un enemigo ha visto al personaje que hace de jugador y da una señal de alarma. Al sonar las alarmas, han pasado a comportamiento de alerta. Los movimientos serán más rápidos y ya no habrá animaciones de espera. Lógicamente, habría que definir cómo se pasa de un comportamiento de patrulla a otro de alerta.

- Situación de peligro: Ésta suele ser para los NPC neutrales o aliados. Un ejemplo de esta situación se encuentra como ejemplo en juegos de guerra como Call Of Duty 4, cuando un enemigo tira una granada, los aliados ayudan a buscar un refugio para resguardarse de la explosión, con lo que se consigue un

entorno mucho más reactivo.
- Situación neutral: Una de las más básicas. La gente se mueve de un lado para otro y, cuando se choca contra ellos, protestan.
- Situación obtener recursos: Esta situación permite obtener más técnicas de ganar el juego según las ayudas que se presenten en el juego.

3.- Establecer Reglas de Evolución: Estas reglas permiten diferenciar las posibilidades a las que van a ir cambiando (accesorios, ropa, carácter, etc).

4.- Definir el momento de evolución: En esta parte interviene la jugabilidad, es decir la forma en que el jugador interactúe con la aplicación. Se deben definir dos tipos: situaciones de evolución creadas por el jugador y las creadas aleatoriamente sin ninguna regla. Ésta última deberá ser la más superficial, como por ejemplo la evolución física del personaje. Pero la primera regla deberá aportar jugabilidad (rapidez, velocidad, daños, etc.).

5.- Definir una propia Evolución: Para lo cual se debe crear misiones que se apliquen cuando el jugador evoluciona hacia algo en concreto, esto permitirá dar un abanico de jugabilidad fantástico al jugador, además de una gran profundidad a la historia. En los juegos de rol se hace más interesante, porque se puede equipar al personaje con determinados poderes o con diversas armas.

Para establecer patrones de comportamiento en los juegos se utilizan máquinas de estados, otra de las aplicaciones de la Inteligencia Artificial.

Máquinas de Estados

Es un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores. En este modelo se define un conjunto de estados que sirve de intermediario en esta relación de entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la

máquina, de forma tal que la salida depende únicamente del estado y las entradas actuales.

Cuando el conjunto de estados es finito, se habla de una *máquina de estados finitos* (o *FSM* por *finite state machine*). Frecuentemente se prescinde del calificativo finito, por lo que al decir *máquina de estados* se suele referir a una de estados finitos.

Las máquinas de estados se pueden clasificar en *aceptoras* o *transductoras*:

- Aceptoras: (también llamadas reconocedoras): Son aquellas donde la salida es binaria (si/no), depende únicamente del estado y existe un estado inicial. Puede decirse, entonces, que cuando la máquina produce una salida "positiva" (es decir, un "si"), es porque ha "reconocido" o "aceptado" la secuencia de entrada. En las máquinas de estados aceptoras, los estados con salida "positiva" se denominan *estados finales*.
- Transductoras: Son las más generales, que convierten una secuencia de señales de entrada en una secuencia de salida, pudiendo ésta ser binaria o más compleja, depender de la entrada actual (no sólo del estado) y pudiendo también prescindirse de un estado inicial.

Teoría de autómatas

La teoría de autómatas es una rama de las ciencias de la computación que estudia matemáticamente máquinas abstractas y problemas que éstas son capaces de resolver. La teoría de autómatas esta estrechamente relacionada con la teoría del lenguaje formal ya que los autómatas son clasificados a menudo por la clase de lenguajes formales que son capaces de reconocer.

Un autómata es un modelo matemático para una máquina de estado finita (FSM sus siglas en inglés). Una FSM es una máquina que, dada una entrada de símbolos, "salta" a través de una serie de estados de acuerdo a una función de transisión (que puede ser expresada como una tabla). En la variedad común "Mealy" de FSMs, esta función de transición dice al autómata a qué estado cambiar dados un estado y símbolo determinados.

Existen dos tipos de autómatas, estos son:

Autómata finito determinista (AFD): Cada estado de un autómata de este tipo tiene una transición por cada símbolo del alfabeto (figura30). [6]

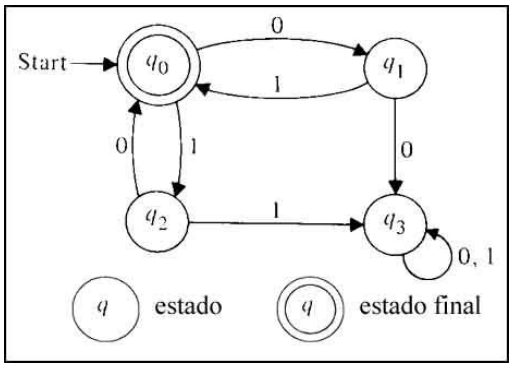


Figura 30. Gráfico de máquina de estado finito7.

Autómata finito no determinista (AFND): Los estados de un autómata de este tipo pueden, o no, tener una o más transiciones por cada símbolo del alfabeto. El autómata acepta una palabra si existe al menos un camino desde el estado q_0 a un estado final F *etiquetado* con la palabra de entrada. Si una transición no está definida, de manera que el autómata no puede saber como continuar leyendo la entrada, la palabra es rechazada. Esto se observa en la figura 30.

Autómata finito no determinista, con transiciones (AFND): Además de ser capaz de alcanzar más estados leyendo un símbolo, permite alcanzarlos sin leer ningún símbolo. Si un estado tiene transiciones etiquetadas con ε, entonces el AFND puede *encontrarse* en cualquier de los estados alcanzables por las transiciones ε, directamente o a través de otros estados con transiciones ε. El conjunto de estados que pueden ser alcanzados mediante este método desde un estado q, se denomina la clausura ε de q[8].

[6] Tomado de: Pathfinding. FSMs. Redes Neurales. Entrenamiento Genético. Lógica Borrosa. Vida Artificial. Comportamiento Emergente.

[7] Tomado de: http://es.wikipedia.org/wiki/Motor_de_juego.

[8] Tomado de: http://www.pc-actual.com/consejos/paso/2008/06/29/Inteligencia-artificial-de-un-videojuego-piensan-nuestros-enemigos.

En los videojuegos se empieza en un estado, y cuando se cumple una condición, se salta a otro estado (ver figura 31). Tanto las condiciones como los estados pueden ser abstracciones de cosas más complejas. De hecho, hasta pueden ser tratados como otros autómatas aparte.

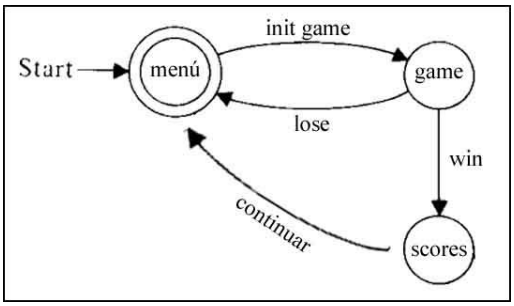


Figura 31. Aplicación de estados en los videojuegos.

Un ejemplo de este concepto se puede observar en el siguiente gráfico (figura32), en el comportamiento de un personaje. Los estados asociados a este personaje son: buscar enemigo, si éste es encontrado su próximo estado será el de persecución y al encontrarse a una distancia adecuada, pasará al estado disparar.[9]

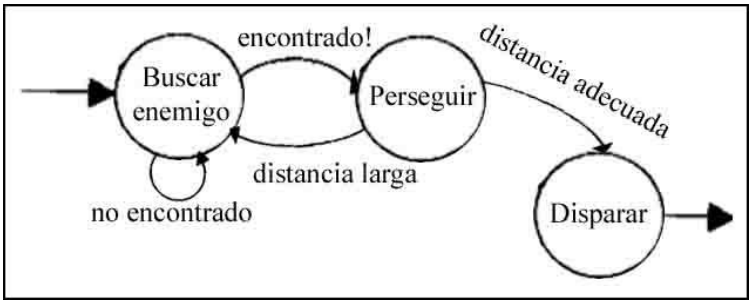


Figura 32. Aplicación de estados en los videojuegos.

[9] Tomado de: http://www.pc-actual.com/consejos/paso/2008/11/17/Inteligencia-Artificial-de-un-videojuego-evolucion-de-un-personaje.

2.10.Consolas de Tercera Generación

Para exportar un videojuego a una consola es importante tomar en cuenta el tipo de videojuego que se va a desarrollar, pues de ello depende la elección de la consola que se va a utilizar para su implementación y posterior uso, ya que cada consola trae una serie de características específicas y únicas como el rendimiento en sus gráficos o el Gameplay, por tanto no es lo mismo implementar juego para la consola Wii que uno para la consola Xbox 360, ni es igual diseñar un título para PSP que otro para la consola de Nintendo DS por lo cual se debe tener en cuenta estas particularidades que, aunque genéricas, permiten dar una idea global de la consola que debemos usar para el juego que se ha desarrollado.

Otro factor a considerar son las licencias al exportar a una consola, si el software, en este caso el juego va a ser comercializado. Es necesario solicitar una autorización para comercializar el juego, para lo cual se toma en cuenta la calidad del producto a comercializar. Cada empresa que fabrica las consolas de videojuegos tiene precios diferentes de licencias que van desde los dos mil dólares (Nintendo), siendo ésta la más asequible del mercado.

Antes de exportar un juego es necesario usar herramientas que permiten probar el Videojuego y comprobar sus características técnicas. Estas herramientas son aplicables independientemente del motor 3D utilizado.

Figura 33. Consolas de Videojuegos.

Actualmente, en el mercado existe gran cantidad de consolas de videojuegos. Entre las más actuales tenemos:

Portátiles:

Son consolas de mano, relativamente pequeñas en cuanto a su tamaño. Entre ellas tenemos:

Nintendo DS

Es la más manejable y permite el desarrollo de Videojuegos más creativos, los cuales tendrían características como soplar o hablarle a la consola.

Por todo esto el diseño del Videojuego será completamente distinto, puesto que no requerir gráficos espectaculares y su desarrollo es relativamente más sencillo. Esta consola es más utilizada por el mercado femenino.

Figura 34. Nintendo DS.

PSP(Play Station Portable)

Tiene una mayor nitidez en cuanto a gráficos, su mando es normal y se juega de forma habitual. En esta consola se puede implementar un Videojuego más exigente y a un nivel de dificultad similar a los juegos que se exporten para la consola PS2.

Figura 35. Pay Station Portable.

Consolas:

Nintendo Wii

Se pensó que esta consola no iba a triunfar en comparación a sus compañeras de mercado las cuales tenían unos gráficos sorprendentes, pero fue todo lo contrario pues ofreció una nueva forma de Gameplay la cual atrajo a Gamers de toda clase incluso aquellos que no les gustaban los Videojuegos. El mando del Wii hace que el desarrollador de Videojuegos piense primero en que hacer para sacarle la máxima diversión al mando.

El diseño del Videojuego tendrá que tener en cuenta cómo realizar mecánicas de juego divertidas que impliquen el uso especial del mando. Así, el desarrollo es más complicado que para la Nintendo DS pero más asequible que hacer una producción para PlayStation 3 o Xbox 360. Por lo que toca al público, lo que más abunda para la Wii son jugadores casuales, muchas chicas y, sobre todo, personas que no se quieren complicar la vida con juegos difíciles y buscan diversión rápida, sencilla y que pueda disfrutarse con varios amigos.

Figura 36. Nintendo Wii.

PlayStation 3

Las consolas de Sony son las que más han tenido seguidores y mejores títulos, aunque ahora su más potente trabajo decayó por el mal manejo de sus precios, aunque parece ser que está recuperando el terreno que le ha pertenecido durante dos generaciones de consolas. Poco hay que hablar de esta consola y de su forma de jugar. Su mando es totalmente conocido y lo único que nos va a

suponer es encontrarnos con un desarrollo muy complejo, costoso y, sobre todo, largo y complicado.

Figura 37. PlayStation3.

Xbox 360

Esta consola ofrece gráficos y sonidos increíbles y hasta ahora lidera en el mundo de los Gamers Hardcore. El diseño sería idéntico que para la PS3. Técnicamente perfecta, las posibilidades de conseguir ambientaciones sonoras como el juego Lost Planet nos abre un amplio abanico de posibilidades. El desarrollo para la Xbox 360, aunque muy laborioso, es más aceptable, ya que por suerte la consola tiene muchas similitudes con el PC. Si a eso le sumamos el hecho de disponer de una plataforma de desarrollo conjunta para Xbox 360 y PC, denominada XNA y que es una interfaz de programación de aplicaciones, el tema mejora mucho. Una inestimable ayuda a la hora de desarrollar, muchas compañías desarrollan para esta consola por su facilidad en la obtención de las herramientas de desarrollo.

Figura 38. XBox 360.

CAPITULO III

Estructura y Diseño planteados para “Evolution”

3.1. Diagramas sobre el concepto de juegos

Un juego consta de una estructura básica sobre la cual se desarrolla el programa. Esta estructura está formada por tres etapas en las cuales el juego desarrolla su ciclo de vida.

El diagrama de un videojuego difiere del flujo básico de un software normal, ya que durante toda su ejecución el juego se encuentra ejecutando una acción hasta su finalización, es por ello que su flujo consta de un ciclo recursivo en la etapa de procesamiento, pues en esta etapa el juego se enfoca en la utilización y reproducción de aspectos visuales como son las coordenadas, imágenes, luces, cámaras, etc., elementos que no siempre están presentes en un software convencional y por tanto requieren un tratamiento diferente.

1. **Etapa de Inicialización**: Esta etapa como su nombre lo indica permite inicializar los datos que el juego implementará en su ciclo, como: librerías gráficas, el sistema de sonido. Aquí también se reserva memoria para objetos, posiciones de personajes, carga de puntajes y récords, también permite cargar las imágenes, sonido y recursos que intervienen en el juego.

2. **Ciclo del Videojuego**: Consta de un bucle en el cual se desarrolla toda la acción del juego. Este ciclo termina cuando el jugador pierde y abandona el videojuego. El ciclo consta de tres partes:

2.1 Entrada: Son todos los dispositivos de entrada por los cuales el jugador hace llegar la información de la acción que ha realizado al juego.

2.2 Procesamiento: Esta fase contiene toda la lógica que utiliza el juego para procesar los datos recibidos en la entrada. Según los datos de entrada se toman decisiones y se realizan cálculos de física e inteligencia artificial.

2.3 Salida: Permite mostrar la respuesta obtenida luego de procesar la información en el paso anterior. La “respuesta” se traduce en las imágenes y sonido que el juego muestra luego de procesada una acción.

3. **Finalización:** En esta fase se libera de memoria los recursos utilizados durante la ejecución del juego como, imágenes, sonido, música y se almacena los puntajes alcanzados.

El flujograma para el desarrollo (figura 39) de un videojuego es el siguiente:

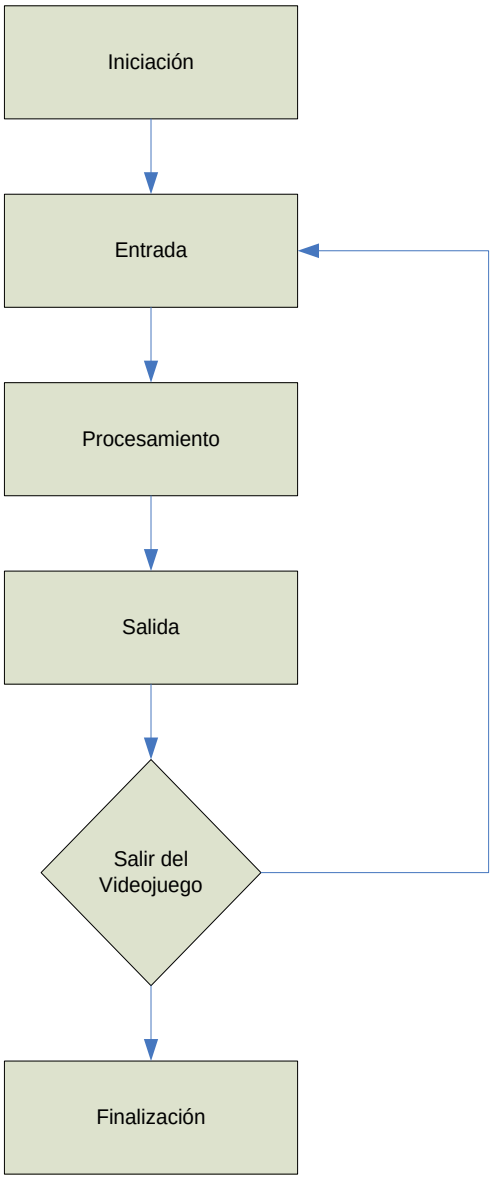


Figura 39. Ciclo de un Videojuego.

Sincronización del Juego

Debido a que un juego tiene un ciclo constante desde su inicio hasta su terminación y durante éste se ejecutan las operaciones matemáticas y físicas que el juego aplica para su funcionamiento, es importante tomar en cuenta que los objetos que intervienen en el juego deben moverse a la misma velocidad, independiente de la velocidad del computador donde se ejecute.

Para ello se debe sincronizar el juego para que tenga un desempeño óptimo y así evitar que éste corra a velocidades diferentes.

Existen dos formas para sincronizar un juego, basadas en framerate y sincronización en base al tiempo.

Sincronización por Framerate:

Consiste en sincronizar al juego en base al número de frames por segundo, para lo cual se detiene el ciclo principal del juego hasta que se ejecute el siguiente ciclo. Es decir si el juego se encuentra corriendo a una velocidad de 24 frames por segundo, se fija el framerate a que cada ciclo dure 0.041 segundos, cuando el ciclo del juego se haya ejecutado se verifica si pasaron 0.041 segundos desde que se inició el juego para pasar a ejecutar el siguiente ciclo.

Esta sincronización es aplicada para un computador con características mínimas de velocidad, puesto que si el juego es ejecutado en un computador de menores características, el juego se volverá más lento.

Figura 40. Ejemplo de Sincronización por Framerate.

Sincronización en Base al Tiempo:

Al implementar esta sincronización no es necesario tomar en cuenta el framerate que tenga el juego, ya que independientemente del valor en su framerate los caracteres se moverán con la misma rapidez sin importar la velocidad de la computadora donde se ejecute el juego, lo cual es una ventaja de esta sincronización. El inconveniente principal de esta sincronización es el uso de

variables que pueden afectar la precisión del juego, además se debe manejar el error redundante si la computadora a utilizarse es más lenta, lo cual dará un movimiento menos fluido al juego.

3.2. Diseño y modelado de escenarios y caracteres

El diseño y modelado es una de las más importantes tareas en el desarrollo de un videojuego ya que del buen trabajo que se logre en su diseño, dependerá que se obtenga un producto que logre cautivar la atención del jugador, y por ende el éxito del juego.

Escenarios:

El escenario comprende los gráficos de paisajes y entorno en general, animación, música, ambientación y escenografía.

En Evolution se diseñaron dos escenarios. La diferencia radica en la herramienta con la que fueron modelados.

El la figura 41 se puede observar el modelo de isla diseñado con XNA, mientras que en la figura 42 se observa el modelo de la isla realizado utilizando la herramienta Torque, la cual define de mejor manera el paisaje, permitiendo obtener mejor calidad de texturas y relieves mas reales.

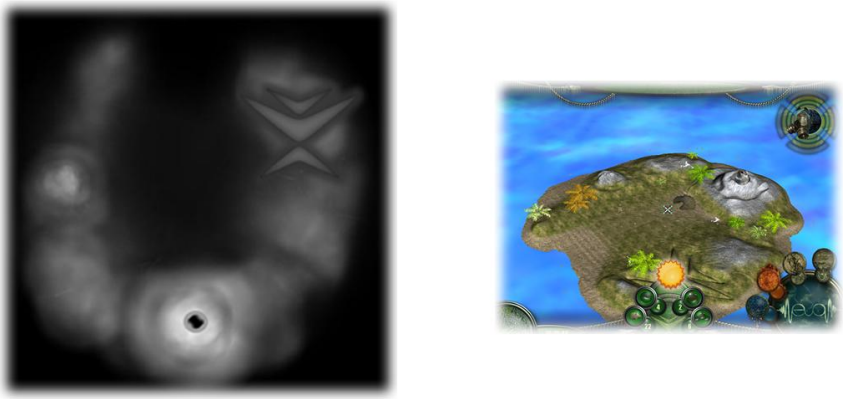


Figura 41. Diseño y Modelado de Escenarios.

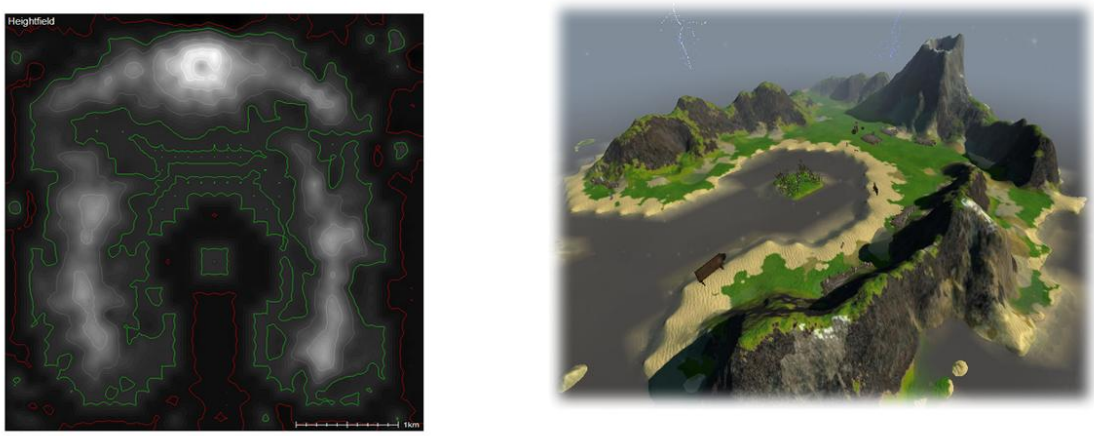


Figura 42. Diseño y Modelado de Escenarios.

Los caracteres corresponden a todos lo personajes y animales que forman parte del juego y que complementan el entorno del juego.

Personajes (habitantes):

En base a los bocetos para cada personaje se diseñan las diferentes animaciones que el personaje va tomando mientras se desenvuelve en el juego.

Para el diseño de cada animación es necesario cumplir con las siguientes condiciones:

- Dibujar los personajes de acuerdo a los movimientos que éste realice.
- Armar los fotogramas para cada secuencia de animación del personaje.
- Digitalizar la secuencia de imágenes del personaje construyendo su esqueleto en base a polígonos.
- Convertir los dibujos en fotogramas utilizando una herramienta de diseño.
- Unir cada fotograma en una secuencia de movimientos en un intervalo de tiempo.
- Igualar los fotogramas para que concuerden unos con otros.
- Aplicar a los fotogramas texturas, volumen y afinar su forma.
- Redimensionar la forma de cada personaje.
- Aplicar sombras y capas en las imágenes.
- Delinear bordes y fondos de las imágenes.

- Exportar las imágenes en archivos de tipo PNG y BMP.

En las figuras 43 y 44 se observan ejemplos de la creación de un personaje usando la herramienta 3D MAX.

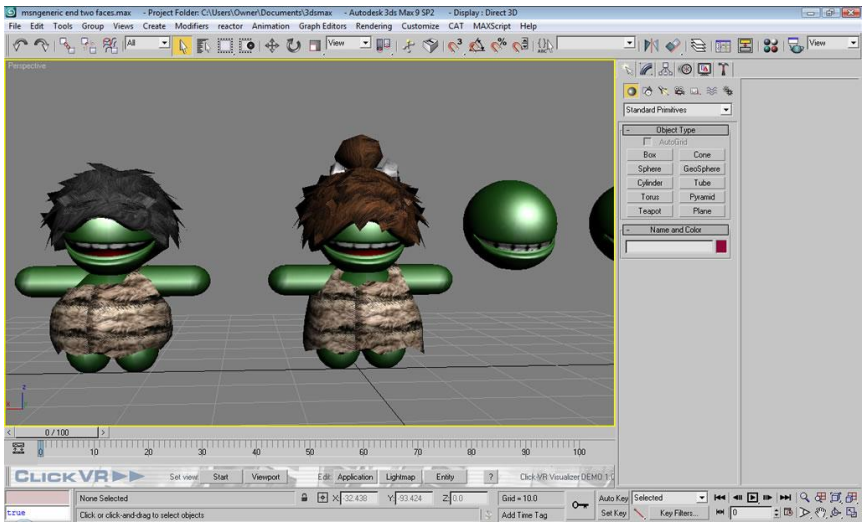


Figura 43. Ciclo de un Videojuego.

Figura 44. Ciclo de un Videojuego.

Animales:

Las características de cada animal son las que permiten definir las texturas aplicadas a cada uno de ellos.

De la misma manera, conocer los movimientos que cada animal realice serán necesarios para elaborar el boceto de secuencia de pasos que ejecute el animal en el juego.

Para realizar la animación de los animales del juego se deben seguir los siguientes pasos:

- Diseñar bocetos con las figuras de cada animal y aplicar texturas y volumen a su cuerpo.
- Ubicar en orden cada dibujo siguiendo una secuencia de movimientos.
- Digitalizar cada imagen tomando como base polígonos para crear su esqueleto.
- Aplicar sombras a los animales y editar los bordes de cada gráfico.
- Exportar los gráficos a archivos de tipo PNG y BMP.

En las figuras 45 y 46 se observan ejemplos de la creación de los animales y las diferentes texturas aplicadas.

Figura 45. 3D MAX (T-Rex Rendering original) y 3D MAX (T-Rex Rendering optimizado)

Figura 46. 3D MAX (T-Rex Rendering)

Tanto para modelado de escenarios y personajes se aplica la técnica de modelado Vértice por Vértice.

Técnica de Modelado Vértice Por Vértice

Esta técnica consiste en crear vértices por cada gráfico a dibujar.

Permite dividir en pequeños polígonos al gráfico basándonos en los ejes x, y, z.

Cada vértice consecutivo puede ser seleccionado y movilizado en diferente posición según lo requiera el dibujo.

En la figura 47 se observan los vértices de la cabeza del muñeco con los cuales se arma el diseño.

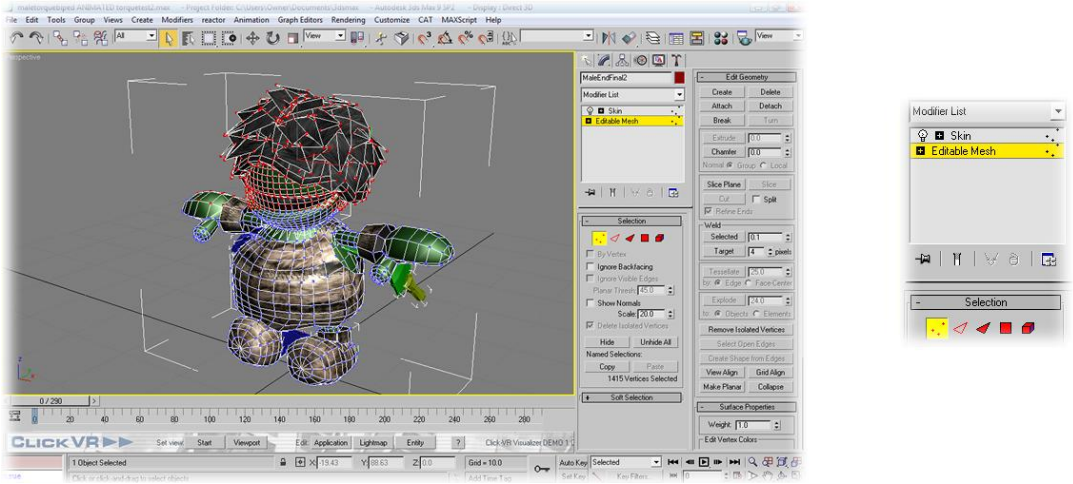


Figura 47. 3D MAX (T-Rex Rendering)

Existen además otras técnicas de modelado detalladas a continuación:

Técnica de Modelado Box Modelling

Esta técnica se basa en el boceto creado, ya que permite partir de lo general a lo particular, por ejemplo, si se quiere crear un ser humano primero se debe dibujar todo el cuerpo, para después irlo modelando parte por parte. Esta técnica ayuda a crear porciones grandes de un gráfico ya que ayuda a tener una idea de cómo va a quedar el modelo antes de detallarlo.

Técnica de Modelado por Rotoscopía

Esta técnica consiste en crear dibujos de frente, perfil, con perspectiva desde arriba y abajo. Esta técnica parte de un mismo elemento para ir dibujando diferentes rostros.

Es usada en conjunto con las técnicas Vértice por vértice y Box Modelling.

Técnica de Modelado por Edges Loops:

Esta técnica parte de los vértices para crear trazos en línea recta con lo cual se puede armar la estructura de un modelo, permitiendo una deformación perfecta en el caracter a la hora de animar o crear alguna pose.

En las caras los principales loops que se marcan son, el que abarca la boca, comienza desde la nariz y termina en la barbilla. Este es esencial para crear las deformaciones de la boca. También se crean loops en las cejas o de manera frontal.

Se debe tomar en cuenta que para el diseño y modelado de escenarios es indispensable contar con las herramientas apropiadas que permitan elaborar un juego atractivo al usuario.

En la figura 48 se muestra un ejemplo del producto final, en el cual se ha aplicado la técnica por Edge Loops y Vértice por Vértice.

Figura 48. Animación de Personajes.

3.3. Animación de Personajes

El proceso de animación de personajes (figura 49) para un video juego es uno de los procesos mas importantes ya que de este depende la estructuración del juego. Existen varios métodos para realizar la animación de personajes así como variedad en las herramientas para este propósito. Las herramientas de animación tienen que ser precisas a la hora de generar las secuencias pero al mismo tiempo tienen que ser óptimas para ahorrar recursos o performance. Por lo tanto la herramienta 3D Studio Max, siendo la mas utilizada para el desarrollo de aplicaciones 3D, será implementada a lo largo del desarrollo del juego.

Figura 49. Animación de Personajes.

El proceso de animación puede pasar de ser muy simple, en el caso de figuras pequeñas o poco complejas, hasta llegar a ser muy extenso y tedioso debido a la complejidad del proceso ya que este debe ser preciso, de lo contrario los resultados pueden llegar a ser desastrosos.

El trabajo no es sencillo y requiere de gran habilidad por parte de los creadores, es por esto que, dentro de las empresas dedicadas a la creación de videojuegos, se tiene un área exclusiva dedicada a la animación con un equipo o personal extenso, e inclusive, este proceso es realizado por otra compañía, ya que en muchos casos la animación se realiza en base a la captura del movimiento, para lo cual se necesita software y equipos especiales.

La captura de movimiento (Mocap: Motion Capture) permite integrar animaciones reales para personajes creados a diferencia del proceso manual o denominado (Key Framing) el cual será utilizado para el desarrollo del videojuego.

Figura 50. Motion Capture.

En la figura anterior (figura 50) se muestra una secuencia de una persona realizando un giro hacia atrás creada con el proceso de captura de moviendo, el cual sería muy difícil de realizarlo con el proceso manual (key frames). Por lo cual la animación depende mucho de lo que se quiere conseguir. Si se elige el

proceso manual, la exigencia de animación debe ser mínima de otra forma el proceso se vuelve muy extenso o tedioso.

Existen pasos previos a la animación, que exigen que el modelo tridimensional tenga ciertas características las cuales son requeridas dependiendo del motor gráfico a ser implementado. Para el caso de la herramienta de Microsoft XNA, requiere que el modelo sea uniforme es decir que sea un único objeto y no varios independientes. Los requerimientos dependerán de la herramienta 3D, en este caso la herramienta 3D Studio Max para alcanzar los requerimientos necesarios por el motor gráfico.

Antes de empezar el proceso, se requiere definir algunos puntos importantes para su funcionalidad.

- Arquitectura del personaje: Es decir cómo se comporta el personaje, cómo camina, cómo actúa dentro del juego. En este caso el personaje tiende a la comicidad, por lo cual su comportamiento es errático y fuera de lo común.
- Cámara: Se define la ubicación de la cámara respecto al personaje, es decir si el juego se orienta a primera o tercera persona.
- Diagrama de Flujo: se refiere al flujo de la animación a tomar lugar dentro del juego como se muestra en la figura.

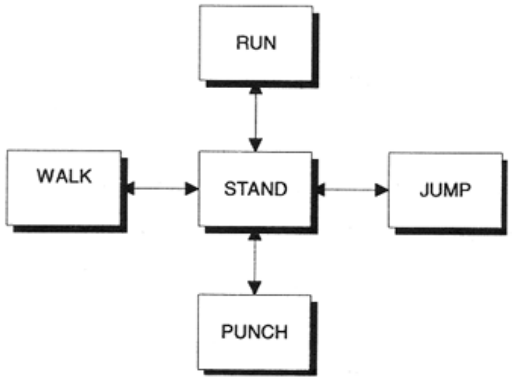


Figura 51. Flujo de Animación.

La figura 51 muestra el flujo de las animaciones, en donde la posición fija o (Stand) es la posición principal del personaje y la posición que generará el resto de posiciones, Correr (run), caminar (walk), Saltar (Jump), Pegar (punch), donde cada posición tiene que pasar por (Stand) para generar otra posición.

Todos los pasos mostrados necesitan ser analizados para no alterar el proceso durante el desarrollo, mejorando el desempeño del equipo de desarrollo. Una vez que se tiene claro el concepto, se utiliza la herramienta gráfica para crear la animación final.

Para comenzar el proceso de animación se debe tener finalizados y texturizados los modelos, ya que al modificar el modelo, el proceso de animación se tiene que empezar desde cero ya que pueden surgir varios inconvenientes a la hora de integrar con el motor gráfico. De igual forma los personajes deben contener la menor cantidad de polígonos posibles para una mayor eficiencia en el motor gráfico.

Figura 52. Habitante Evolution con Texturas.

Los pasos a seguir para realizar la animación dentro de la herramienta 3D Studio Max, se describen a continuación:

Exigencias Motor Gráfico

Antes que nada en esta etapa se definen ciertos aspectos que exigen los motores gráficos ya que varían dependiendo del motor y su aplicación. Los aspectos a tomar en cuenta son:

Tipo de formato: Se refiere al formato a ser exportado para un cierto motor gráfico en particular. Existen formatos universales, así como formatos exclusivos que solo pueden ser utilizados en una sola aplicación. Los formatos más comunes y universales son:

.X: El formato mas común usado en varios motores gráficos el cual es propiedad de Microsoft específicamente derivado de DirectX. Es comúnmente utilizado tanto para objetos estáticos, como para objetos dinámicos o animados.

.FBX: Un formato conocido pero poco compatible con herramientas gráficas. Muy difícil de manejarlo, requiere de mucha precisión, de otra forma generara muchos errores.

.DTS: Otro formato muy común para objetos estáticos y animados, debido a su eficacia y compatibilidad con el motor gráfico a utilizar es el más recomendado.

.DIF: Utilizado principalmente para interiores, así como para la creación de edificios y edificaciones en general. Altamente compatible con el motor gráfico.

.MAP: Muy similar a (.DIF) pero no compatible con el motor gráfico a utilizar.

(.X) (.DIF) y (.DTS) son los formatos más compatibles con el motor gráfico, por lo tanto van a ser utilizados durante el proceso.

Escala: Se refiere al tamaño de los objetos ya que existen diferencias en tamaños dentro de las herramientas gráficas así como dentro del motor gráfico, debido a las diferentes escalas que poseen. En el caso de 3D Studio Max, 1 metro equivalen a 10 metros en XNA. Para lo cual se necesita tener configurado dentro de la herramienta gráfica para no encontrar dificultades con el tamaño de objetos en el mundo virtual.

Tipo: El tipo es la composición del personaje dentro de la herramienta gráfica. Los objetos pueden ser (Poly) o (Meshes) los cuales no difieren en la herramienta gráfica pero si tienen gran diferencia dentro del motor ya que dependiendo de la estructura, el motor produce y calcula: sombras (Shadows) brillo (Shades) y relieves (Bump) en base a polígonos o meshes.

Formato: El formato del personaje principalmente esta relacionado con el tipo de texturas a implementar. El formato de las texturas, en este caso y en la mayoría de casos usa el formato (.jpg) debido al poco espacio que ocupa en disco. De la misma forma se toma en cuenta las dimensiones de las texturas, en donde se recomienda usar texturas con la menor dimensión posible ya que de igual forma influyen en el performance del juego. En este punto cabe resaltar que las dimensiones deben mantenerse uniformes tanto en el largo como en el ancho de la textura.

Diagrama de Flujo: El Diagrama de flujo se refiere a la serie de animaciones que se van a tomar en cuenta para cada personaje. Definidos por la lógica a ser implementada como se muestra en la figura siguiente (figura 53).

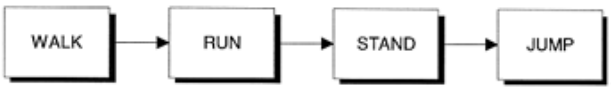


Figura 53. Diagrama de Flujo de Animaciones.

Animación

Esta etapa trata sobre el proceso de animación dentro de la herramienta donde se tienen varias subetapas, que son:

Preparación del Objeto o Personaje: Dentro de esta subetapa se tiene como objetivo establecer todos los parámetros de tal forma que el personaje u objeto puedan ser animados correctamente. Esto incluye: la verificación de texturas, Escalas y estructura del objeto o personaje.

Creación de Huesos: es la estructura principal y central, la cual permite la animación principalmente en personajes. El proceso puede ir de simple a muy extenso dependiendo de la complejidad del personaje. Esto esta dado por la creación de huesos de forma:

- **Manual**: Esta etapa es una de las mas largas de realizar ya que cada hueso es ubicado y configurado para obedecer comportamientos como en el caso de brazos donde al animar el brazo no puede ir atrás del codo.
- **Bípedos**: Por otro lado 3D Max Studio ofrece una cierta facilidad con personajes de forma humana, mediante el uso de Bípedos, los cuales son huesos con características humanas que son fácilmente configurables y adaptables, permitiendo una versatilidad, eficacia y ahorro de tiempo.

Secuencias de animación: En esta etapa se da vida al personaje con las animaciones de acuerdo al diagrama de flujo. En esta sección también se elije el método de animación, ya sea este manual (key frames) o mediante archivos de captura de movimiento (Motion Capture)

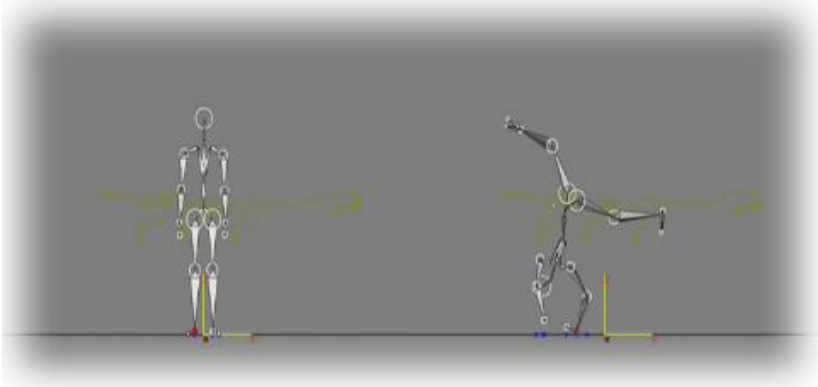


Figura 54. Motion Capture.

Exportación

Una vez animado el personaje u objetos se procede a exportarlos al formato más apropiado que se adapte al motor gráfico. Para lo cual la Herramienta 3D Studio Max ya ofrece varios formatos, pero para mayor seguridad se recomienda usar exportadores (Plugins) externos los cuales no solo facilitan el proceso, sino también ayudan a generar un objeto que realmente se adecúe a las exigencias del motor gráfico. Estos plugins en su mayoría son lanzados por las compañías propietarias del motor gráfico. Este es otro motivo por el cual la herramienta 3D Studio Max es de gran utilidad ya que este cuenta con una muy amplia gama de plugins y extras para el desarrollo de videojuegos.

En el proceso de exportación de objetos y caracteres los plugins a ser utilizados son los siguientes:

Pandasoft Direct X Exporter: El plugin permite exportar a formato (.X) los objetos animados o inanimados creados para ser adaptados al Motor Gráfico XNA.

Torque DTS Exporter: El plugin permite exportar a formato (.DTS) los objetos o personajes que contengan animaciones, huesos y secuencia lógica de animación.

3D Max Studio 3DS Exporter: El plugin principalmente permite exportar objetos que serán usados en interiores de escenarios como edificios, casas, etc. El plugin viene incluido en la herramienta 3D Studio Max.

Integración

La integración se da en la herramienta de desarrollo, así como en el motor gráfico. Una vez alcanzados los requerimientos de un modelo o personaje, este puede ser introducido en el motor gráfico para ser ejecutado en tiempo real, de tal manera que se pueda implementar una lógica a seguir por el juego.

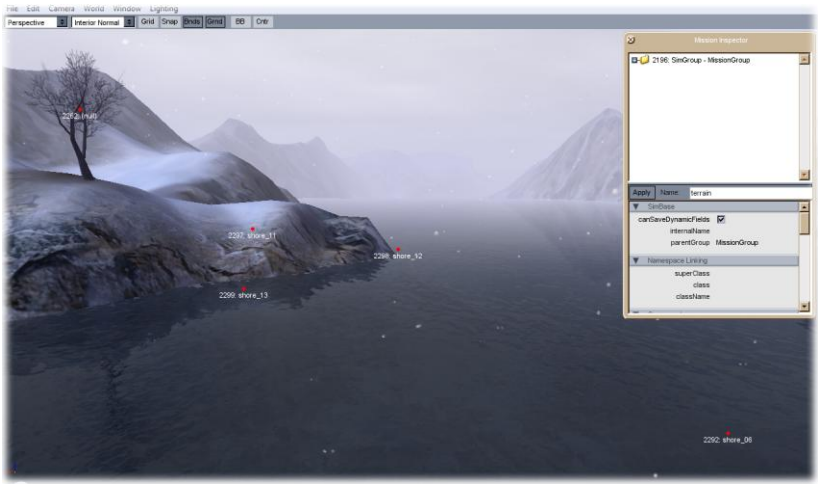


Figura 55. Integración con la herramienta de desarrollo.

3.4. DirectX para la exportación de Escenarios y Caracteres

DirectX SDK

Al descargar la herramienta SDK, esta incluye varias herramientas así como plugins que permiten trabajar y mejorar la experiencia del jugador. Una de las herramientas que incluye el SDK es el DirectX Viewer, el cual permite visualizar tanto los elementos estáticos, las animaciones así como los personajes exportados. La previsualización simula al objeto dentro del motor gráfico, tal como si estuviese dentro del juego.

Figura 56. (Previsualización Objeto 3D)

Esta herramienta permite visualizar muchos errores que pueden ser poco notorios dentro de la herramienta, errores que pueden ser fatales dentro de la aplicación, o simples errores en cuanto a la animación. Los errores pueden ser de diferente índole, como errores en texturas, en la adaptación de los huesos, o en la animación como se muestra en la figura 57.

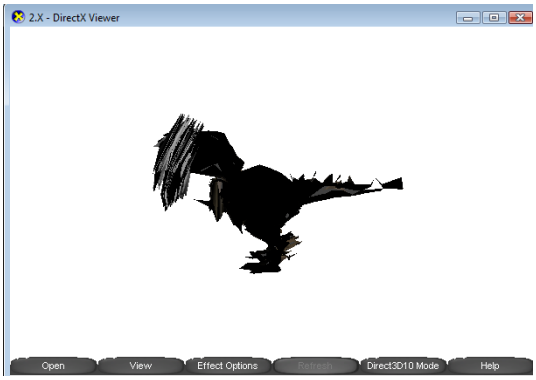


Figura 57. (Previsualización de Errores)

Una vez instalado el SDK, trabajará en conjunto con el plugin Pandasoft DirectX Exporter, previamente instalado en el proceso de animación. Por otro lado la herramienta permite previsualizar a los efectos dados a los objetos como efectos de luz (shaders) o relieve (Bump), así como una vista previa a objetos animados.

DirectX, también incluye prácticas herramientas para la edición de objetos y personajes como el mejoramiento en texturas y detalle, generando efectos de

luz y sombras. Efectos que serán reconocidos posteriormente por el motor gráfico a implementar mejorando la capacidad de editar objetos reduciendo tiempo y mejorando la experiencia visual del usuario final.

3.5. Modelos Matemáticos Implementados

Introducción a los Modelos Matemáticos

Antes de poder escribir un programa para desarrollar un juego es necesario construir un modelo matemático para representar simbólicamente los elementos de la situación o problema que se van a animar.

Un modelo matemático emplea fórmulas para expresar relaciones, proposiciones sustantivas de hechos, variables, parámetros y relaciones entre variables, u operaciones, para estudiar comportamientos de sistemas complejos ante situaciones difíciles de observar en la realidad.

El objetivo de un modelo matemático es entender el fenómeno para predecir un comportamiento futuro. Para elaborar un modelo matemático es importante seguir un proceso dado por los siguientes pasos:

- Encontrar un problema del mundo real
- Formular un modelo matemático acerca del problema, identificando las variables dependientes e independientes y estableciendo hipótesis lo suficientemente simples para tratarse de manera matemática.
- Aplicar los conocimientos matemáticos que se posee para llegar a conclusiones matemáticas.
- Comparar los datos obtenidos como predicciones con datos reales. Si los datos son diferentes, se reinicia el proceso.

Clasificaciones de Modelos

Los modelos matemáticos pueden clasificarse de la siguiente manera:

- Determinista: En este modelo se conoce de manera puntual la forma del resultado ya que no hay incertidumbre. Además, los datos utilizados para alimentar el modelo son completamente conocidos y determinados.
- Estocástico: Es un modelo probabilístico, en el cual no se conoce el resultado esperado, sino su probabilidad y existe por tanto incertidumbre.

Además con respecto a la función del origen de la información utilizada para construirlos los modelos pueden clasificarse de otras formas. Se puede distinguir entre modelos heurísticos y modelos empíricos:

- Modelos heurísticos: Son los que están basados en las explicaciones sobre las causas o mecanismos naturales que dan lugar al fenómeno estudiado.
- Modelos empíricos: Son los que utilizan las observaciones directas o los resultados de experimentos del fenómeno estudiado.

Además los modelos matemáticos encuentran distintas denominaciones en sus diversas aplicaciones.

Los modelos se clasifican también según su campo de aplicación:

- Modelos conceptuales: Son los que reproducen mediante fórmulas y algoritmos matemáticos más o menos complejos los procesos físicos que se producen en la naturaleza
- Modelo matemático de optimización: Los modelos matemáticos de optimización son ampliamente utilizados en diversas ramas de la ingeniería para resolver problemas que por su naturaleza son indeterminados, es decir presentan más de una solución posible.

Representación del modelo

La representación puede ser de la siguiente manera:

- Conceptual: Por una descripción cualitativa bien organizada que permite la medición de sus factores.

- Matemático: Se refiere a una representación numérica por aspectos lógicos y estructurados con aspectos de la ciencia matemática.

  Un modelo puede ser representado por números, letras, imágenes o símbolos que empleen ecuaciones que son traducidas en expresiones visuales basadas en aspectos cuantificables la ciencia matemática.
- Físico. Basado en aspectos de la ciencia física, movimientos de los cuerpos, y que además es cuantificable. Estos modelos generalmente representan el fenómeno estudiado utilizando las mismas relaciones físicas del prototipo pero reduciendo su escala para hacerlo manejable. Por ejemplo pertenecen a este tipo de modelo las representaciones a escalas reducidas de presas hidráulicas, puertos, etc.

Categorías por su aplicación

Por su uso suelen utilizarse en las siguientes tres áreas, sin embargo existen muchas otras como la de finanzas, ciencias etc.

- **Simulación**. De situaciones medibles de manera precisa o aleatoria, por ejemplo con aspectos de programación lineal cuando es de manera precisa, y probabilística o heurística cuando es aleatorio.
- **Optimización**. Para determinar el punto exacto para resolver alguna problemática administrativa, de producción, o cualquier otra situación. Cuando la optimización es entera o no lineal, combinada, se refiere a modelos matemáticos poco predecibles, pero que pueden acoplarse a alguna alternativa existente y aproximada en su cuantificación.
- **Control**. Para saber con precisión como está algo en una organización, investigación, área de operación, etc.

En Evolution la aplicación de modelos matemáticos es utilizada en el diseño de modelos de objetos en 3D, gráficas y para el realismo físico del juego, en donde se aplican conocimientos de física matemática, inteligencia artificial y geometría.

Gráficos Vectoriales

Una imagen vectorial (figura 58) es una imagen digital formada por objetos geométricos independientes (segmentos, polígonos, arcos, etc.), cada uno de ellos definido por distintos atributos matemáticos de forma, de posición, de color, etc.

Las imágenes en los gráficos vectoriales no se construyen píxel a píxel, sino que se forman a partir de vectores, los cuales son objetos formados por una serie de puntos y líneas rectas o curvas definidas matemáticamente.

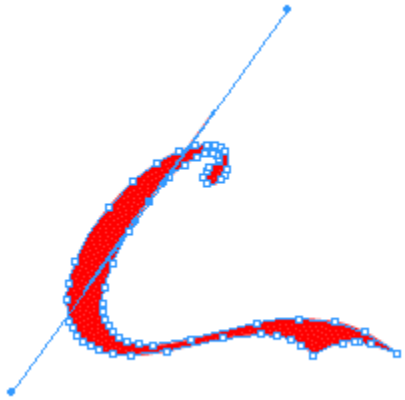


Figura 58. Gráfico Vectorial.

Los gráficos vectoriales están formados por curvas de Bézier las cuales son curvas de tercer grado que permiten representar la curvatura de un vector.

Una curva Bézier (figura 59) queda totalmente definida por cuatro puntos característicos, los puntos inicial y final de la curva (nodos o puntos de anclaje) y dos puntos de control (puntos de control, manejadores o manecillas), invisibles en el gráfico final, que definen su forma. Para modificar su forma, basta cambiar de posición uno de sus nodos o uno de sus puntos de control.

Las curvas son capaces de adaptarse a casi cualquier forma imaginable, por lo que son muy usadas para diseñar logotipos e iconos y para copiar cualquier figura.

También son enormemente versátiles, pudiendo adoptar desde curvaturas muy suaves como líneas rectas y curvaturas muy fuertes, pasando por todos los

valores intermedios. Pueden, incluso, cambiar de cóncavas a convexas alrededor de un punto.

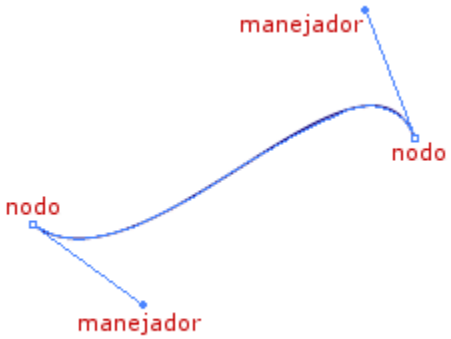


Figura 59. Curvas de Bézier.

Los gráficos vectoriales tienen la característica de poder ampliar el tamaño de una imagen sin sufrir el efecto de escalado que sufren los gráficos rasterizados. De igual manera permiten mover, estirar y retorcer imágenes de manera sencilla. El uso de estos gráficos está muy extendido en la generación de imágenes en tres dimensiones tanto dinámicas como estáticas.
Las imágenes vectoriales se almacenan como una lista que describe cada uno de sus vectores componentes, su posición y sus propiedades.

Los gráficos vectoriales son independientes de la resolución, ya que no dependen de una retícula de píxeles dada. Por lo tanto, tienen la máxima resolución que permite el formato en que se almacena.

Para representar gráficos vectoriales en un computador, es necesario traducir los gráficos vectoriales a gráficos rasterizados o mapa de bits ya que la pantalla está constituida físicamente por píxeles.
En la figura 60 se muestra un ejemplo de la representación vectorial utilizada en el diseño de Evolution.

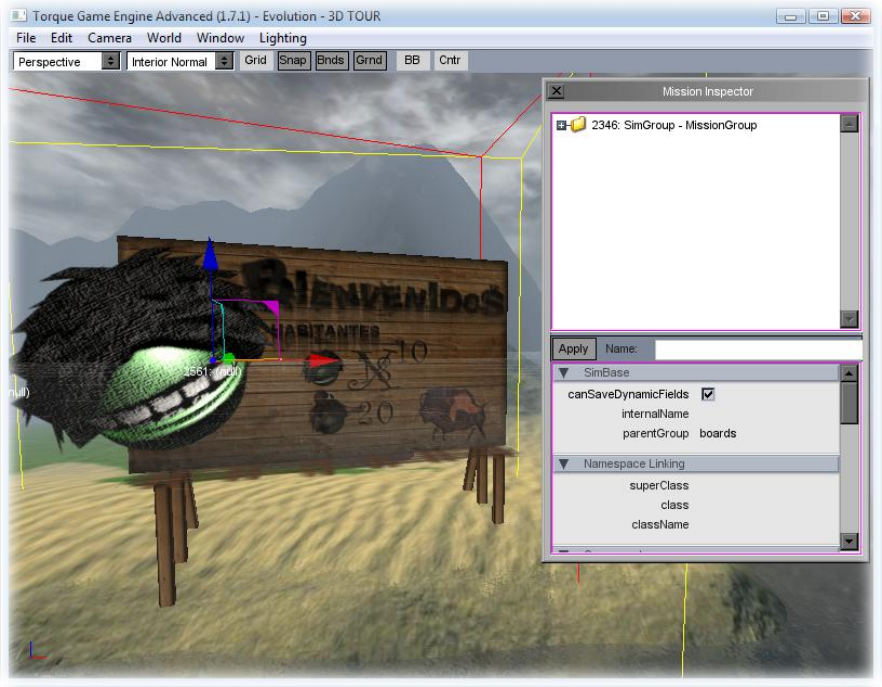


Figura 60.Gráficos vectoriales

Operaciones vectoriales

- Muchos generadores de gráficos vectoriales permiten rotar, mover, reflejar, estirar, inclinar y realizar finas transformaciones de los objetos, como combinar objetos primarios para formar objetos más complejos

- Hay otro tipo de operaciones de un nivel más sofisticado que incluye acciones sobre objetos cerrados tales como: unir o soldar, combinar, interceptar y diferenciar.

3.6. Diseño del Motor Gráfico

El motor Gráfico de una herramienta en particular permite controlar, generar y renderizar en tiempo real los modelos y escenarios 3D. Es decir, el motor controla los gráficos de la aplicación y procesos involucrados, y por otro lado también permite generar la lógica, inteligencia artificial, eventos y física.

La figura 61 muestra un ejemplo de colisiones utilizando el motor gráfico.

Figura 61. Motor gráfico que permite dar movimiento a los caracteres.

Físicas (Physics)

Una de las principales características que un motor gráfico ofrece, es el manejo de físicas en tiempo real, lo cual permite a los desarrolladores diseñar y crear ambientes reales que reaccionen o se comporten de forma lógica y de acuerdo a leyes físicas predeterminadas como gravedad, fuerzas externas, detección de colisiones, entre otras.

Cada motor gráfico ofrece una lógica en físicas diferente en cuanto a los cálculos a realizar dentro de la simulación. Los métodos en que el motor gráfico calcula las físicas empleadas van a influir directamente en el rendimiento de la aplicación. Es por esto que al diseñar un juego se debe buscar un motor gráfico que se adecue o adapte a las exigencias del juego.

XNA ofrece varias soluciones en cuando al cálculo de físicas, en donde se utilizan métodos que van de acuerdo a las necesidades de rendimiento. Los cálculos pueden ser complejos para simulaciones reales y precisas así como pueden ser más simples para un mejor desempeño disminuyendo la exactitud.
En el caso de simulaciones que requieran de más precisión, se tiene el uso de métodos o cálculos en base a vértices y polígonos, teniendo como ventaja la

exactitud pero en cuanto al desempeño la aplicación decaerá dependiendo del hardware y otros factores netamente técnicos, ya que los cálculos en base a vértices o polígonos se basan en la estructura del modelo 3D, por lo cual mientras mas complejo sea el modelo, el calculo tomara mas tiempo y recursos del sistema. Es por esto que el uso de tarjetas graficas es fundamental.

Figura 62. (Calculo de colisiones en terreno en base a vértices)

Otro método comúnmente usado para mejorar el performance en aplicaciones 3D, es el uso de cálculos en base a figuras geométricas básicas (figura 62), como el: cuadrado, circulo, cilindro, capsula, etc. Lo cual mejora notoriamente el desempeño aumentando el (frame rate) o cuadros por segundo en la aplicación. El cubo es la figura que menos recursos utiliza por lo cual es la más utilizada principalmente en detección de colisiones como se muestra en la figura siguiente (figura 63).

Figura 63. (Detección de colisiones en base a figuras geométricas)

Todo el desempeño dependerá de la plataforma así como del hardware a ser implementado. Por lo cual se tiende a diseñar y crear de acuerdo a las exigencias en hardware ya que no es lo mismo desarrollar para una consola ya sea esta un PlayStation, Xbox o Nintendo debido a que cada una de ellas posee características en hardware totalmente distintas. En el caso de las consolas de tercera generación, es factible crear aplicaciones que involucren eventos o simulaciones reales ya que el hardware se encuentra más evolucionado y permite que el desarrollador pueda crear sin límites que previas consolas mantenían.

En cuanto al desarrollo en consolas de segunda generación, las simulaciones o efectos de física, se basaban exclusivamente en eventos predeterminados o previamente simulados, los cuales nunca cambian a diferencia de las consolas de tercera generación las cuales permiten crear simulaciones en tiempo real, lo cual hace que la aplicación no sea repetitiva ni mucho menos preestablecida o muy monótona.

Figura 64. (Simulado)

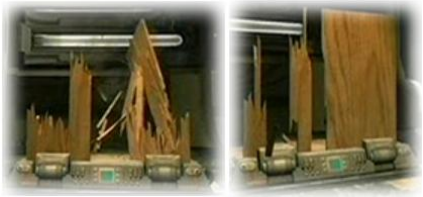


Figura 65. (Tiempo real)

El motor gráfico que ofrece la herramienta XNA, permite simular las físicas más elementales que un juego de tercera generación requiere (figura 64 y 65), como lo es: la gravedad, detección de colisiones y reacción. Donde la gravedad y detección de colisiones permiten determinar parámetros como peso y velocidad.

Gráficos (Graphics)

Por otro lado, una característica fundamental en el desarrollo de un videojuego de tercera generación, esta relacionado al diseño y a la calidad grafica que ofrecen los motores, los cuales dependiendo de la capacidad de procesamiento pueden generar escenarios u objetos realmente impresionantes con detalles asombrosos.

Cada motor existente en el mercado ofrece un sinfín de opciones para mejorar la calidad grafica, así mismo la diferencia se vera reflejada en el precio de cada motor, ya que a mayor calidad, mayor es el precio. Esto es debido a que la calidad grafica de un videojuego es un factor clave a parte de la trama o Plot, para que el juego sea un éxito o un simple fracaso.

Figura 66. Logotipo Juego Gears of War.

Ejemplos para demostrar esto, se pueden apreciar en juegos que utilizan motores gráficos como (Unreal) o (Crytek), los cuales han sido base para crear los juegos más exitosos y con una calidad grafica insuperable, como lo es el juego (Gears of War (figura 66)) de Microsoft. El cual explota las capacidades graficas del Xbox de una manera inigualable, mostrando escenarios realmente impresionantes.

Los motores gráficos se enfocan principalmente al uso de sombras y reflejos los cuales permiten hacer objetos más reales y estilizados. En la figura a continuación se muestra la comparación entre un objeto normal y el mismo objeto implementado shaders y sobras.

Figura 67. (Objeto Sin Efectos)

Figura 68. (Objeto con Efectos)

El ejemplo muestra claramente la diferencia visual en cuanto a la calidad y aproximación a la realidad (Figura 67 y 68).

Shaders

Para un mejor entendimiento Shader es parte del proceso de generación de imágenes que realiza el motor gráfico siendo el responsable de crear el color adecuado del objeto.

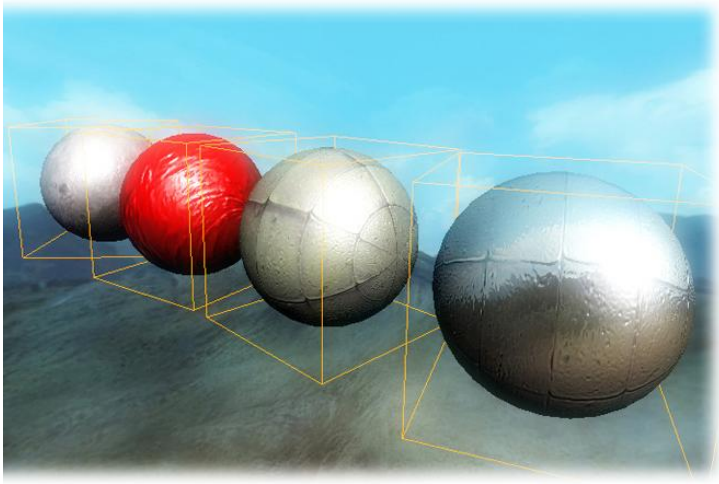


Figura 69. (Diferentes Tipos de Shaders)

El uso de Shaders (figura 69) dentro del desarrollo de juegos mejora la experiencia visual del jugador ya que permite simular patrones como: reflejo, simulación de metales, plásticos, madera, entre otros.

Bump (Alto Relieve)

La técnica de alto relieve o (Bumping) que se muestra en la figura 70, permite crear superficies irregulares en base a objetos planos o imágenes, simulando un relieve en la superficie del objeto tomando en cuenta la opacidad de la imagen.

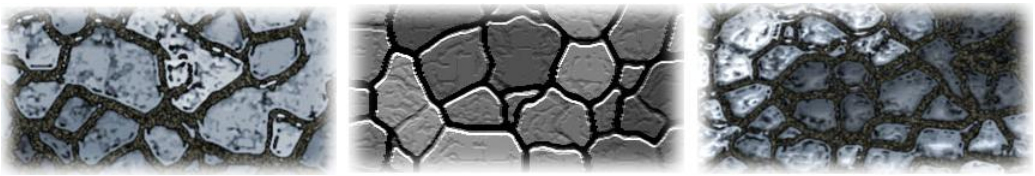


Figura 70. Ténica de alto relieve.

Este efecto es producido por la claridad y opacidad de la imagen en base a la escala de grises. En donde mientras mas clara la imagen o mientras mas blanca, el objeto adquiere un mayor relieve. Por otro lado mientras mas oscura la imagen o mas negra, el objeto no presenta tanto relieve o nada en lo absoluto.

Height Map

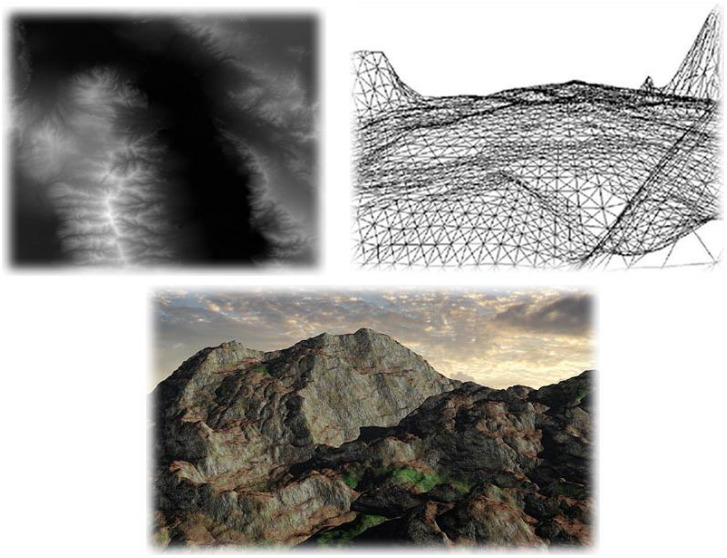


Figura 71. (Proceso de un Height Map)

Un mapa de altura o (height map) (figura 71) se basa en el mismo concepto de un Bump map, es decir que un height map, a partir de una imagen puede

generar relieves o profundidad de acuerdo a la coloración. La diferencia con un Bump map es que un Height map se utiliza para generar principalmente terreno como por ejemplo montañas y paisajes. Como se muestra en la figura la imagen en blanco y negro genera las profundidades y relieves para generar un sistema montañoso.

3.7. Manejo de Escenarios y Caracteres

El manejo de escenarios y caracteres dentro del motor gráfico de XNA puede ser realizado de dos formas, las cuales poseen ventajas particulares así como de desventajas que afectan principalmente la eficiencia en el desarrollo, pero solo una de ellas es la mas utilizada e implementada por desarrolladores en la mayoría de juegos actuales. La primera forma puede ser descrita como Manual ya que requiere de codificación, mientras que la segunda forma es grafica ya que el entorno de trabajo es netamente visual. A continuación se muestran ambos métodos en el manejo de escenarios y caracteres los cuales van a ser implementados dentro de las etapas de desarrollo de la aplicación.

Método Manual o Codificado

El método manual permite al programador crear y manipular los objetos dentro del escenario en forma de código (figura 72), es decir que tanto la escala como la ubicación de todos los objetos en el juego se lo realizara en base a programación y cálculos matemáticos. Sin duda mediante este método se obtiene una excelente precisión pero el tiempo y costo para realizar una aplicación es muy alto ya que si se requiere construir un escenario extenso o integrar varios objetos en la escena el uso de este proceso tomaría un tiempo extremadamente largo.

Figura 72. boxModel = content.Load<Model>("content/plane");

Por lo cual la complejidad del método en cuestión no satisface las necesidades de un equipo de desarrollo de videojuegos, haciendo de este un método poco usado en el desarrollo de grandes proyectos. Otra gran desventaja esta dada en cuanto a la visualización de objetos, debido a que al programar, se necesita ejecutar el programa constantemente para constatar que cada objeto se encuentre en su lugar y tenga la escala correcta por lo cual el programador deberá ser muy hábil en el manejo de objetos y personajes.

Método Gráfico

El método gráfico, dentro del manejo de escenarios y caracteres es relativamente nuevo en el desarrollo de aplicaciones 3D ya que en su mayoría este método ha sido exclusivo de grandes compañías de desarrollo que disponen de motores gráficos exclusivos o que disponen de altos presupuestos para comprar los motores existentes, los cuales ofrecen soluciones bastante practicas para el desarrollo de aplicaciones 3D así como de facilidades y herramientas que ayudan a disminuir tiempo y esfuerzo por parte de los desarrolladores.

En la siguiente figura se observa un ejemplo utilizando el motor gráfico Unreal.

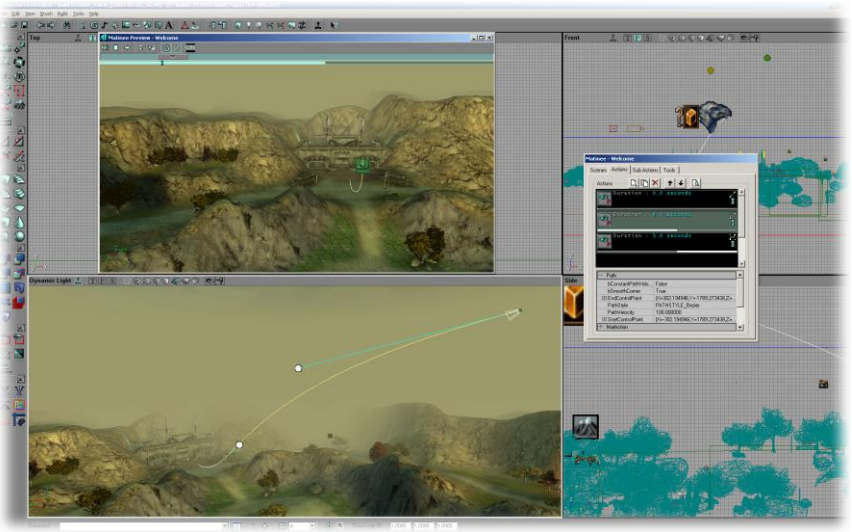


Figura 73. Motor Unreal (Unreal Engine)

Hoy en día existen pocas herramientas las cuales ofrecen similares características a un bajo costo, pero los resultados todavía no se aproximan a los generados por las grandes empresas. La herramienta Torque similar a XNA permite integrar escenarios y caracteres de una forma más interactiva y rápida, emulando en tiempo real los resultados de cada cambio en el ambiente tridimensional como muestra la figura 74.

Figura 74. Escenario realizado en Torque.

Las ventajas del método gráfico son superiores a las del método manual, permitiendo administrar a los objetos en escena así como modificar varios aspectos como es el terreno y ambiente. Torque incluye herramientas de edición

de terreno y texturas lo cual no es posible en XNA. Otra gran solución es el uso del editor en tiempo real del juego pudiendo pasar del modo editor al modo juego tan solo presionando (F11). Como se muestra en la figura 75 a continuación.

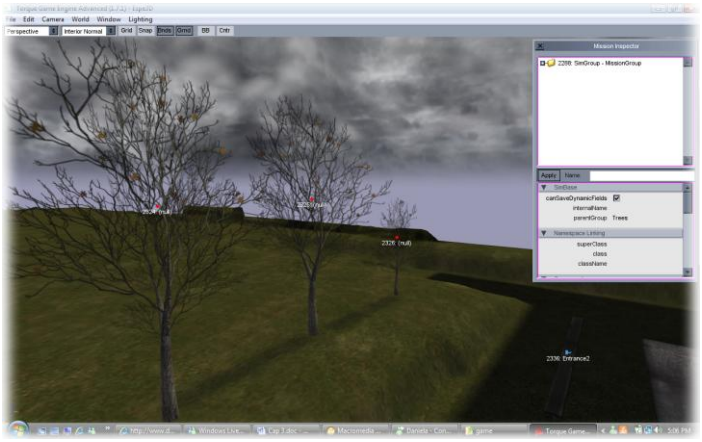


Figura 75. Editor de Torque.

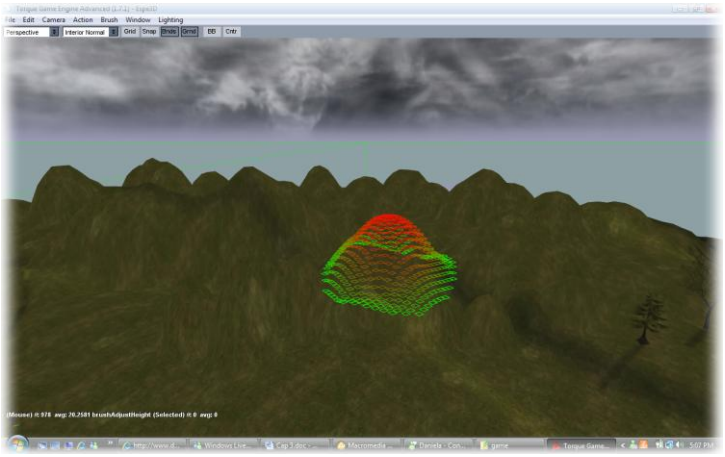


Figura 76. Edición de Terreno (Height map)

En la figura 76, se muestra la edición de terreno y generación de texturas del mismo, lo cual ahorra código y tiempo por parte de los desarrolladores. En cuanto al manejo de objetos y caracteres, depende en gran parte a la función que van a realizar cada uno de ellos. Para objetos estáticos o inanimados la herramienta permite integrarlos de manera interactiva a través de una biblioteca de archivos previamente generados (figura 77).

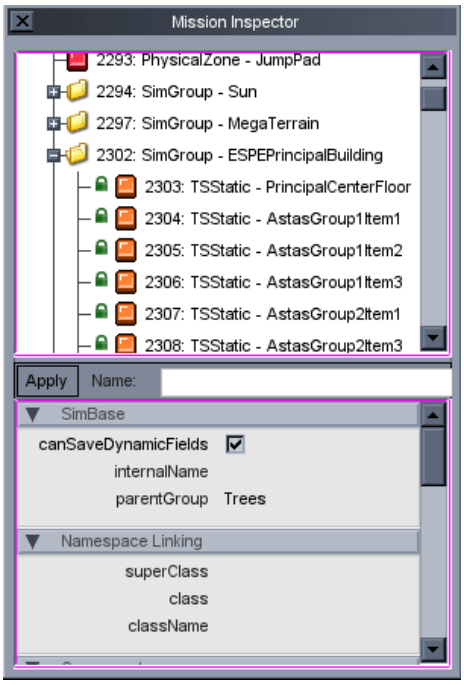


Figura 77. (Menú Edición)

El menú de integración y edición de objetos permite cambiar parámetros como nombre, escala, posición entre otros. Lo mismo ocurre para objetos animados, pero única y exclusivamente para objetos de naturaleza repetitiva o que no interactúan de forma directa con el usuario. Por ejemplo, un molino el cual girara de manera continúa en el escenario. Por otro lado, para el caso de objetos o caracteres animados interactivos (figura 78), el proceso requiere de codificación, la cual ayude a establecer tiempos, animaciones, etc. En la figura a continuación se muestra un personaje el cual es integrado mediante código para su posterior animación e interactividad con el usuario.

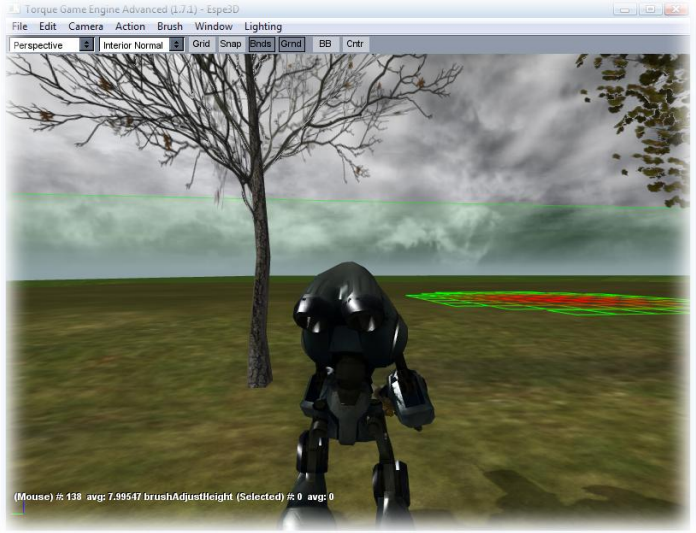


Figura 78. (Manejo de caracteres)

Las herramientas ayudan en gran parte al proceso de diseño y animación pero también pueden ser utilizadas para el desarrollo de la lógica y control, como el manejo de cámaras y eventos dinámicos.

3.8. Manejo y control de Cámaras

El control de cámaras permite a los videojuegos adquirir un mayor realismo al transmitir la imagen al jugador además de buscar una manera óptima de determinar la información que se debe mostrar y la manera de conseguirlo, para lograr una mejor expresividad e impresión sobre el espectador.
En un videojuego se debe tomar en cuenta aspectos como los parámetros de entrada para generar movimientos mas sutiles debido a que la cámara en la mayoría de los juegos se mueve por el por el teclado, mouse o j*oystick***,** para lo cual se debe existir un correcto mapeo entre el parámetro de entrada y el sistema de cámaras, evitando la oclusión de elementos en el cono de visión.

Los movimientos y posiciones no son totalmente libres ya que están conectados directa o indirectamente con la interactividad que realice el jugador dentro del videojuego. Es por ello que las cámaras actúan de manera, estática, dinámica,

en primera persona cuando las cámaras corresponden a los ojos del personaje en el juego, en tercera persona cuando se tiene un conjunto de posiciones prefijadas, o un rango entre una distancia máxima y mínima, orientadas hacia el personaje principal reproduciendo los movimientos de éste, y en escenarios donde el jugador tiene el control absoluto de la cámara.

Frustum

Indica el volumen 3D de visualización de una cámara en un punto determinado. El tamaño de volumen afecta la forma en que una imagen es proyectada desde el espacio de cámara hasta la pantalla. Para poder visualizar los objetos en forma de cámara se emplea la proyección en perspectiva, la cual permite hacer que los objetos cercanos a la cámara aumenten en tamaño y difieran de los objetos situados a distancia.

La vista frustum (ver figura 79) forma una pirámide con punta recortada entre la cámara y el plano de vista, donde el plano que se encuentra entre los planos de corte de frente y de atrás corresponde a la vista frustum.

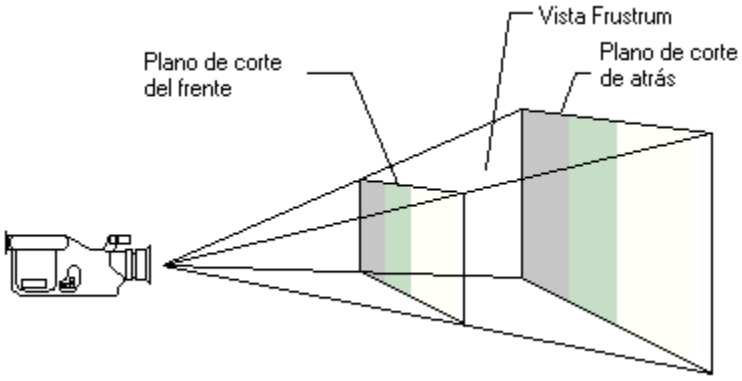


Figura 79. Vista Frustum.

Para enfocar un objeto se deben tener en cuenta aspectos como:

Encuadre: Es la relación que existe entre la cámara y el objeto que se va a enfocar. El encuadre se puede enderezar tomando como referencia los bordes verticales y horizontales de la pantalla.

Zoom: Es la capacidad que tiene la cámara para acercar o alejar el encuadre con respecto al objeto enfocado.

Se debe tomar en cuenta la perspectiva que se quiere tener para la imagen ya que ésta permite agrandar, alargar, reducir o dar movimiento al objeto. Con esto mientras la cámara se acerca o se aleja se logra aumentar o disminuir la percepción de perspectiva que se tiene sobre el objeto.

Proyección

La proyección gráfica es una técnica de dibujo empleada para representar un objeto en una superficie. La figura se obtiene utilizando líneas auxiliares proyectantes que partiendo de un punto, denominado foco, reflejan dicho objeto en un plano –a modo de sombra como se observa en la figura 80.

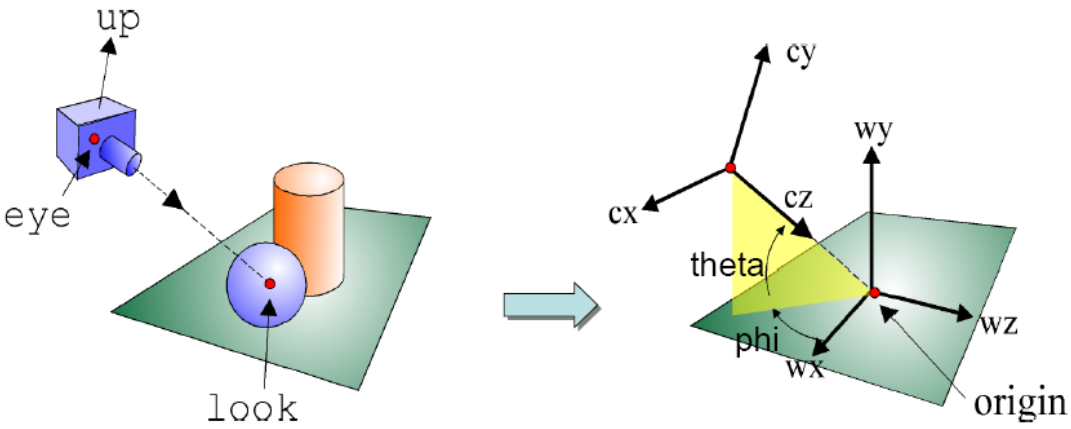


Figura 80. Proyección.

Los elementos principales de la proyección son el punto de vista o foco de proyección (V), el punto que se desea proyectar (A), el punto proyectado (A'), la línea proyectante (VAA') y el plano sobre el que se proyecta, que recibe diferentes denominaciones como plano de proyección, plano de cuadro o plano imagen como se observa en las imágenes 81 y 82.

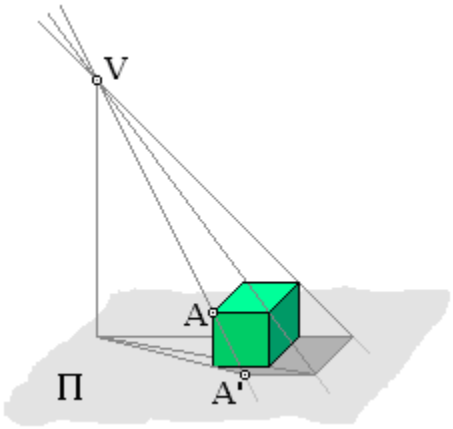


Figura 81. Proyección Central.

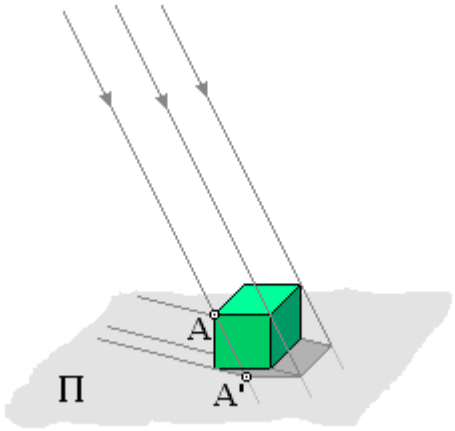


Figura 82. Proyección Paralela.

Tipos de Planos

Existen múltiples tipos de planos, pero tradicionalmente éstos son los principales:

- **Plano Panorámico o Gran Plano General:** El plano panorámico muestra un gran escenario o una multitud. Tiene un valor descriptivo y puede adquirir un valor dramático cuando se pretende destacar la soledad o la pequeñez del hombre frente al medio. Se da así más relevancia al contexto que a las figuras que se enfocan. También se utiliza para mostrar los paisajes.

- **Plano General:** El Plano general (P.G.) presenta a los personajes de cuerpo entero y muestra con detalle el entorno que les rodea.

- **Plano Conjunto:** El Plano conjunto (P.C.) es el encuadre en donde se toman la acción del personaje principal con lo más cercano. Podría ser un ejemplo, una conversación. Puede salir un solo personaje o más de uno. Si dentro de este plano no hay un movimiento interno, podría transmitir los mismos significados que un plano general.

- **Plano Figura**: El plano figura (P.F.), es el encuadre donde los límites superiores e inferiores coinciden con la cabeza y los pies del sujeto.

- **Plano Entero:** El Plano entero (P.E.), o plano figura, denominado así porque encuadra a todo el sujeto.

- **Plano Americano:** El Plano americano (P.A.), o también denominado 3/4, o plano medio largo, recorta la figura por la rodilla, aproximadamente.

- **Plano Medio:** El Plano medio (P.M.) recorta el cuerpo a la altura de la cintura. Se correspondería con la distancia de relación personal, distancia adecuada para mostrar la realidad entre dos sujetos, como es el caso de las entrevistas.

- **Plano Medio Corto:** El Plano medio corto (P.M.C.) capta el cuerpo desde la cabeza hasta la mitad del pecho. Este plano permite aislar una sola figura dentro de un recuadro, y descontextualizarla de su entorno para concentrar en ella la máxima atención.

- **Primer Plano:** El Primer plano (P.P), en el caso de la figura humana, esta incluye el rostro y los hombros. Este tipo de plano, al igual que el Plano detalle y el Primerísimo primer plano, se corresponde con una distancia íntima, ya que sirve para mostrar confidencia e intimidad respecto al personaje.

- **Primerísimo Primer Plano:** El Primerísimo primer plano (P.P.P.) capta el rostro desde la base del mentón hasta la punta de su cabeza. También dota de gran significado a la imagen.

- **Plano Detalle:** El Plano detalle (P.D.), recoge una pequeña parte de un cuerpo u objeto. En esta parte se concentra la máxima capacidad expresiva, y los gestos se intensifican por la distancia tan mínima entre

cámara y sujeto/objeto. Sirve para enfatizar algún elemento de esa realidad. Destaca algún detalle que de otra forma pasaría desapercibido.

Angulaciones de la Cámara

Es importante a la hora de componer una imagen de estar consciente de los elementos visuales que distraen a la vista, ya que dependiendo del ángulo con el que se compone una imagen se da una distinta comunicación visual. Para que un cambio de ángulo funcione debe ser mínimo de 30º acompañado de un cambio de plano para que se note en la edición.

- **Normal**: El ángulo de la cámara es paralelo al suelo.
- **Picado**: La cámara se sitúa por encima del objeto o sujeto mostrado, de manera que éste se ve desde arriba.
- **Contrapicado**: Es el opuesto al picado.
- **Nadir o Supina o Contra Picado Perfecto**: La cámara se sitúa completamente por debajo del personaje, en un ángulo perpendicular al suelo.
- **Cenital o Picado perfecto**: La cámara se sitúa completamente por encima del personaje, en un ángulo también perpendicular.

En la figura 83 se muestra el primer plano de una vista con una proyección central.

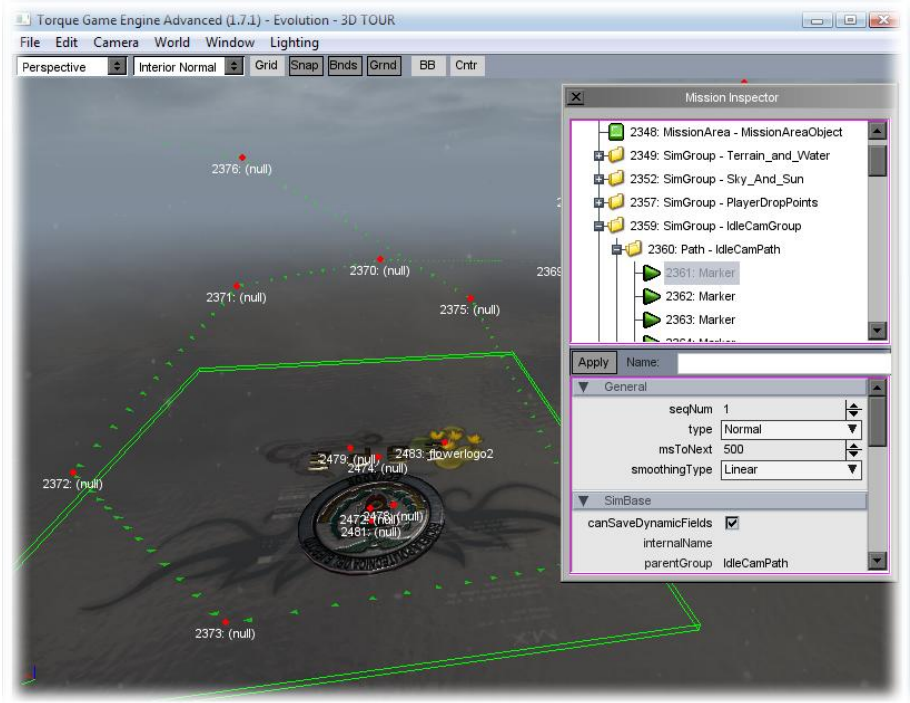


Figura 83. Primer Plano de enfoque de cámara.

3.9. Control de Iluminación y Sombras

El control e iluminación de sobras se da dentro del motor gráfico. Dependiendo del motor gráfico, la simulación de sombras será más precisa o simulada. Existen varios métodos y procedimientos para cálculos de sombras de acuerdo a la iluminación (figura 84), los cuales son posibles dependiendo de la capacidad gráfica del computador en cuanto a la potencia de la tarjeta gráfica.

Figura 84. Proyección de Sombras.

Cuando no es posible procesar sombras, debido a las limitaciones técnicas, se puede recurrir a otros mecanismos que reduzca el consumo de recursos. Estos mecanismos eran comúnmente usados en consolas de segunda generación, las cuales no disponían de la capacidad grafica actual para calcular sombras en tiempo real. Por lo cual se recurría comúnmente a usar una imagen fija la cual nunca iba a cambiar sin importar las condiciones de luz. La figura 85, muestra un efecto de sombra pre-simulado, mediante el uso de una imagen de forma circular el cual se proyecta en superficies planas dando una ilusión de sombra al objeto.

Figura 85. Efecto básico (Sombra)

Los motores actuales permiten en su mayoría renderizar los objetos tridimensionales dentro de la escena de una forma mas real calculando distancia profundidad y nivel de opacidad como se muestra en la figura 86.

Figura 86. (Calculo de sombras)

Sin embargo a pesar de la capacidad gráfica de computadores así como de consolas actuales, sigue siendo un inconveniente procesar sombras en varios objetos en escena. Por lo tanto es necesario disminuir la calidad de las sombras de acuerdo a la importancia del objeto, es decir para objetos o personajes principales o que son visiblemente notables, la calidad de sombras es normal, pero para objetos distantes se puede disminuir la calidad o simplemente no utilizar sombras para mejorar el performance.

Figura 87. (Niveles de Procesamiento de Sombras)

En la figura 87 es posible notar la diferencia en calidad de sombras pasando de no tener sombras a procesar las sombras en tiempo real con un nivel elevado de detalle.

Existen varias alternativas para desarrolladores de aplicaciones 3D que requieren optimizar sus aplicaciones, para lo cual solo se necesita un poco de visión para poder disminuir el consumo de recursos. Algunos métodos comúnmente utilizados en previas consolas son realizados directamente en las herramientas de diseño, con lo cual se consigue una optimización visual en base a las texturas implementas. Este método es denominado pre renderización o (Pre-Render), en donde dentro de la herramienta de diseño se ubican los objetos o personajes tal como si estuviesen dentro del motor gráfico, renderizando sombras y texturas (figura 88 y 89) para integrarlos posteriormente en la aplicación.

Figura 88. (Imagen normal) **Figura 89. (Imagen Pre renderizada)**

3.10. Efectos de Iluminación

Los efectos de iluminación permiten crear diversos ambientes dentro de las aplicaciones 3D. Su objetivo es dar diversos efectos visuales a objetos y escenarios sin afectarlos directamente en las aplicaciones de diseño, teniendo en cuenta su posterior implementación y efecto que se quiere recrear o emular en el usuario final. Por ejemplo, en juegos donde los desarrolladores desean recrear un ambiente tétrico y una sensación de miedo, se utilizan efectos de iluminación pobres en luz (figura 90), que permitan resaltar los factores previamente señalados. Por otro lado, en el caso de juegos más vistosos se utilizan tonalidades claras para denotar la emoción y vivacidad de la aplicación.

Figura 90. (Efectos de Iluminación)

Es posible crear efectos de iluminación y de postproducción en la fase final de renderización de cada cuadro o (frame) que genera el motor gráfico. Esta etapa

permite dar los colores adecuados a los objetos, simulando sombras, brillos y contrastes los cuales fueron previamente establecidos.

En los ejemplos a continuación (figuras 91, 92 y 93), se puede apreciar claramente la diferencia en efectos visuales generados en la etapa de renderización afectando la iluminación y control de sombras creando diferentes sensaciones a un único objeto en escena.

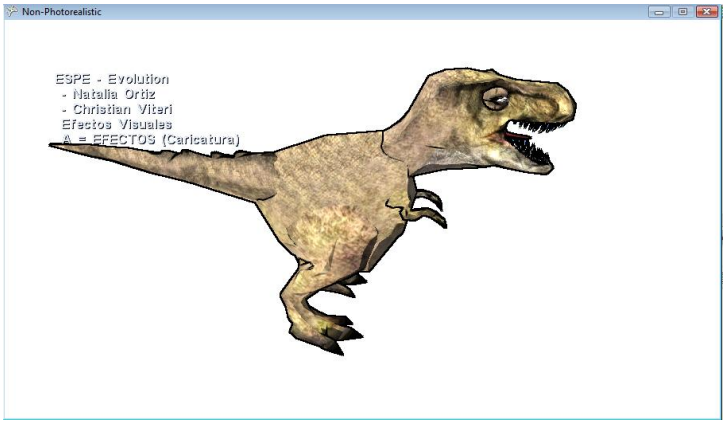


Figura 91. Efecto (Bordes)

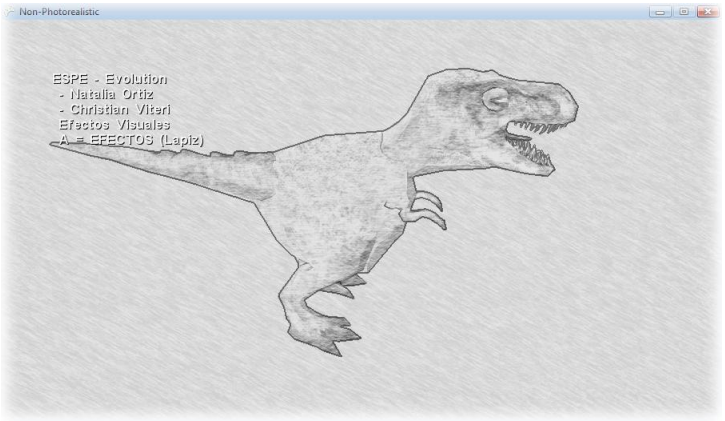


Figura 92. Efecto (Bosquejo)

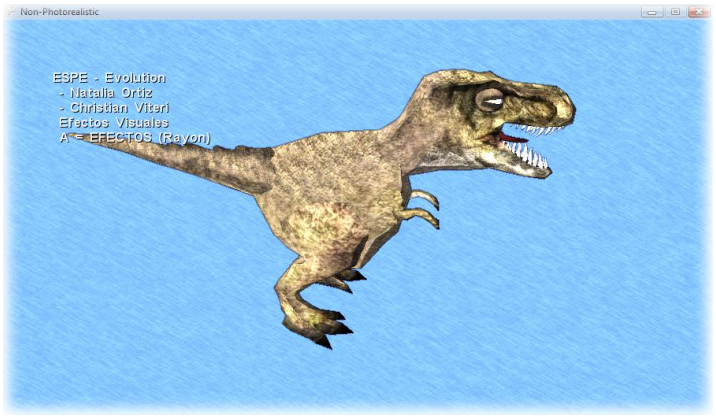


Figura 93. Efecto (Cartoon)

3.11. Análisis de la Inteligencia Artificial aplicada.

Máquinas de Estado Finitos

Las Máquinas de Estados Finitos, también conocidas como Autómatas de Estados Finitos, son modelos de comportamiento de un sistema o un objeto complejo, con un número limitado de modos o condiciones predefinidos, donde existen transiciones de modo. Estas están compuestas por 4 elementos principales:

- Estados que definen el comportamiento y pueden producir acciones.
- Transiciones de estado que son movimientos de un estado a otro.
- Reglas o condiciones que deben cumplirse para permitir un cambio de estado.
- Eventos de entrada que son externos o generados internamente, que permiten el lanzamiento de las reglas y permiten las transiciones.

Una máquina de estados finitos debe tener un estado inicial que actúa de punto de comienzo, y un estado actual que recuerda el producto de la anterior transición de estado. Los eventos recibidos como entrada actúan como disparadores, que causan una evaluación de las reglas que gobiernan las transiciones del estado actual a otro estado.

La representación que se da a estas máquinas es mediante grafos o diagramas de flujo que representan un sistema de control donde el conocimiento está representado en los estados, y las acciones están restringidas por las reglas.

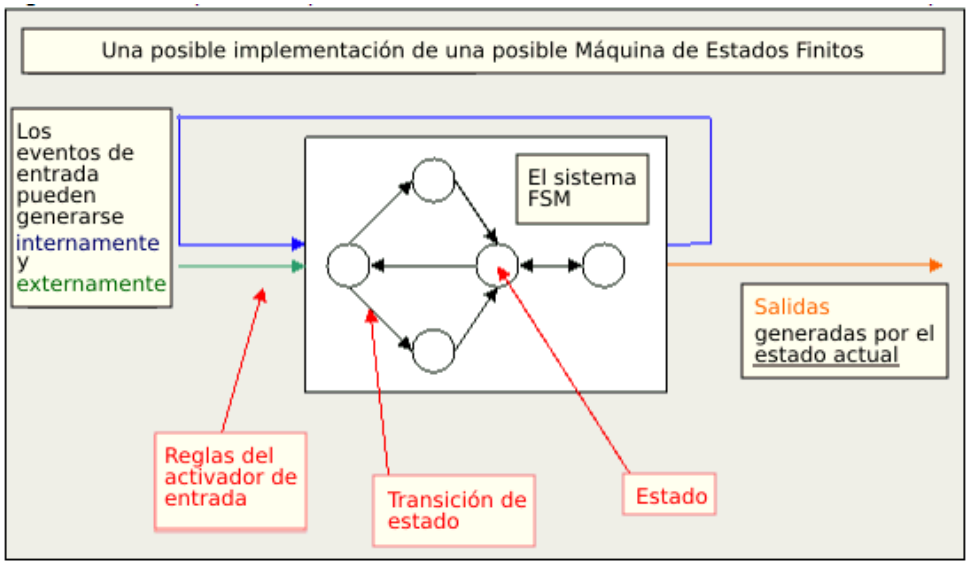


Figura 94. Una posible implementación de un sistema de control mediante Máquina de Estados Finitos

Las máquinas de estado finito se emplean en Evolution para definir la secuencia del juego y predecir los posibles movimientos que se pueden dar al producirse una acción.

Como ejemplo podemos citar una acción MoverPersonaje(), la cual puede ser usada tanto por el estado evadirEnemigo como por el estado AtacarEnemigo.

El estado evadirEnemigo puede consistir en muchas acciones, algunas evaluaciones y algunas directivas de movimiento. Si la entidad ha sido arrinconada, por ejemplo, puede haber una transición del estado evadirEnemigo a atacarEnemigo, donde la acción de ser arrinconado es el activador.

Análisis Práctico de las Máquinas de Estado Finitas

Como ejemplo de uso de una máquina de estado finita se pretende describir el funcionamiento de un juego llamado Quake, sobre el cual se va a aplicar los conceptos de FSM (Finite State Machine).

Quake es un videojuego de acción en primera persona que utiliza modelos tridimensionales para los jugadores y los monstruos en vez de sprites bidimensionales. Su historia habla de unos portales que conducen a otros mundos donde habitan seres malignos que han sido abiertos, y el jugador es la única persona que puede viajar a través de esos portales para cerrarlos.

Dentro de este juego se emplean misiles para atacar a los monstruos como se observa en la figura 95.

Figura 95. Enemigo sufre el dolor supremo.

Figura 96. Lanzagranadas en el juego Quake.

Un misil en Quake es un proyectil disparado por el arma Lanzadora de Misiles que puede ser poseída y operada por un jugador (figura 96).

En la siguiente figura (figura 97), se muestra una representación de los estados que puede tener un proyectil misil al ser lanzado para el ataque.

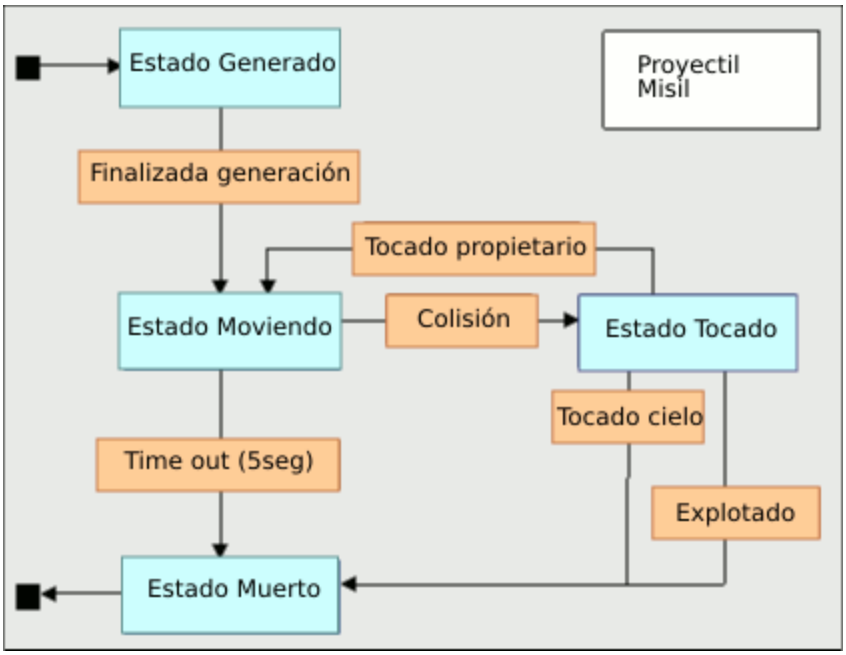


Figura 97. Representación de la transición de estados de un proyectil misil de Quake.

En la figura 97, la máquina de estados finitos ha sido representada usando un acercamiento muy similar al de un Diagrama de Transiciones de Estado. Las cajas azules son los estados, las naranjas los disparadores y las flechas son las transiciones de estado. Las cajas negras muestran el punto de entrada y de salida del sistema.

El diagrama muestra el ciclo de vida completo del misil en el juego. Es importante tomar en cuenta que el proyectil aparece como producto de una acción de otra Máquina de Estado Finita, concretamente la del "lanzador de misiles" desde su acción "disparar". Cuando la instancia del misil muere se elimina del juego y deja de existir.

Esta presentación es una implementación subjetiva del código. Otra

representación válida podría romper el "estado tocado" más profundamente en los estados de tocada y explosión. Personalmente veo la explosión como una acción o efecto realizado por el objeto misil en su estado tocado.

Otro aspecto importante es cuando el proyectil está en estado “tocado”, uno de sus efectos es intentar dañar a todo aquello que alcance su tacto. Si ha conseguido dañar otra entidad en el mundo del juego, la acción de dañar se convertirá en la entrada que activará algún cambio de estado en la entidad afectada.

Quake hace uso extensivo de las máquinas de estado finitas como mecanismos de control que gobiernan las entidades que existen en el mundo del juego.

Otro ejemplo que aplica el concepto de máquinas de estado finitas es el aplicado a un Shambler, que es una entidad o monstruo grande y malo del componente single player (un jugador) de Quake. Su misión en la vida es matar jugadores una vez que los ha localizado como se muestra en la figura 98.

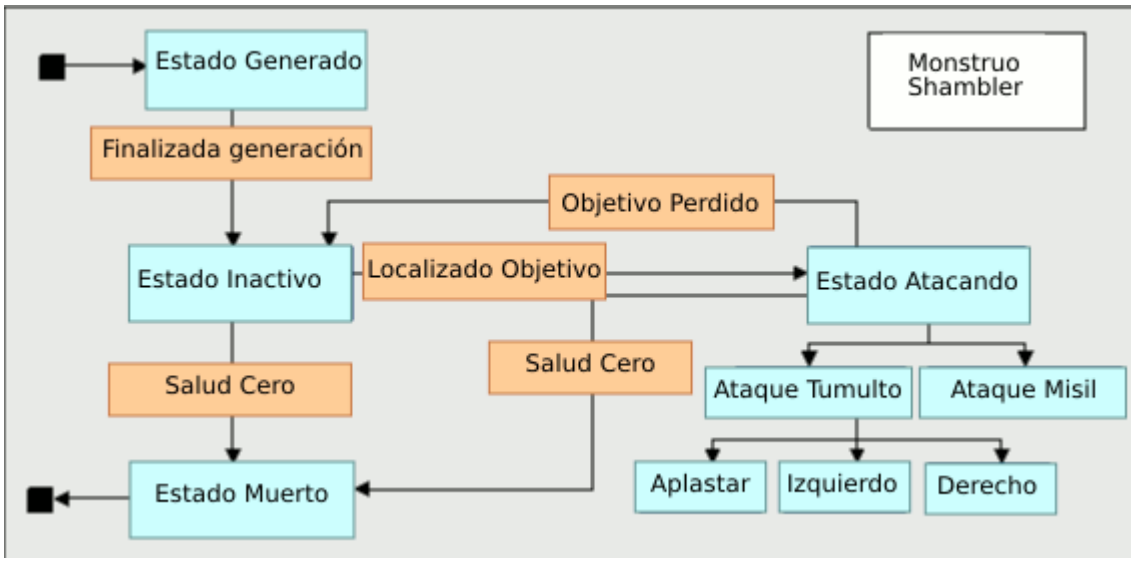


Figura 98. Transición de estados representando el monstruo Shambler de Quake.

De la misma forma que el misil, esta entidad tiene un estado inicial (estado engendrado), y el sistema finaliza cuando la entidad muera (estado muerto). Solo existen cuatro estados principales, pero el Shambler es un buen ejemplo que ilustra la capacidad de tener una jerarquía de sub-estados. Aunque los sub-estados de este ejemplo pueden ser considerados como acciones del "estado atacando", también son sub-estados ya que el monstruo solo puede realizar uno (o estar en uno) de ellos por cada ejecución del estado atacando.

Cuando en el estado es “atacando”, la instancia de Shambler toma una decisión basada en la evaluación de las entradas para realizar un ataque de estilo tumulto (cercano) o misil (larga distancia). Al seleccionar un ataque de estilo tumulto (estado atacando tumulto), las entradas se continúan evaluando con un número al azar para seleccionar el tipo de ataque tumulto (aplastar con los dos brazos, aplastar con el brazo izquierdo o abrazar con el brazo derecho).

El uso de un número aleatorio en la selección del sub-estado del ataque tumulto añade un nivel de impredecibilidad a la selección. Cada nivel en la jerarquía puede ser considerado como una sub-máquina de estados finitos de una entidad monstruo mayor, y en este caso la sub-FSM del estado ataque estilo tumulto puede ser clasificado como no-determinista.

Es importante comprender el uso de FSM en capas o jerarquía, porque cuando se usan, como se hace en el monstruo Shambler, permiten comportamientos mucho más complejos. Esta técnica es usada con mucha frecuencia en Quake por todas las entidades del mundo del juego.

Haciendo una simplificación del ejemplo se tiene en los activadores que causan la transición de estados. Cuando ocurre una transición de estado de "estado ataque" a "estado inactivo" el activador ha sido simplificado como "perdido objetivo". Es cierto que la transición ocurre por el hecho de perder la entidad objetivo, pero un objetivo puede ser perdido por el Shambler de diferentes

maneras que se evalúan en diferentes puntos del código, incluyendo un time-out de otra entidad.

Otro punto clave referente a este ejemplo es el uso de objetivos como principal motivador para la FSM. Esta técnica no ha sido discutida, aunque es un ejemplo del poder y flexibilidad de una FSM como técnica de control. Existe una jerarquía de máquinas de estados finitos, la FSM de más alto nivel se guía por el deseo de la entidad de localizar su objetivo y atacarlo. El objetivo es usualmente un jugador humano pero incluso puede ser un monstruo del mundo del juego. Debe tenerse en cuenta que mientras el monstruo está en el estado inactivo, mientras está simplemente vagando sigue buscando objetivos.

Quake no provee la mejor experiencia single player imaginable, pero es un juego divertido y adictivo, ambos atributos claves para el éxito de un juego. Es un buen ejemplo que muestra las capacidades tanto de máquinas de estados finitos muy simples como de FSM más complejas construidas con una jerarquía de FSM y motivadas por objetivos, como el monstruo Shambler (figura 99).

Figura 99. Monstruo Shambler.

CAPITULO IV

Motor gráfico e implementación en “Evolution”

4.1. Sprint 1

4.1.1 Creación del Product Backlog.

Scrum plantea el uso del *Product Backlog* en el cual se organizan y se planean las tareas a realizarse en un determinado *Sprint*. Dentro del documento se detallan las horas de trabajo y se determina el esfuerzo diario realizado por el equipo de trabajo de manera individual.

El primer paso es establecer los parámetros iniciales del *Sprint* como se muestra en la tabla *4*.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
1	10-nov-08	20	10

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis Codificación Prototipado Pruebas Reunión	Pendiente En curso Terminada Eliminada	Christian Natalia	

Tabla 4. Parámetros Iniciales

La tabla *4.* muestra ciertos datos relevantes del *Sprint*, como la fecha de inicio, número de días laborables y jornada.

Las tareas se detallan de acuerdo al tipo y estado, las cuales pueden darse dentro del desarrollo de un determinado *Sprint*. *Equipo*, describe a las personas involucradas en el desarrollo de las tareas. *Festivo*, muestra fechas especiales en las que no se trabaja.

El paso siguiente es crear una tabla describiendo cada tarea y detallando las horas de trabajo pendientes para cada día laborable del *Sprint* (*Tabla 5*)

SPRINT	INICIO	DURACIÓN
1	10-nov-08	20

	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X
	10-nov	11-nov	12-nov	13-nov	14-nov	15-nov	16-nov	17-nov	18-nov	19-nov	20-nov	21-nov	22-nov	23-nov	24-nov	25-nov	26-nov	27-nov	28-nov	29-nov	30-nov	1-dic	2-dic	3-dic
Tareas pendientes	14	14	14	13	13	13	13	13	13	12	11	11	11	11	9	9	8	8	8	7	7	6	4	
Horas de trabajo pendientes	200	190	180	170	160	150	150	140	130	120	110	100	90	90	80	70	60	50	40	30	30	20	10	

PILA DEL SPRINT				ESFUERZO																							
Backlog	Tarea	Tipo	Estado	Responsal																							
	Creación del Product Backlog	Análisis	En curso	Natalia	19	18	17	16	15	14	14	13	12	11	10	9	8	8	7	6	5	4	3	2	2	1		
	Concepto de Jugabilidad	Prototipado	Terminada	Natalia	8	3	1																				
	Implementación de Motor Grafico	Pruebas	Terminada	Christian	19	19	16	12	11	8	8	7	7	6	6	5	5	5	5	5	3	1	1	1	1			
	Carga de escenarios y Personajes	Análisis	Terminada	Christian	7	7	7	6	6	4	4	4	4	4	3	3	3	3	1	1								
	Ambientación	Codificación	Terminada	Natalia	11	9	7	6	5	5	5	4	2															
	Skybox	Codificación	Terminada	Natalia	9	9	9	9	7	5	5	2	2	1														
	Terreno	Codificación	Terminada	Christian	16	16	16	16	15	15	15	14	9	6	6	3	3	3										
	Agua	Codificación	Terminada	Christian	7	7	7	7	5	5	5	4	4	4	2	2	2	2										
	Animación	Codificación	Terminada	Christian	24	24	24	24	24	24	24	24	24	24	23	22	17	17	17	14	10	10	10	8	8	5	1	
	Rigging	Codificación	En curso	Christian	18	18	18	18	18	18	18	18	18	18	18	18	16	16	16	14	14	14	10	6	6	6	4	
	Skinning	Codificación	En curso	Christian	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	11	11	11	8	8	8	5	4	
	Exportación	Codificación	Pendiente	Christian	10	10	10	10	10	10	10	10	10	10	8	6	6	6	6	6	6	1	1					
	Estabilización y Pruebas	Codificación	Pendiente	Natalia	19	18	17	16	15	14	14	13	12	11	10	9	8	8	7	6	5	4	3	2	2	1		
	Toma de Resultados	Codificación	Pendiente	Natalia	20	19	18	17	16	15	15	14	13	12	11	10	9	9	8	7	6	5	4	3	3	2	1	

Tabla 5. Tareas Pendientes

	10-nov	11-nov	12-nov	13-nov	14-nov	15-nov	16-nov	17-nov	18-nov	19-nov	20-nov	21-nov	22-nov	23-nov	24-nov	25-nov	26-nov	27-nov	28-nov	29-nov	30-nov	1-dic	2-dic	3-dic
Christian	114	114	111	106	102	97	97	94	89	85	79	72	65	65	58	51	44	37	30	23	23	16	9	
Natalia	86	76	69	64	58	53	53	46	41	35	31	28	25	25	22	19	16	13	10	7	7	4	1	

Tabla 6. Horas Pendientes

La tabla *5*, detalla las tareas y las organiza según el tipo, estado y persona responsable. El esfuerzo se mide en base a las horas planteadas, las cuales disminuyen en el transcurso del tiempo. La tabla mostrada en la tabla *6*, señala el número de horas necesarias por cada día para cada integrante del equipo.

Una vez culminado el *Sprint uno*, se analiza el esfuerzo (*Figura 100*), el desarrollo de las tareas (*Figura 101*) y las horas empleadas por cada integrante del equipo de trabajo (*Figura 102*).

Proyecto
Evolution

SPRINT	INICIO	DÍAS
1	10-nov-08	20

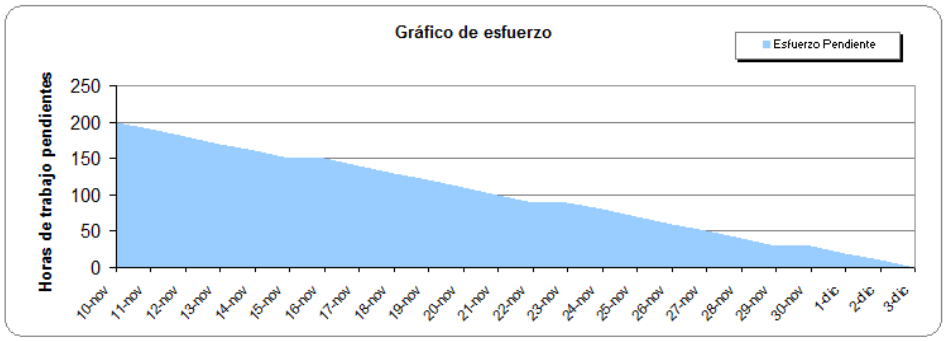


Figura 100. Esfuerzo

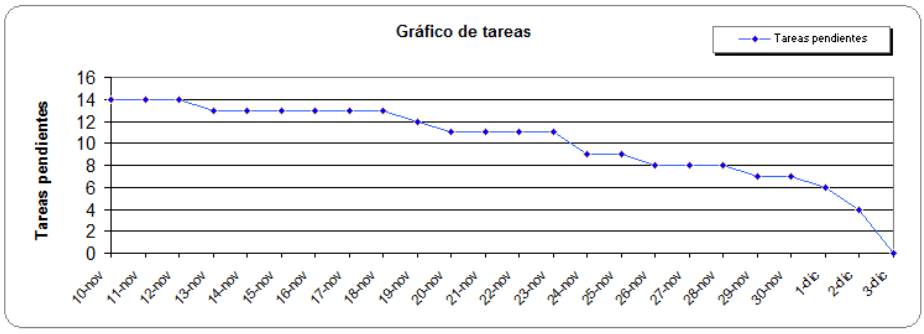


Figura 101. Tareas

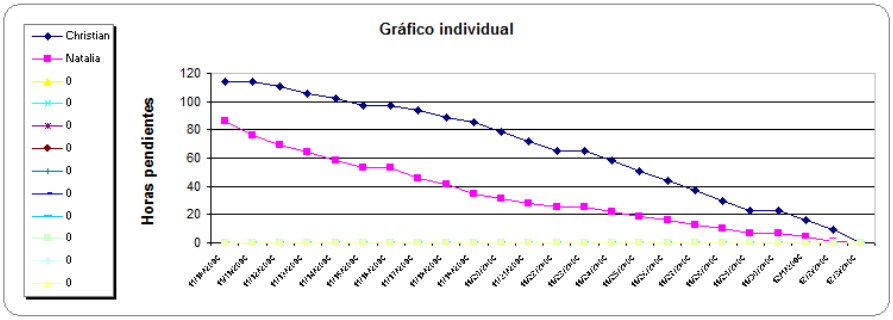


Figura 102. Horas Pendientes

4.1.2 Aplicación del concepto de la Jugabilidad en “Evolution”

La jugabilidad se plantea por el equipo de trabajo al inicio del proyecto, en esta etapa el grupo analiza el comportamiento del juego, detallando cada uno de sus componentes.

Es necesario el uso de bosquejos llamados *Story Boards,* para tener una noción general y básica del proyecto. La figura *103* muestra *e*l *Story Board* diseñado para *Evolution*.

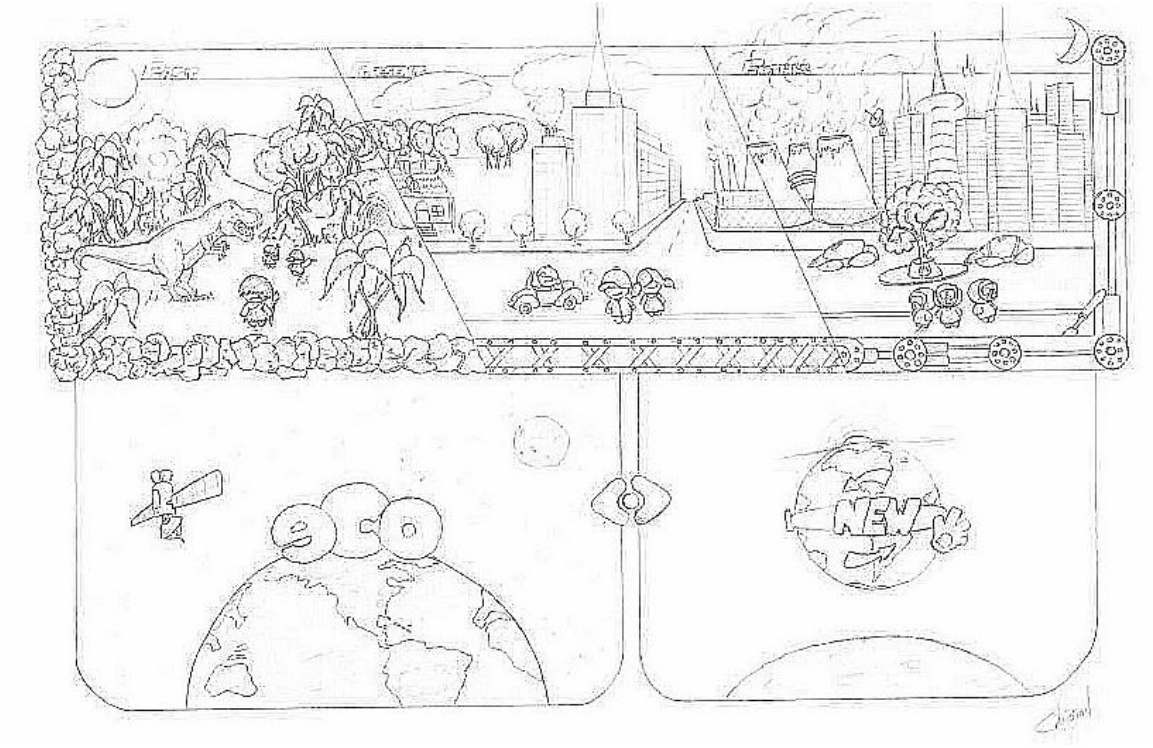


Figura 103. *Story Board*

La figura *103* muestra aspectos clave del juego como: el escenario, el manejo del menú principal, la interfase de usuario y el *Plot* principal de la historia. En cuanto al escenario, la historia se desenvuelve en una isla remota en medio de una jungla ambientada principalmente en la prehistoria (*Figura 104*).

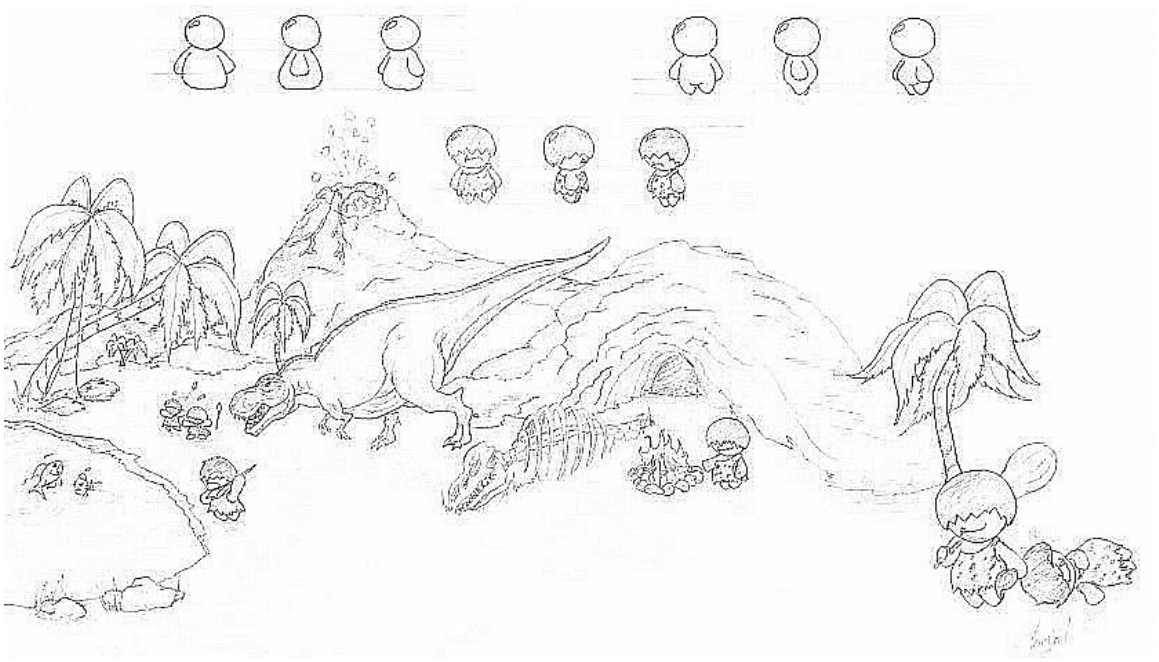


Figura 104. *Story Board*

El menú principal del juego tiene similitud con un sistema planetario el cual gira alrededor del sol (*Figura 105*). La interfase del juego o *GUI*, también es plasmada para una referencia posterior en el proceso de desarrollo (*Figura 105).*

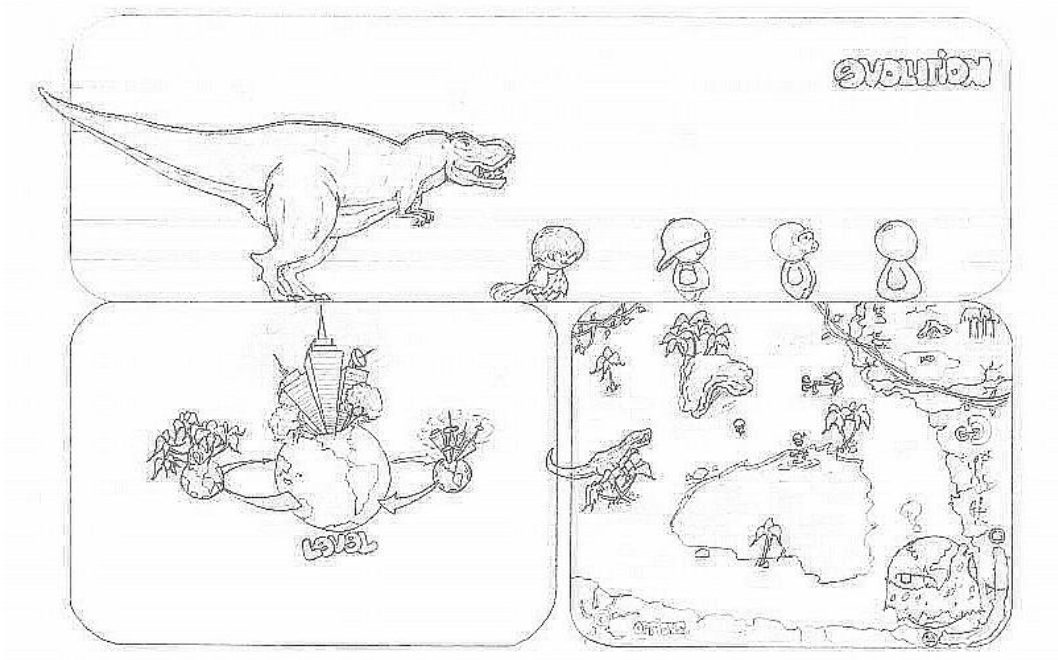


Figura 105. *Story Board*

Finalmente, el Plot principal se encuentra ilustrado mostrando la relación individuo – naturaleza. Todo lo expuesto debe estar plasmado en papel o computador ya que las ideas deben estar muy claras antes de empezar, de tal forma que todo el equipo trabaje hacia un objetivo común.

4.1.3 Implementación del Motor Gráfico dentro de XNA

El proceso comienza con el manejo y familiarización de la herramienta o motor gráfico. En este caso se manejan dos Motores Gráficos de Microsoft.

- XNA
- TORQUE

Tanto XNA como Torque tienen la misma orientación hacia el desarrollo de los videojuegos, pero difieren en su costo, herramientas y soporte.
XNA por una parte, es de libre distribución mientras que Torque tiene un precio; por otro lado XNA es usado principalmente en aplicaciones 2D y Torque es exclusivamente 3D. XNA puede ser usado para crear aplicaciones Windows y Xbox 360, en cambio Torque puede crear aplicaciones para PC, Xbox 360, PlayStation 3 y Wii.

XNA, es útil para aprender a crear proyectos orientados al mundo de los videojuegos y es muy recomendable empezar con esta herramienta antes de trabajar con Torque, debido a que posee funciones avanzadas aunque tienen un nivel de aprendizaje alto tanto a nivel gráfico como a nivel de codificación. De igual forma, Torque posee herramientas que facilitan cierto tipo de tareas como la integración de escenarios y personajes.

Evolution será integrado en ambos motores gráficos, con el objetivo de analizar cuál ofrece mayores ventajas para crear aplicaciones 3D de calidad y que puedan competir con otros productos existentes en el mercado.

4.1.4 Carga de escenarios y Personajes

La carga de escenarios y Personajes contiene los pasos iniciales para la creación de un videojuego, los mismos que son fundamentales para los ítems subsiguientes. Principalmente está relacionado con el ambiente en el cual el juego tomará lugar. Una vez terminado el equipo de trabajo puede crear objetos y animaciones que tengan características similares que puedan ser integrados sin dificultad.

Ambientación

La ambientación trata acerca del diseño, modelado e implementación del entorno visual del juego, el proceso involucra elementos como: terreno, luz, partículas, así como otros elementos relacionados que posteriormente serán simulados por el motor gráfico. En este caso en particular, la ambientación del juego será desarrollada dentro de una isla remota, por lo tanto, el entorno visual será conformado principalmente por: terreno (isla), luz (sol) y agua (océano).

Este proceso adquiere gran importancia al inicio del proyecto, ya que este permite a los desarrolladores crear e importar objetos que se adapten a las necesidades del juego en base al ambiente, de tal forma que el resultado sea satisfactorio.

Antes de empezar, los desarrolladores optan por diseños conceptuales del entorno, los cuales son obtenidos de (story boards) previamente elaborados. Una vez que el equipo de trabajo tiene una noción básica del entorno, se procede a crear cada uno de los elementos ambientales del juego.

En cuanto a Evolution se tiene planeada la creación de los siguientes elementos:

- Skybox
- Terreno
- Agua

- Partículas

Skybox

Skybox o caja atmosférica, tiene como objetivo, encerrar la escena principal del juego, con una figura básica (en este caso un cubo), el cual se encuentra texturizado internamente con imágenes de ambientes exteriores en sus cuatro planos. Cada plano muestra una imagen diferente como por ejemplo: nubes en el plano superior y terreno en el plano inferior. En la figura *106* se encuentran las imágenes a ser utilizadas así como el resultado dentro del motor gráfico.

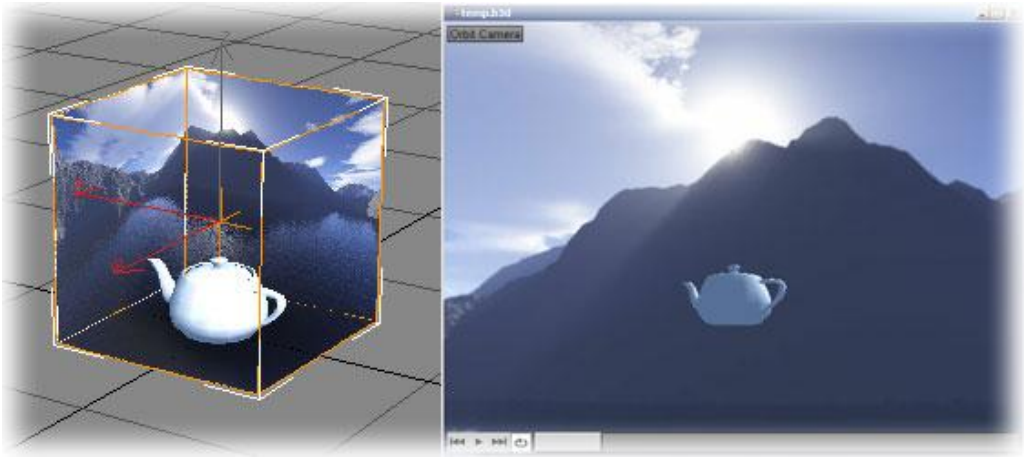


Figura 106. *Skybox*

Para crear un *Skybox*, se requiere el uso de una herramienta de diseño 2D, la cual permitirá juntar imágenes para cada lado o cara del cubo como se muestra en la figura 107.

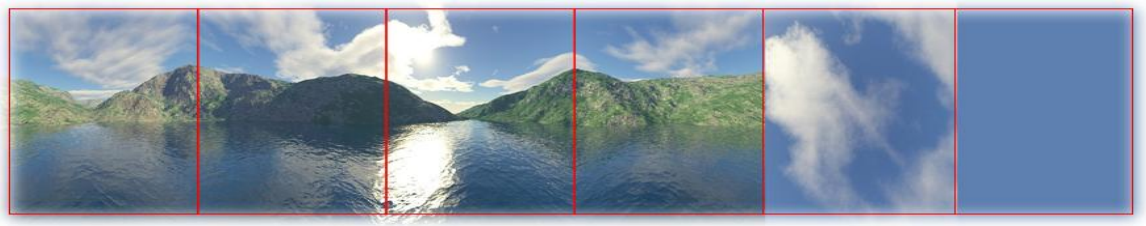


Figura 107. *Imagen Skybox*

Terreno

El terreno, básicamente es la estructura base en donde los elementos u objetos van a ser situados para el desarrollo del videojuego. Existen varios métodos y

programas que permiten generar terrenos, los cuales difieren principalmente entre calidad, precisión y desempeño.

Dentro del proyecto se plantean y se analizan dos alternativas para la creación del terreno. La primera es de simple creación e implementación para personas con poca o nula experiencia y la segunda es de uso profesional con opciones avanzadas para personas con un conocimiento medio-alto ya que ofrece resultados visiblemente superiores.

Método Simple.

Mediante el uso de una herramienta de diseño 2D se puede conseguir el efecto deseado en el motor gráfico, para lo cual es necesario el uso de herramientas básicas de pintura y difuminación como se ejemplifica en la figura 108.

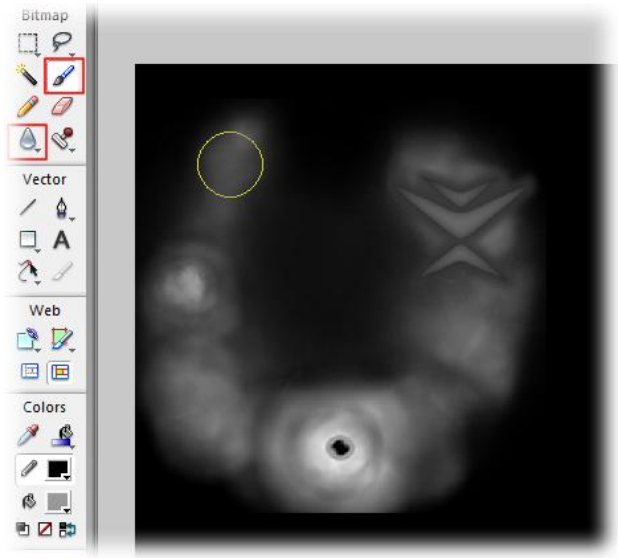


Figura 108. *Height Map*

Height map o mapa de altura es una imagen en blanco y negro que tiene el objetivo de recrear la superficie terrestre, donde los colores blancos denotan altura mientras que el color negro denota profundidad. Como se aprecia en la figura 108 mediante el uso de difuminación y brochas circulares en el editor gráfico 2D, es posible crear montañas y un volcán principal. Aparentemente la imagen tiene semejanza con una nube pero el motor gráfico transforma la imagen en terreno como se muestra en la figura *109.*

Figura 109. *Height Map Implementado*

Dentro del motor gráfico es necesario crear una función que permita crear el height map en base a la imagen terminada. Como se muestra en el código siguiente, escrito en lenguaje C#:

```
public HeightMapInfo(float[,] heights, float terrainScale)
{
    if (heights == null)
    {
        throw new ArgumentNullException("heights");
    }
    this.terrainScale = terrainScale;
    this.heights = heights;
    heightmapWidth = (heights.GetLength(0) - 1) * terrainScale;
    heightmapHeight = (heights.GetLength(1) - 1) * terrainScale;
    heightmapPosition.X = -(heights.GetLength(0) - 1) / 2 * terrainScale;
    heightmapPosition.Z = -(heights.GetLength(1) - 1) / 2 * terrainScale;
}
```

En la función se especifican valores como posición *heightmapPosition.X* y altura o relieve *heightmapWidth*, los cuales permiten crear la altura y dimensión deseada en las montañas dentro del juego.

Método Óptimo (Profesional)

Este método involucra el uso de una herramienta profesional orientada a la creación de superficies terrestres. El programa *L3DT 2.6 Pro*, permite a desarrolladores de videojuegos generar y texturizar superficies que posteriormente serán integradas en diversos motores gráficos. *L3DT* maneja varios formatos como *map* y *atlas*, siendo *atlas* el formato apropiado que será implementado.

L3DT soporta modos de operación automática, manual y de diseño. Cualquier modo brinda beneficios particulares, pero cada uno tiene características de acuerdo a la capacidad técnica del equipo. El modo automático crea terrenos indiferentemente dando parámetros generales como: Altura, nivel del mar, tamaño, precisión y tipo de terreno. En el modo manual y de diseño, se describe y detalla todos los parámetros desde la forma hasta el color del terreno.

Ya que *Evolution* esta ambientado dentro de una isla la cual posee un volcán principal, el uso del modo manual es recomendado. Para empezar, el programa solicita información de tamaño tanto del terreno como del entorno de trabajo como se muestra en la figura *110.*

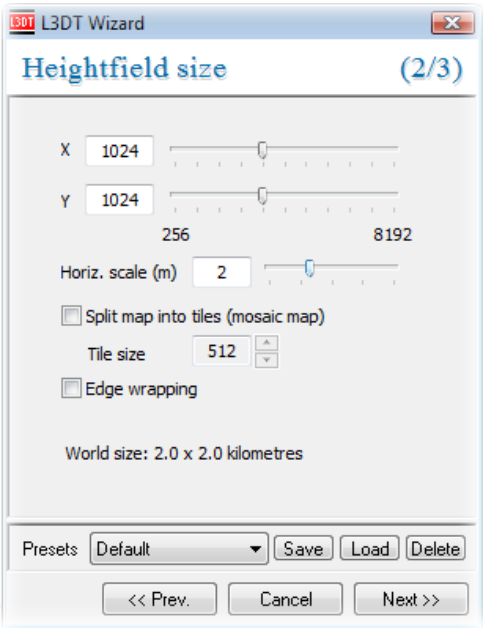


Figura 110. ***Configuración Inicial L3DT***

El tamaño del terreno debe ser múltiplo de dos, en este caso 1024x1024, por otro lado se determina el valor de escala horizontal (2) el cual es fundamental de acuerdo al tipo de juego que se desea desarrollar, ya que determina el nivel de detalle del terreno. Los siguientes valores van de acuerdo al tipo de juego:

- Juego en Primera persona Valor: 1 – 2
- Juego estrategia Valor: 3 – 4
- Juego simulador de vuelo Valor: 5 – en adelante

Evolution tiene por objetivo ser en primera y tercera persona, por lo que se da el valor de 2 para obtener un buen nivel de detalle final. El siguiente paso es dibujar el terreno en el mapa de diseño con ayuda del cuadro de edición (*figura 111*). Este provee herramientas de generación de altitud, ubicación de ríos, océano, y texturas.

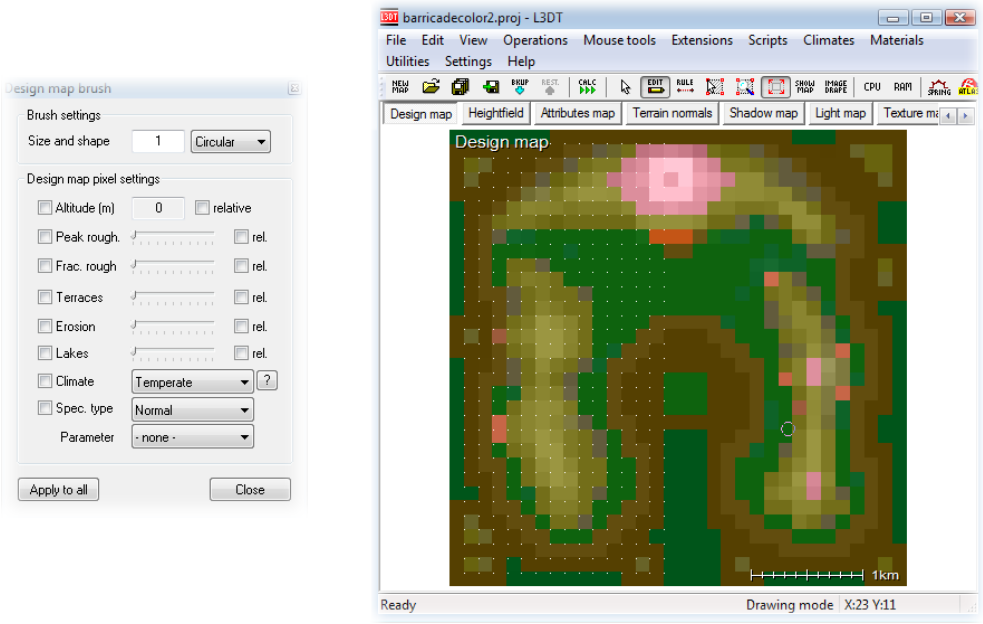


Figura 111. *Mapa de Diseño*

Una vez que se obtengan los resultados esperados, se calcula el *Height Map* del terreno (*Figura 112*).

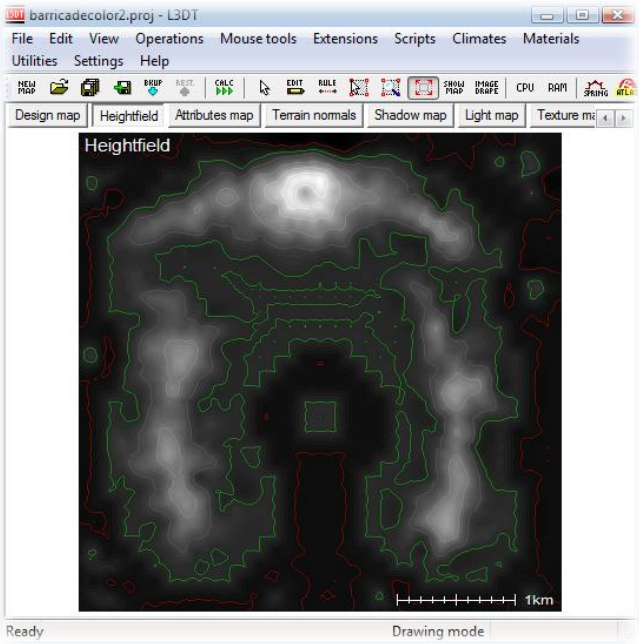
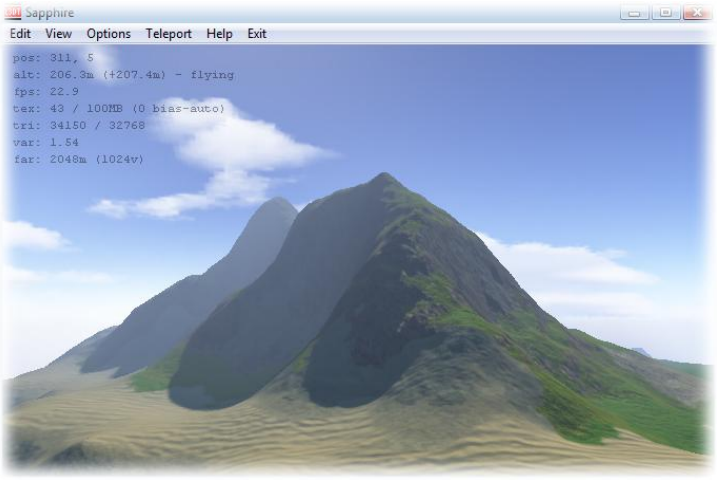


Figura 112. *Heightfield*

Con el *height map* obtenido es posible generar un ejemplo tridimensional que ayuda a verificar que todo este de acuerdo a lo planeado (*figura 113*).



,

Figura 113. *Previsualización*

Finalmente se procede a generar el mapa con la iluminación correcta para definir sombras y texturas a ser usadas por el motor gráfico. En este caso es necesario activar las opciones:

- *Attributes map*: o mapa de atributos, en el cual se especifican valores como largo, ancho y detalle.
- *Terrain normals:* o normales de terreno, permite ajustar valores de brillo y contraste del mapa.
- *Light map*: o mapa de luz, simula el sol en una posición en particular que ayuda a generar sombras.
- *Texture map*: o mapa de textura, crea las texturas dependiendo del tipo de terreno a ser utilizado.

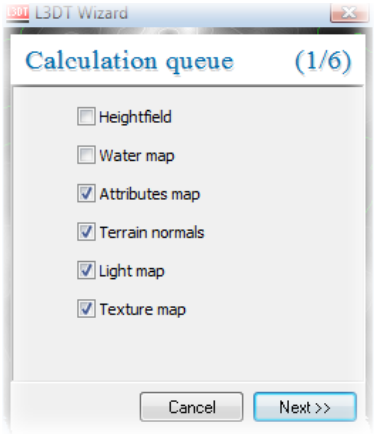


Figura 114. *Cálculo de Atributos*

Los valores señalados se presentan en la figura 114; Water map o mapa de agua, no es utilizado ya que el agua va a ser generada dentro del motor gráfico.

El proceso de renderización o generación de terreno toma un tiempo significativo, de acuerdo a la capacidad de procesamiento del computador. El tiempo aproximado que lleva este proceso, es de 8 horas, tiempo que puede variar dependiendo de la calidad y tamaño de las texturas utilizadas por el usuario.

Una vez concluido el proceso se genera el tipo de archivo *atlas*, el cual va a ser importado por el motor gráfico. El motor gráfico ya incluye el código necesario para integrar archivos tipo *atlas* pero deben cumplir con ciertos parámetros antes expuestos. Por lo cual el código para su implementación es el siguiente:

```
New AtlasInstance (AtlasTerrain) {
        CanSaveDynamicFields = "1";
        Enabled = "1";
        Position = "0 0 0";
        Rotation = "1 0 0 0";
        Scale = "1 1 1";
        DetailTex = "scriptsAndAssets/data/terrains/details/detail4.jpg";
        AtlasFile = "scriptsAndAssets/data/terrains/Atlas4/evo.atlas";
        LightmapDimension = "0";
    };
```

En el código se especifican valores como: ubicación, rotación escala, detalle y archivo *atlas*, que posteriormente es exportado por el motor gráfico al momento de cargar objetos en escena. El resultado final se muestra en la figura *115.*

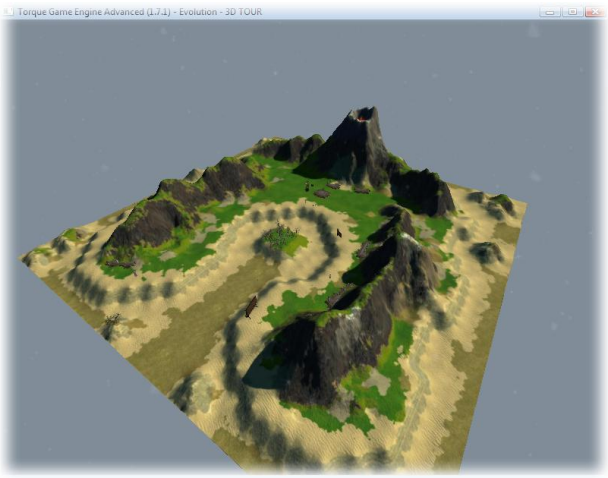


Figura 115. ***Terreno Implementado***

Agua

En el caso de Evolution, el agua es al igual que el terreno, un elemento que rodea el escenario del juego.

Existen varias técnicas para generar o simular agua en el medio ambiente, las cuales van de acuerdo a la capacidad gráfica del motor así como de la habilidad del equipo de trabajo. Para el juego se plantean dos alternativas, la primera es muy simple y se logra mediante el uso de texturas, en una superficie que puede ser multiplicada o replicada varias veces en un mismo plano para conseguir un

efecto de océano. Este tipo de texturas ayudan a no desperdiciar espacio en disco ni abusar del cálculo gráfico, ya que con una textura de tamaño (512 x 512) es posible crear una imagen de dimensiones indefinidas como en la figura *116*.

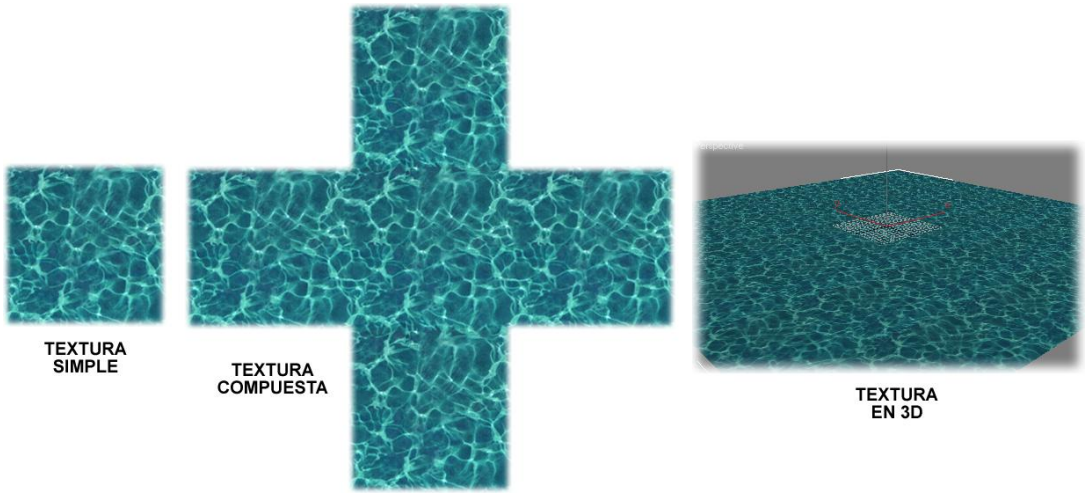


Figura 116. *Textura Replicable*

La segunda alternativa implementa cálculos matemáticos así como efectos de luz y sombras *shaders*, los cuales hacen lucir aun más realista al agua generada en el motor. El código usado es el siguiente:

```
New WaterBlock (WaterBlockObject) {
        CanSaveDynamicFields = "1";
        Enabled = "1";
        Position = "1095.42 1010.32 -353";
        Rotation = "1 0 0 0";
        Scale = "40960 40960 450";
        WaveDir [0] = "0 1";
        WaveDir [1] = "0.707 0.707";
        WaveDir [2] = "0.5 0.86";
        WaveDir [3] = "0.86 8.5";
        WaveSpeed [0] = "-0.065";
        WaveSpeed [1] = "0.09";
        WaveSpeed [2] = "0.04";
        WaveSpeed [3] = "0.025";
        WaveTexScale [0] = "7.14 7.14";
```

```
    WaveTexScale [1] = "6.25 12.5";
    WaveTexScale [2] = "50 50";
    WaveTexScale [3] = "0 0";
    ReflectTexSize = "1024";
    BaseColor = "60 57 53 155";
    UnderwaterColor = "30 70 90 190";
    GridSize = "205";
    SurfMaterial [0] = "Water";
    SurfMaterial [1] = "Underwater";
    SurfMaterial [2] = "Water1_1FogPass";
    SurfMaterial [3] = "WaterBlendMat";
    SurfMaterial [4] = "WaterFallback1_1";
    FullReflect = "1";
    Dlarity = "1";
    FresnelBias = "0.12";
    FresnelPower = "8";
    FisibilityDepth = "15";
    RenderFogMesh = "1";
      ReanderShadow = "1";
      SunEnabled = "1";
      SunMaterial = "WaterSunReflection";
      SunPosition = "100 100";
      SunSize = "50 50";
      UnderWaterDistortionScale = "0.015";
  };
```

En este código se tienen parámetros de ubicación, rotación y escala. Adicionalmente, se tienen parámetros como (waveDir[0] = "0 1";), el cual indica la dirección de una ola simulada con (shaders); (waveSpeed[1] = "0.09";) simula la velocidad con la que la ola se genera; (surfMaterial[0] = "Water";) es la textura del mar; (fullReflect = "1";) un efecto especial que refleja el exterior del mar; (visibilityDepth = "15";) simula el efecto de profundidad del mar y (sunEnabled = "1";),(sunMaterial = "WaterSunReflection";),(sunPosition = "100 100";) y (sunSize = "50 50";), simulan el reflejo del sol en el agua. Finalmente,

(underWaterDistortionScale = "0.015";), crea una ilusión de distorsión dentro del agua.

Una vez terminado el código el resultado es mostrado en la figura *117.*

Figura 117. Agua Implementada

4.1.5 Animación

Rigging (Creación de Huesos).

Rigging, se refiere a la creación y colocación de huesos previo a la animación de los personajes. Para empezar el proceso de animación, se debe tener finalizado el modelo a ser animado. Es decir es necesario tener un único elemento, el cual no puede estar compuesto por elementos independientes. También se debe tomar en cuenta que el modelo, debe estar completamente texturizado como ya fue expuesto en pasos previos, de manera que si fuese necesario algún tipo de modificación, solo se proceda a modificar el archivo tipo imagen *JPG* para no tener problemas con el modelo 3D.

El modelo 3D final obtenido de un habitante, se muestra en la figura 118:

Figura 118. Modelo o Personaje

El modelo 3D del habitante se encuentra totalmente terminado como elemento independiente y totalmente texturizado. Otro punto muy importante a la hora de crear los huesos para el personaje es crear el personaje con todas sus extremidades extendidas como se puede apreciar en la figura, debido que facilita el movimiento del personaje sin comprometer su fluidez en la animación.

Para empezar, el modelo a ser usado tiene que estar completamente terminado (Figura 119).

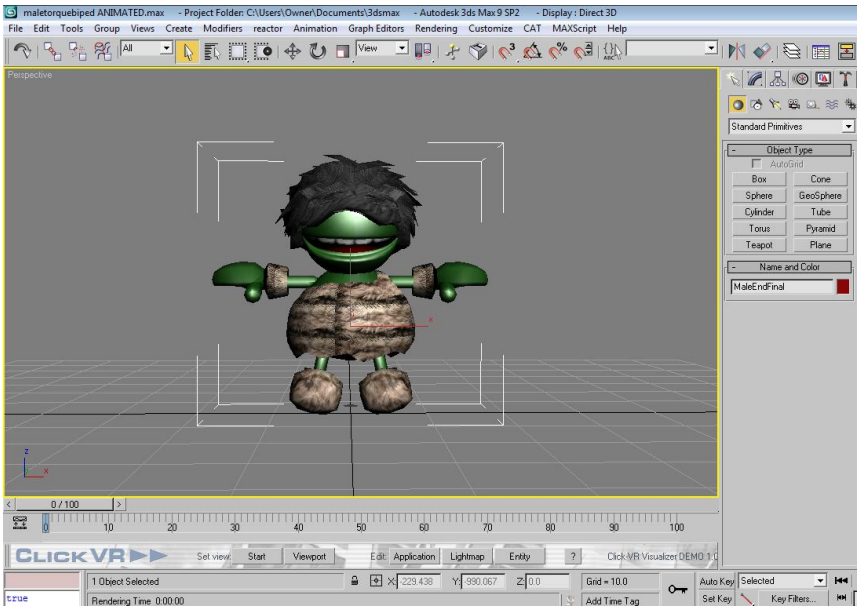


Figura 119. *Ambiente 3DMAX*.

Una vez listo el modelo, es necesario organizar las capas *Layers* que se van a usar para crear el esqueleto del habitante de tal manera que cada elemento que forma parte de la animación resultante se encuentre debidamente organizado (*Figura 120*).

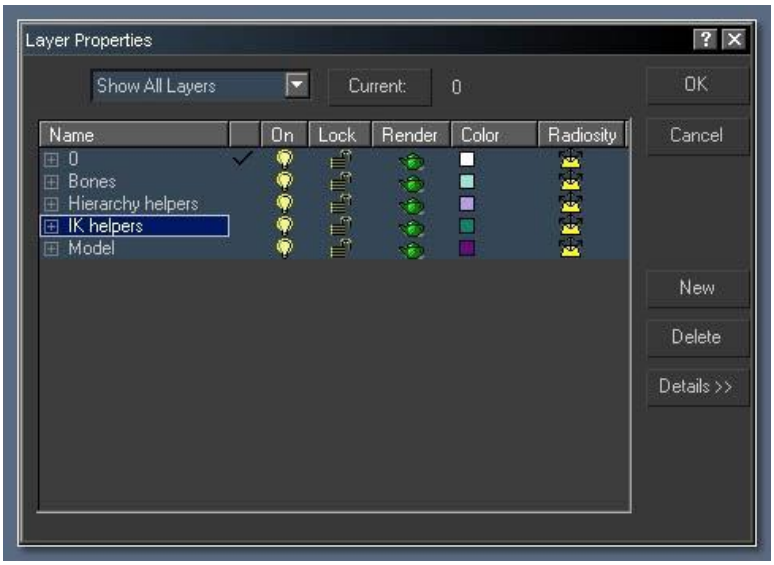


Figura 120. Elementos en Escena

En la figura 120 las capas se encuentran determinadas de la siguiente manera:

- Bones
- Hierarchy Helpers
- IK Helpers
- Model

Bones, se refiere a la estructura interna del personaje o los huesos q lo forman. *Hierarchy Helpers*, define las herencias de los huesos padres hacia los huesos hijos. *IK Helpers*, son las ayudas externas para poder animar sin dificultad al personaje. *Model*, es el modelo del personaje actual.

Ahora se debe tener en cuenta que para un óptimo funcionamiento en el motor gráfico *Game Engine*, el número de huesos aplicados en el personaje debe ser mínimo para mejorar su rendimiento, es decir, para consumir menos recursos del computador. Para lo cual se debe previamente analizar los puntos clave de movimiento como es en este caso: los brazos, las piernas y la cabeza.

Se tienen dos formas para animar un personaje:

- La primera, es crear los huesos uno a la vez y conectarlos de acuerdo a su jerarquía, permitiendo crear libremente cualquier estructura. En este caso es más difícil determinar parámetros como el movimiento de brazos y piernas es decir limitar a que los brazos no sobrepasen los codos o que una mano no gire 360 grados simulando el comportamiento normal de un brazo. Todo esto es determinado o controlado por ayudas externas para que el personaje luzca real a la hora de animar.

- La segunda, es mediante el uso de un Bípedo *Biped*, el cual es una estructura de huesos similares a las de un ser humano donde se predefine el número de huesos para luego generar automáticamente su estructura. Se debe tomar en cuenta que este método sirve exclusivamente para caracteres de forma humana.

Para cualquiera de los dos procesos mencionados hay que tener clara la estructura jerárquica de los huesos y así mismo se debe tener en cuenta que debe existir un solo hueso padre que contenga al resto de huesos hijos.

En esta sección se analizará los dos procesos así como sus ventajas y desventajas.

Creación de Huesos (Libre)

Se sitúa al modelo en el centro de coordenadas para tener un mejor control y para exportar el modelo de la forma más conveniente para el motor gráfico (*Figura 121*).

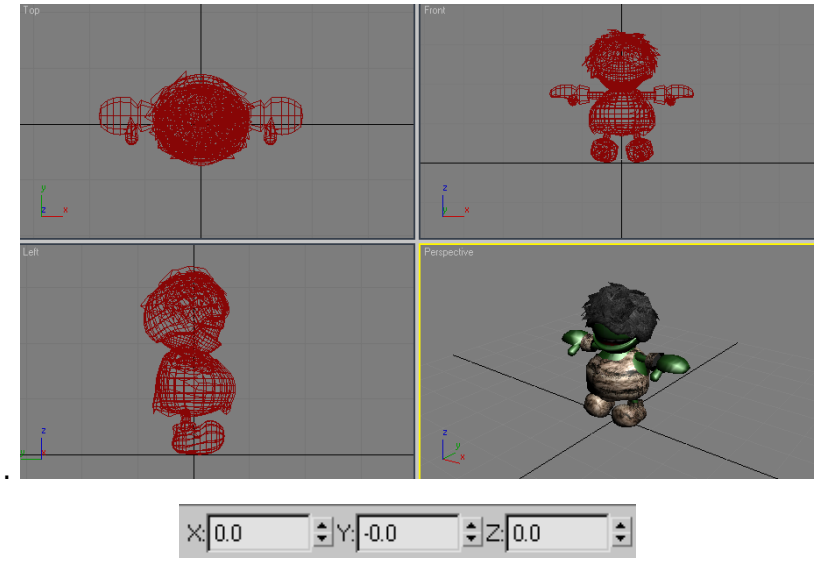


Figura 121. ***Preparación del Personaje***

Una vez posicionado el personaje, se puede proceder a la creación de los huesos (*Figura 122*).

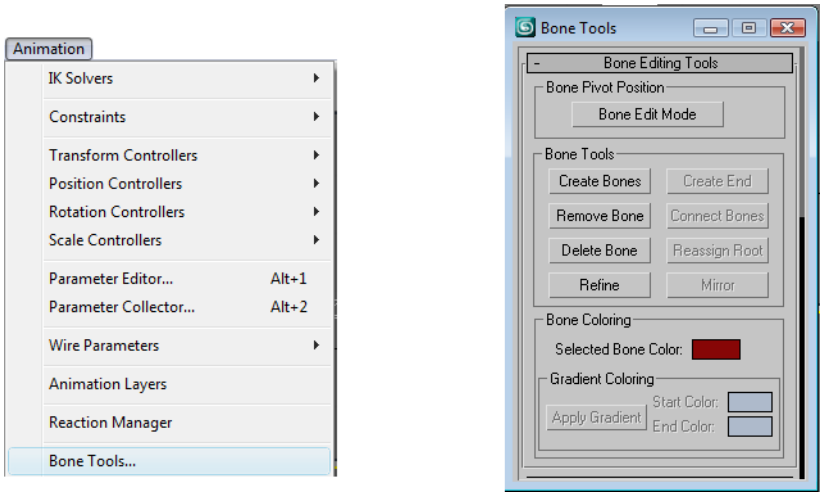


Figura 122. ***Herramientas de Creación***

Primero se configuran algunos parámetros del personaje dentro del panel de propiedades *Object Properties*, seleccionando las opciones:

- Freeze: La cual bloquea al personaje para no moverlo accidentalmente al momento de colocar los huesos.

- See Through: El cual permite ver a través del personaje para colocar los huesos internamente (*Figura 123*).

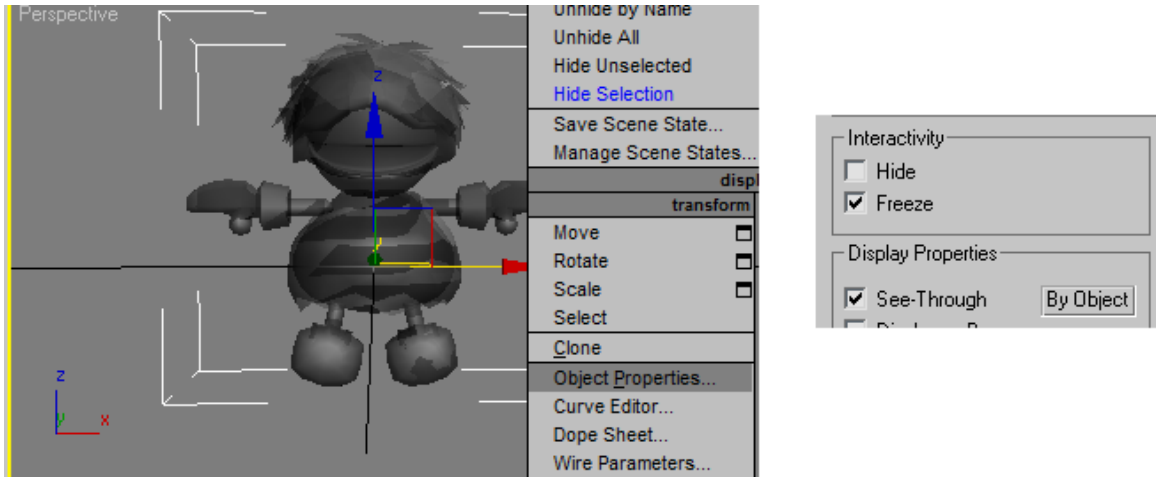


Figura 123. Propiedades

Estos parámetros permiten al usuario trabajar con fluidez a la hora de la creación de huesos. Al momento de crear huesos se debe tener en cuenta que el exportador puede causar problemas cuando existen varios huesos en escena por lo cual se recomienda usar el menor número posible.

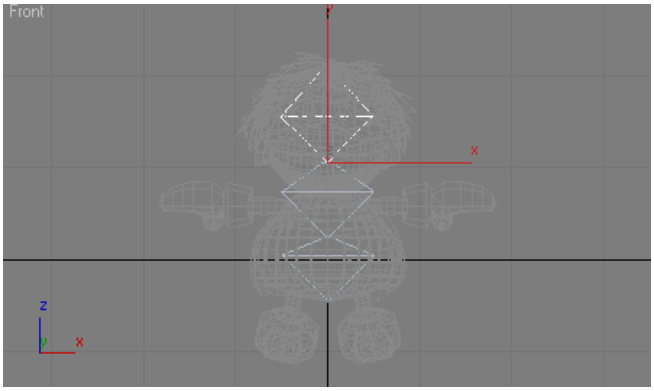


Figura 124. Creación de Huesos

Al crear huesos, éstos no tienen una forma coherente con su estructura, como se muestra en la figura *124*, Por lo cual se modifican ciertos parámetros para ajustar su tamaño y forma (*Figura 125*).

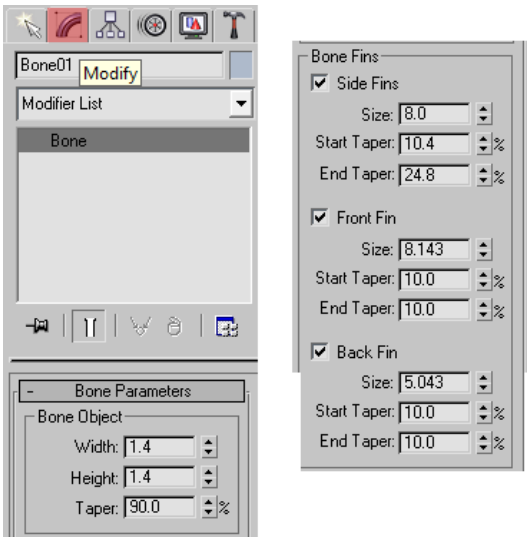


Figura 125. Propiedades de Ajuste

Las propiedades son las siguientes:

- Width
- Height
- Taper

Donde *Width* y *Height* permiten modificar el espesor del hueso mientras que *taper* determina el grosor de la base del hueso. Otras propiedades como *Bone Fins* permiten expandir frontal, posterior y lateralmente los huesos para rellenar en lo posible al personaje (*Figura 126).*

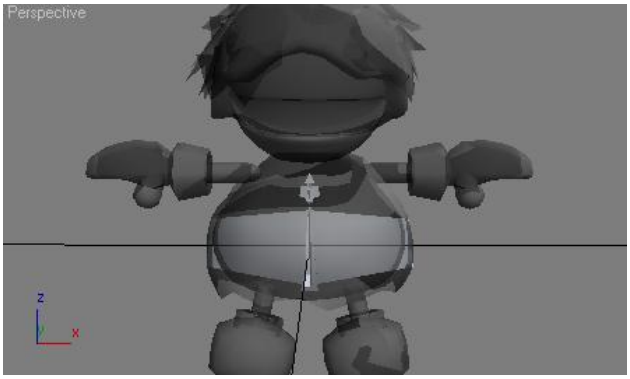


Figura 126. Ajustes

Para brazos y piernas no es posible clonar ni copiar los huesos de un brazo al otro o de una pierna a la otra ya que al momento de clonar o copiar existen errores que pueden producir efectos inesperados, como girar incorrectamente.

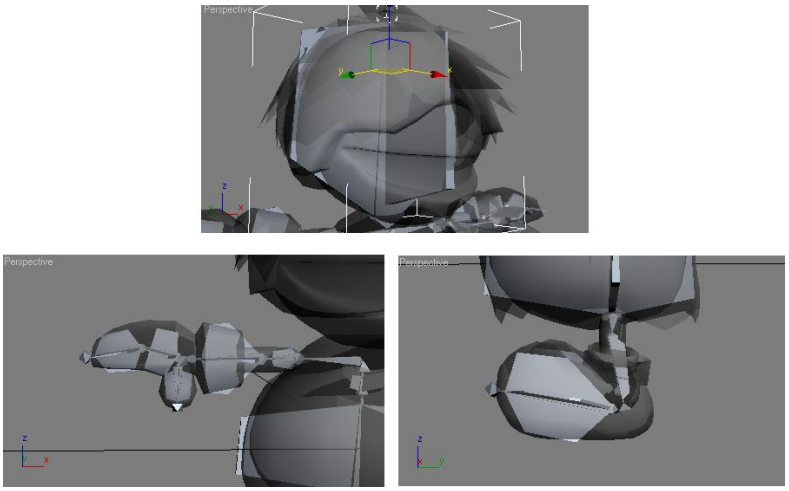


Figura 127. Creación de Huesos.

Se repiten los pasos para cada sección del cuerpo (*Figura 127*).

Dado que cada parte del cuerpo fue creada independientemente (columna, brazo izquierdo, brazo derecho, pierna izquierda, pierna derecha, cabeza), es necesario interconectar los huesos para definir jerarquías. Para lo cual dentro de la opción *Schematic View* o vista esquemática, es posible manipular las jerarquías de los huesos creados (*Figura 128*).

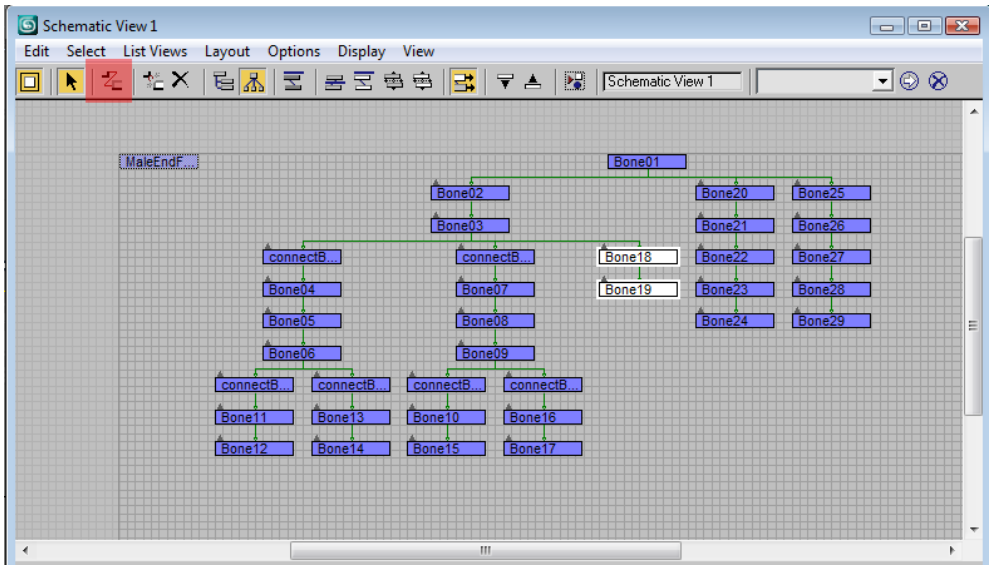


Figura 128. Jerarquías.

Una vez que los huesos del personaje tienen las jerarquías correspondientes, es necesario implementar *helpers* o ayudantes que faciliten el proceso de animación. Los *IK Solvers* (*Figura 129*) limitan el movimiento de los huesos, con

lo cual es posible simular el movimiento de brazos y piernas limitados por codos y rodillas respectivamente.

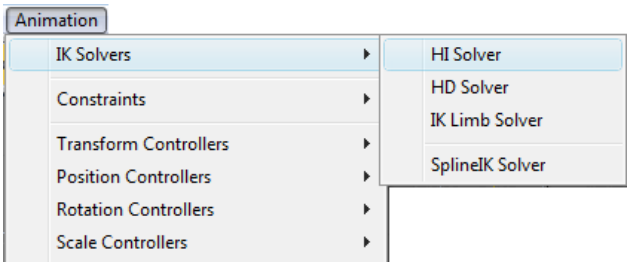


Figura 129. IK Solvers

Este proceso se lo repite en todas las extremidades del personaje, obteniendo el siguiente resultado (*Figura 130*).

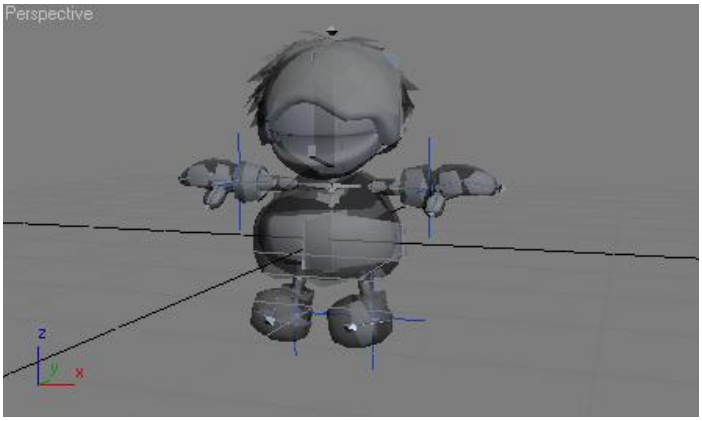


Figura 130. IK Solvers

Existen otro tipo de ayudantes llamados *Point* que facilitan la animación tanto de la cabeza como del cuerpo ya que solo es posible animar a los ayudantes más no a los huesos (*Figura 131*).

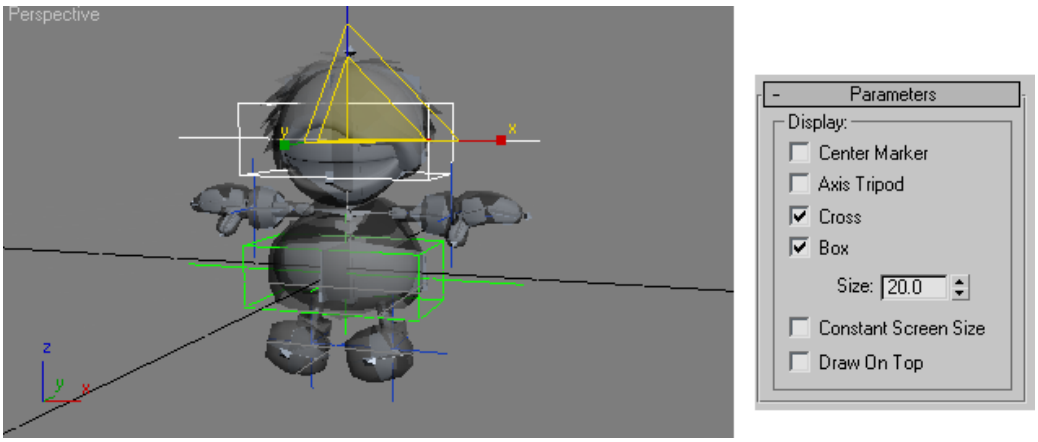


Figura 131. Pint Helpers

Una vez concluido este proceso, se obtiene el siguiente resultado (*Figura 132*).

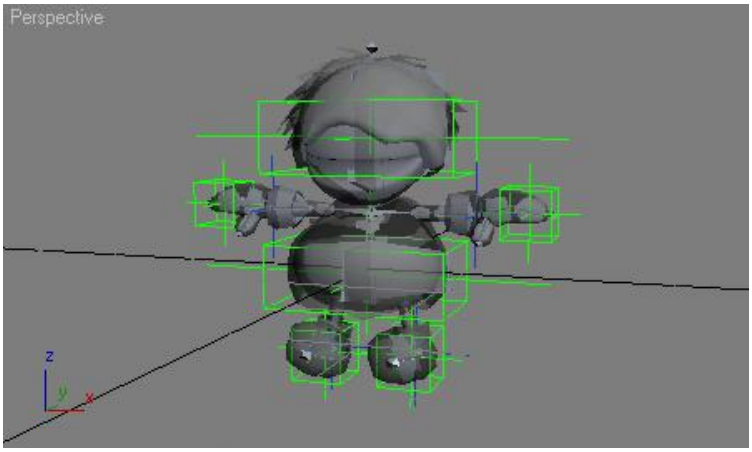


Figura 132. Ayudantes

Finalmente el proceso de creación de huesos termina. En este punto el usuario será capaz de mover libremente la estructura interna del personaje pero este movimiento no afectará al mismo. El proceso varía entre personajes debido a su movimiento. Como es en el caso de animales, criaturas alienígenas, dinosaurios, etc.

Creación de Huesos (Biped)

Biped, es otra forma de crear huesos para un personaje mucho más simple que en el proceso anterior. Hay que tener en cuenta que un *Biped* solo puede ser

usado para personajes de forma humana, es decir de dos piernas y dos brazos, de ahí proviene el nombre *biped* o bípedo. Para personajes como animales o criaturas extrañas, este proceso es el más utilizado.

Al igual que en el anterior proceso, se debe bloquear al personaje y hacerlo transparente para mayor facilidad en su implementación. Una vez que se configura al personaje, la estructura *biped* es creada en escena (*Figura 133*).

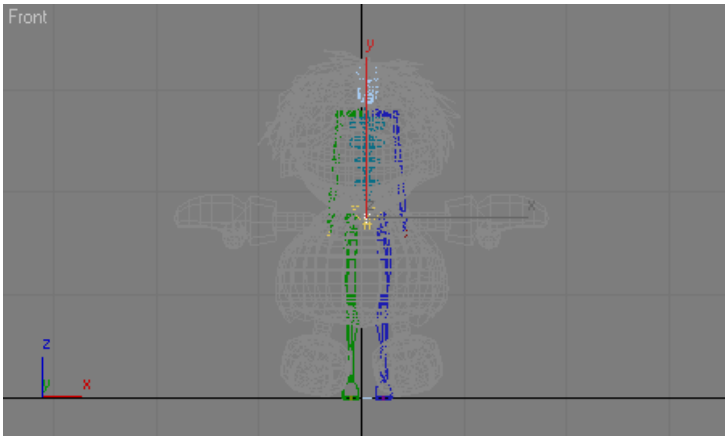


Figura 133. Biped

Es posible editar el esqueleto para ajustarlo al personaje (*Figura 134*).

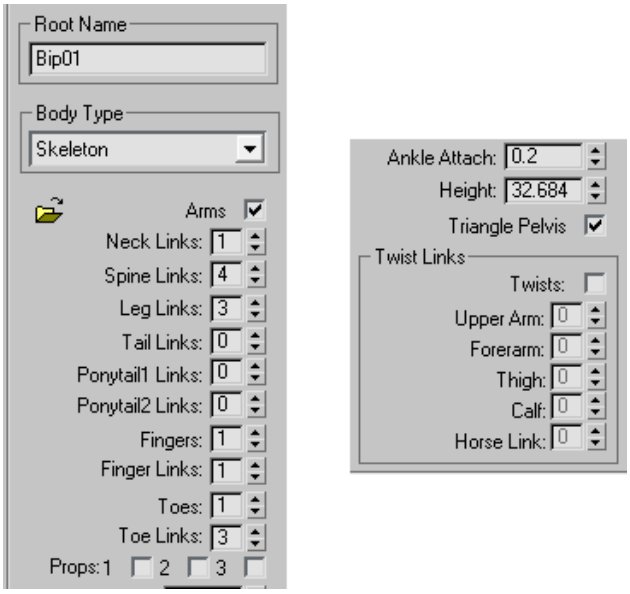


Figura 134. Panel de Propiedades

En el panel es posible editar propiedades como:

- *Root Name*: Nombre del esquelo, en este caso *Bipedmale*,
- *Spine Links*, *Fingers, toes*: Número de huesos de la columna dedos y pies.

En este punto se define el número de huesos para el personaje el cual debe ser mínimo si se lo va a usar dentro del motor gráfico.

Cuando el usuario está editando la posición de los huesos, también ajusta el sistema *Biped* al personaje. La figura *135* muestra el sistema Biped integrado al modelo.

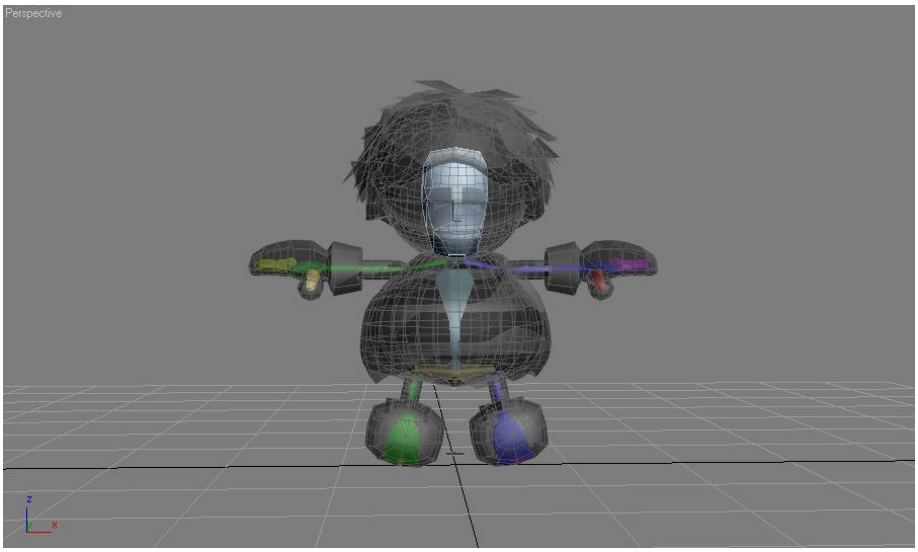


Figura 135. Biped Final

El esqueleto presenta ciertas ventajas como la jerarquización correcta de cada hueso (*Figura 136*), así como el movimiento limitado en piernas y brazos. Es notoria la versatilidad y rapidez en comparación con el método anterior.

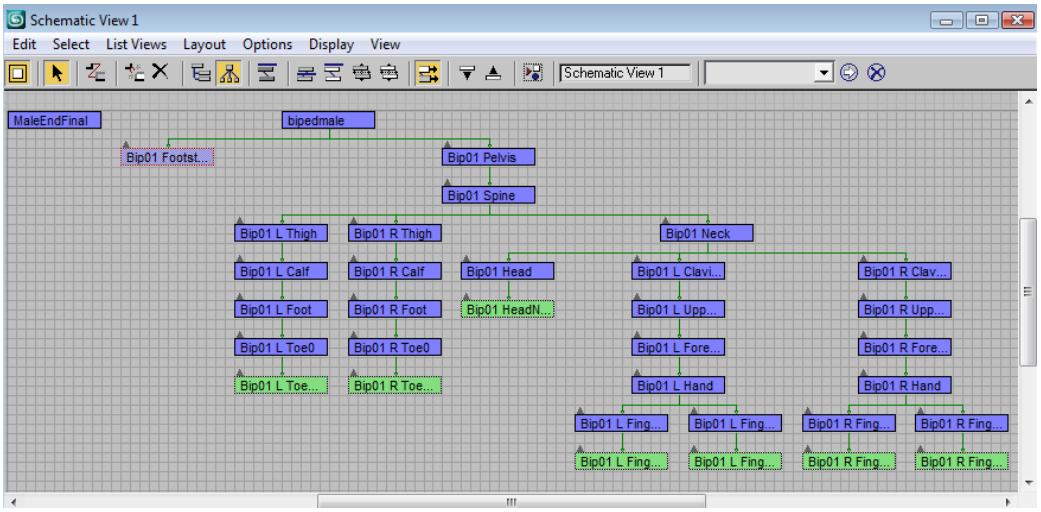


Figura 136. Jerarquía

Skinning

Skinning, es el proceso por el cual el personaje se vincula al esqueleto. Para la lo cual el modificador *Skin*, es activado en el personaje (*Figura 137*).

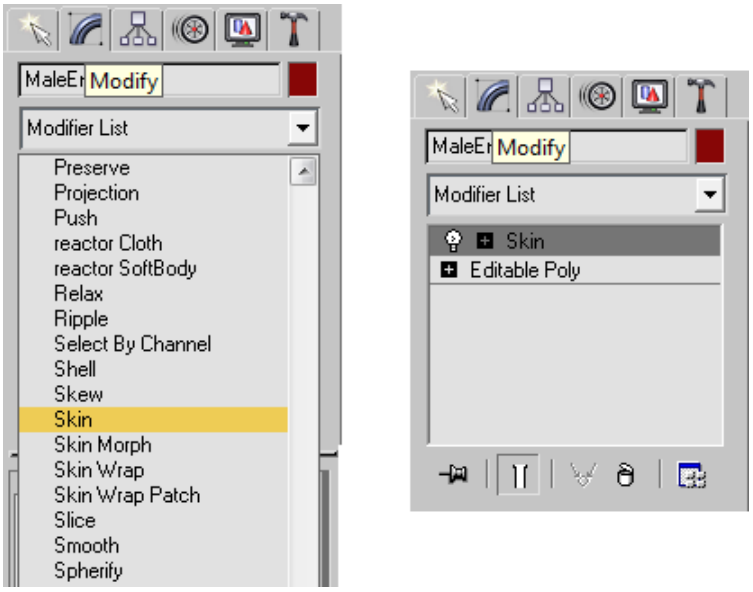


Figura 137. Skin

Al seleccionar *Skin*, este aparecerá en la lista de modificadores (*Figura 137*). El siguiente paso es agregar los huesos creados al modificador *Skin* (*Figura 138*).

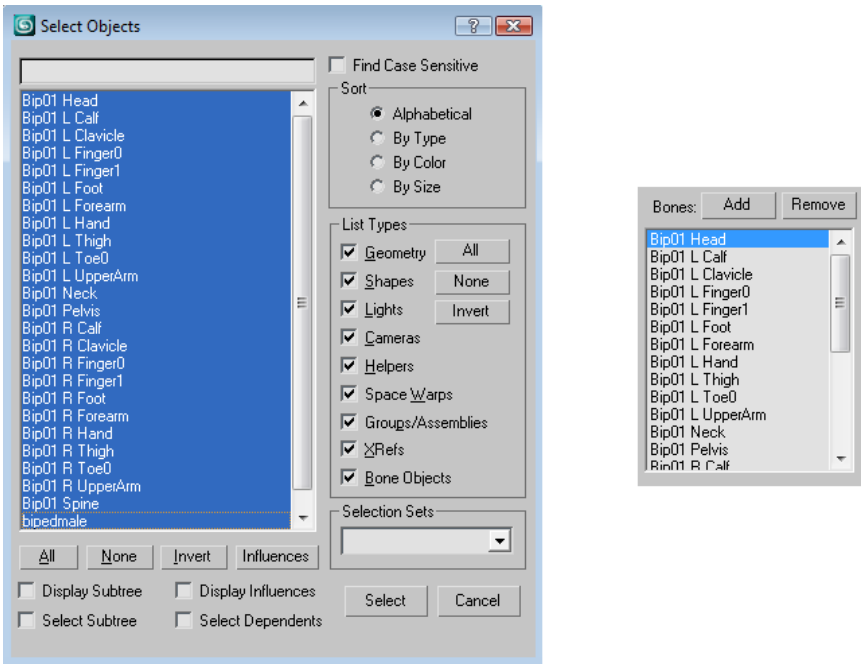


Figura 138. Añadir Huesos

Una vez seleccionados todos los huesos y añadidos al modificador (Skin) es posible constatar que el esqueleto se encuentra vinculado al personaje ya que éste puede mover sus huesos (*Figura 139).*

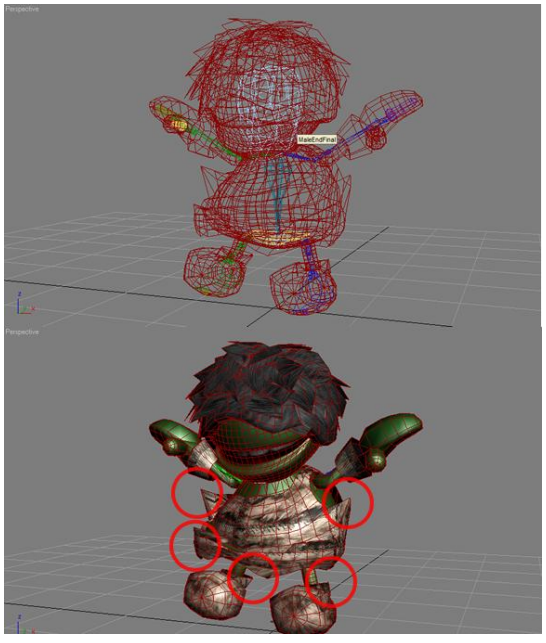


Figura 139. Verificación

El personaje puede presentar ciertas fallas a la hora de animar, como vértices que se mueven incorrectamente, lo cual es normal debido a que todavía no se determinan los pesos apropiados en los vértices correspondientes a cada hueso.

Edit Envelopes

Edit Envelopes o editar envolturas, permite modificar los pesos de una forma fácil y sencilla ya que el proceso requiere que el usuario limite la acción de los huesos con una envoltura similar a una cápsula la cual se puede agrandar en sus extremos de acuerdo a la conveniencia del usuario (*Figura 140*).

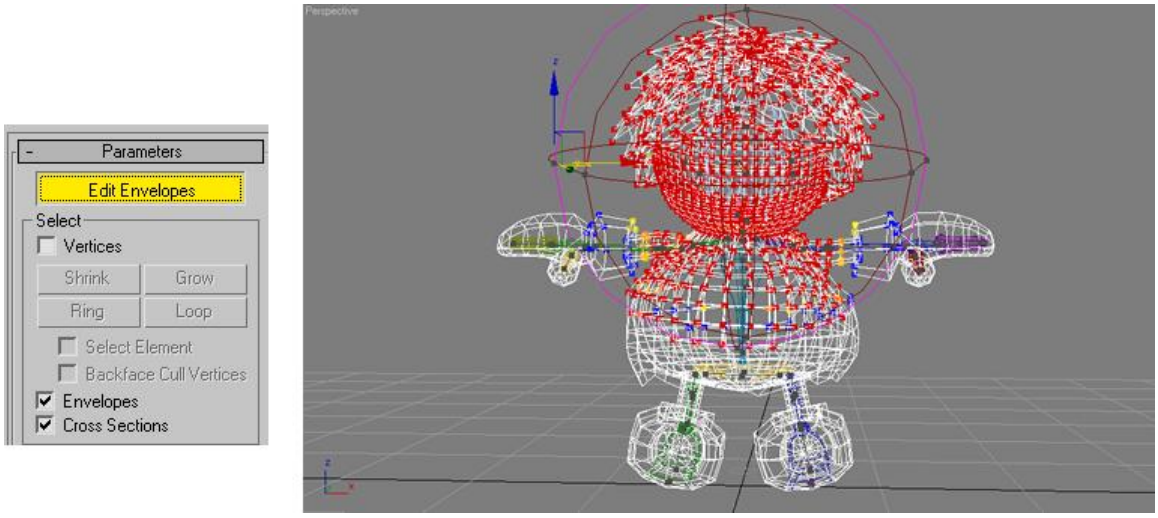


Figura 140. Edit Envelopes

Al activar esta opción, los vértices aparecerán mostrándose con coloraciones diferentes. Esto representa el grado en que cada hueso afecta su entorno. Las coloraciones van de blanco pasando por azul a rojo.

- Los vértices de color blanco están fuera del alcance del hueso seleccionado.
- Los vértices de color azul se encuentran afectados levemente. Los vértices de color azul también informan que existe otro hueso que lo afecta en un porcentaje igual o mayor al que influye el hueso actual.

- Los colores tomates y amarillos representan un grado de afectación mayor pero al igual que en el anterior caso, informa que existe otro hueso que lo afecta en un menor grado.
- Finalmente los vértices rojos están afectados en un 100% por el hueso.

Una vez terminado el proceso, el usuario podrá apreciar una mejoría en el movimiento del personaje (*Figura 141*).

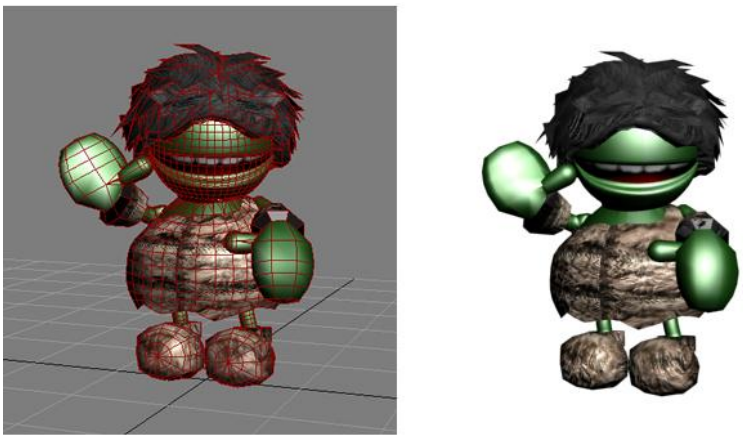
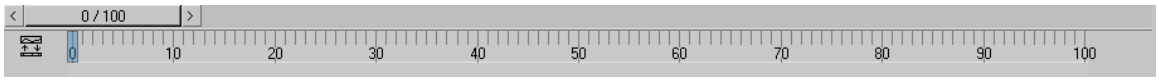


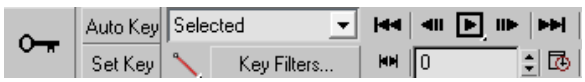
Figura 141. Resultado Final

Animación del Personaje

Terminado el proceso de adaptación del esqueleto al personaje, se procede a la animación del mismo, para lo cual se requiere el uso de la barra de animación, en donde se establecen *Keys* o puntos clave donde se almacenan los movimientos realizados por el personaje (*Figura 142*).



Tiempo de Animación (Animation Timer)



Herramientas de Animación (Animation Tools)

Figura 142. Animación

Para empezar la animación del personaje es necesario determinar ciertos parámetros dentro del menú *Time Configuration* o Configuración de tiempo, determinando parámetros como la velocidad y el número de *frames* (*Figura 143*).

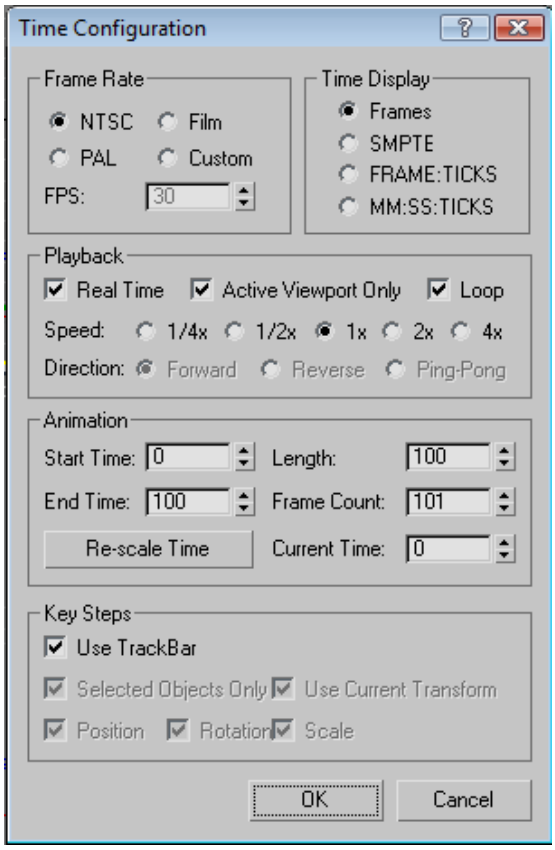


Figura 143. Configuración del Tiempo

La animación de personajes puede ser ejecutada de diferentes maneras, en esta sección se explican dos formas comúnmente usadas:

- Animación Manual
- Animación por medio de archivos (BIP)

Animación Manual

La animación manual o en base de *key frames* o cuadros clave, es el proceso por el cual se anima un personaje moviendo cada hueso individualmente. El primer paso es activar el modo de animación *Auto Key*, en el panel de animación (*Figura 144*). Tanto el botón como la barra de animación cambiarán a color rojo advirtiendo que todas las animaciones quedarán registradas por el *timer*.

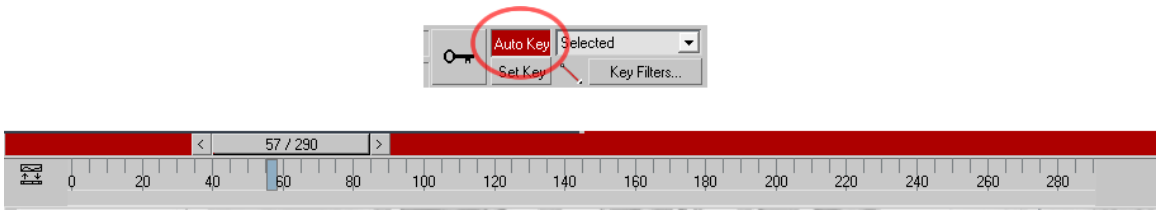


Figura 144. Panel de Animación.

Dentro del modo *Auto Key*, el personaje puede ser animado al mover los ayudantes para simular el moviendo en piernas y brazos. Es posible previsualizar este comportamiento mediante el uso de las herramientas de control (*Figura 145*).

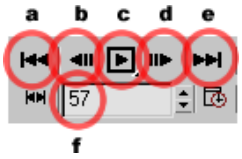


Figura 145. Control de la Animación

Los controles representan:

a. *Inicio:* Recorre hacia el inicio de la animación.
b. *Frame anterior*: Avanza un cuadro hacia atrás.
c. *Play:* Ejecuta la animación.
d. *Frame Siguiente*: Recorre un cuadro hacia adelante.
e. *Fin*: Recorre hasta el final de la animación.
f. *Frame Actual*: indica el frame en el que se encuentra la barra de animación.

Las animaciones (*Figura 146*) creadas para el personaje son las siguientes:

- *Root*: Se ejecuta cuando el usuario no maneja al personaje.
- *Run*: El personaje corriendo
- *Jump*: Animación de salto.
- *Side*: Movimiento a un lado

.

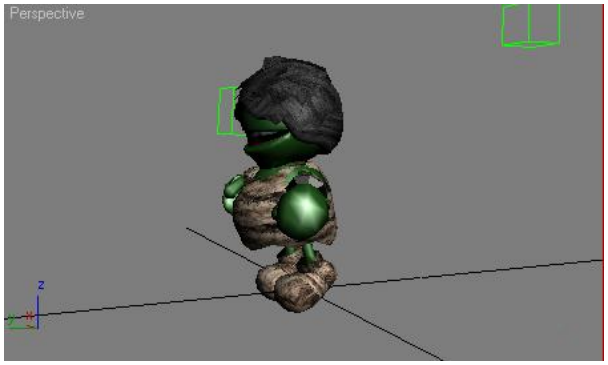


Figura 146. Jump - Saltar

Existen infinitas posibilidades de movimiento para un personaje, de acuerdo a lo que se busca obtener. En este caso es el movimiento básico del personaje.
Un aspecto a tomar en cuenta es que las animaciones deben ser consecutivas, es decir que una animación debe ir a continuación de la otra.

Animación usando archivos BIP.

El Proceso de animación mediante archivos *BIP* o *Bípedos*, es un mecanismo que facilita el proceso de animación. Las animaciones son creadas con equipos especiales que capturan el movimiento de una persona, lo cual garantiza fluidez y precisión (*Figura 147*).

Figura 147. Animación BIP

La animación es cargada en el personaje mediante la opción *Motion* (*Figura 148*).

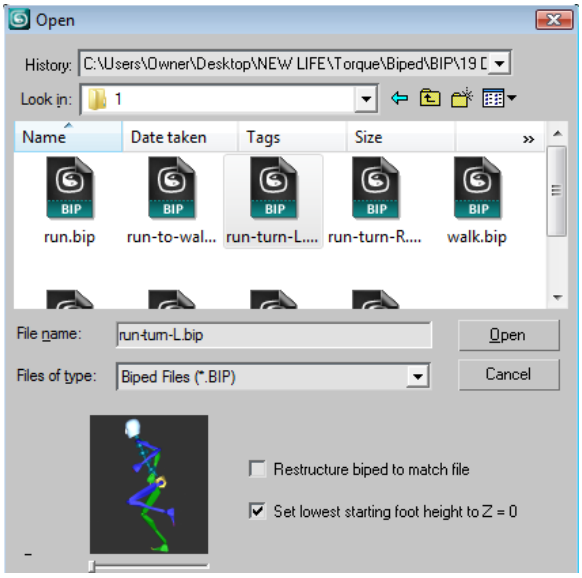
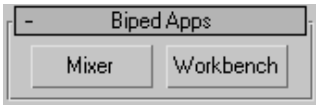


Figura 148. Motion

Al cargar el archivo en escena se obtiene el movimiento deseado para el personaje. En el caso que el personaje requiera mezclar animaciones se activa la opción *Mixer* (*Figura 149*).



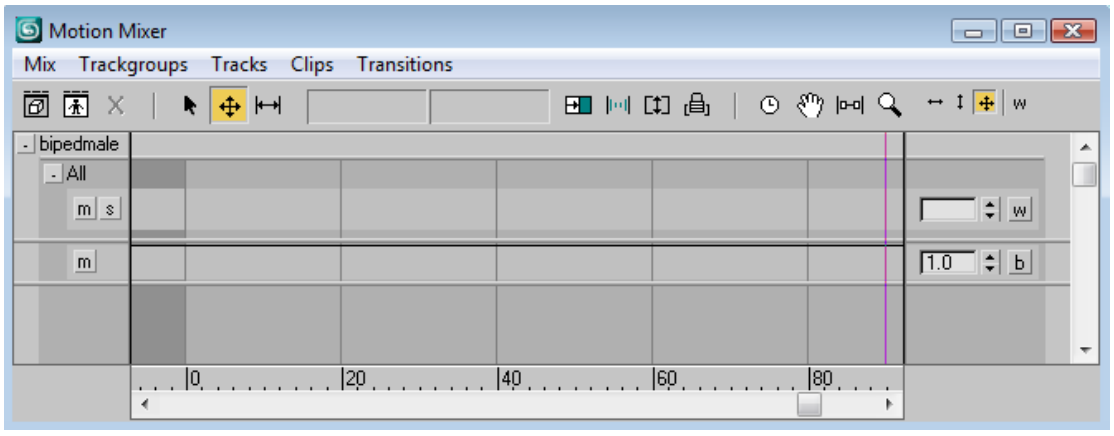


Figura 149. Mixer

En donde se importan y se mezclan las animaciones como muestra la figura *150*.

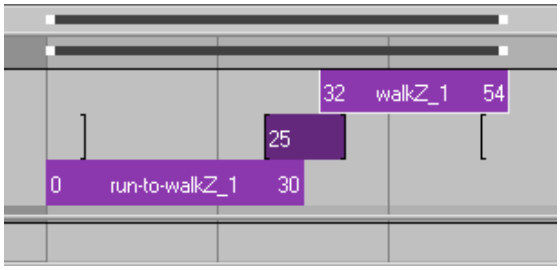


Figura 150. Mezcla de Animaciones

Exportación de un Personaje Animado.

Una vez terminado el proceso de animación, es posible exportar el personaje de dos formas o en dos posibles formatos:

- DirectX
- DTS

A continuación se explican ambos procesos mostrando tanto sus ventajas como sus desventajas.

Exportación *DirectX .x*

Este proceso es relativamente sencillo en relación al formato DTS pero carece de ciertos parámetros que necesitan ser implementados, como son las cámaras, secuencias y colisiones.

La exportación requiere el Plugin *Pandasoft*, el cual permite generar archivos de extensión (.x) (*Figura 151*).

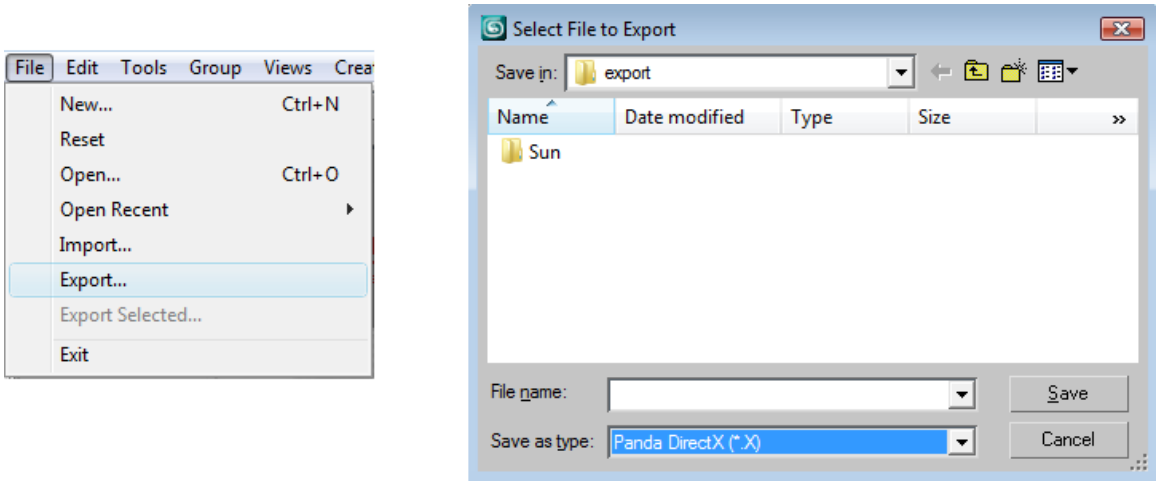


Figura 151. Exportar

Al momento de exportar las opciones de configuración, se especifican opciones como tipo de objeto (*estático – animado*), tipo de textura y uso de huesos (*Figura 152*).

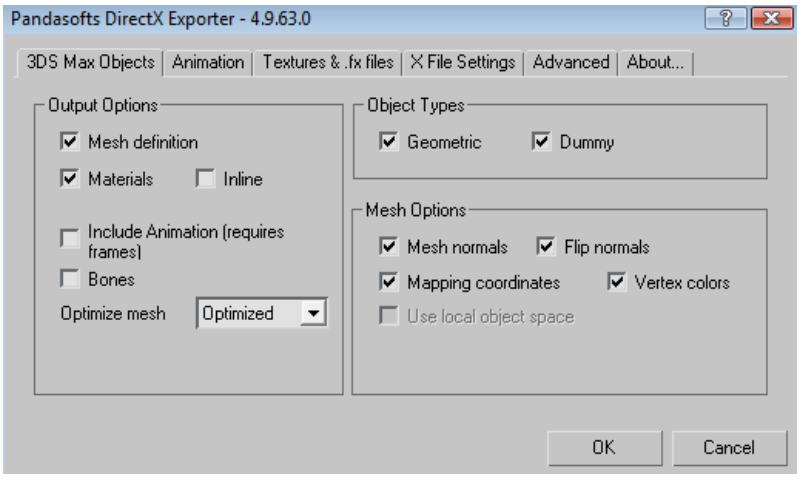


Figura 152. Opciones

En este caso se requieren las opciones:

- Include Animation
- Bones
- Texturas tipo JPG

Exportación DTS

El método de exportación mediante el uso de archivos DTS, es el proceso más usado e implementado por desarrolladores de videojuegos. Permitiendo mejorar, optimizar y disminuir el proceso de integración con el motor gráfico. El uso de archivos DTS ayuda a generar colisiones, posición de cámara, posición de objetos, generación de animaciones, e inclusive controlar la calidad gráfica del objeto dentro de la escena.

Primero, el objeto es ubicado en el eje de coordenadas, como se muestra en la figura *153*. A continuación, el objeto es nombrado, teniendo en cuenta que el nombre del objeto tiene que terminar en “2” ya que el motor gráfico identifica al valor como objeto para distinguirlo del resto de elementos.

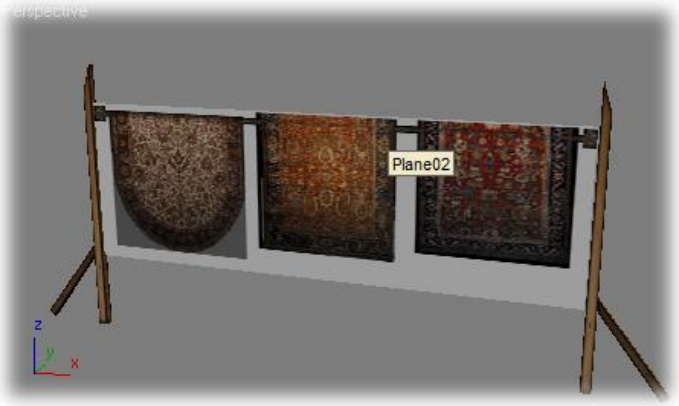


Figura 153. Objeto

El siguiente paso es encerrar al objeto dentro de una caja, la cual se encuentra dentro de las figuras básicas. A esta caja se le da el nombre de *bounds* nombre por el cual, la herramienta gráfica se guía para importar un objeto en escena (*Figura 154*).

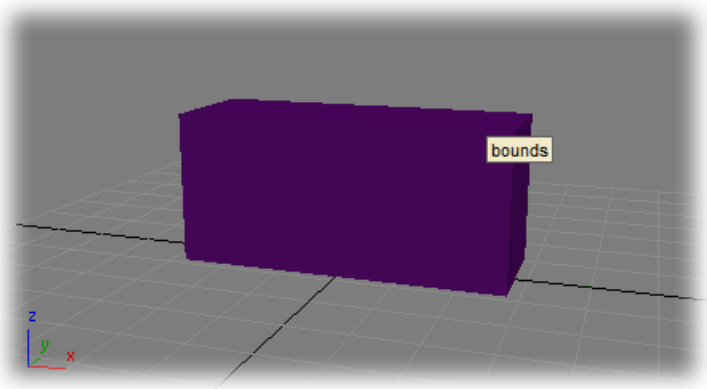


Figura 154. Bounds

Dentro de las opciones del exportador se activa la opción *Embed Shape* (*Figura 155*).

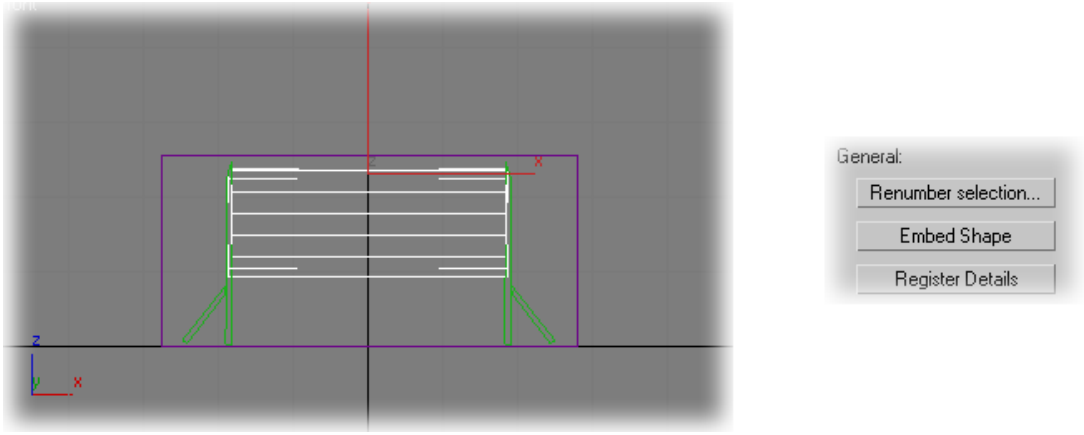


Figura 155. Embed Shape

Esta opción activa una estructura en forma de árbol que el motor gráfico puede interpretar (*Figura 156*).

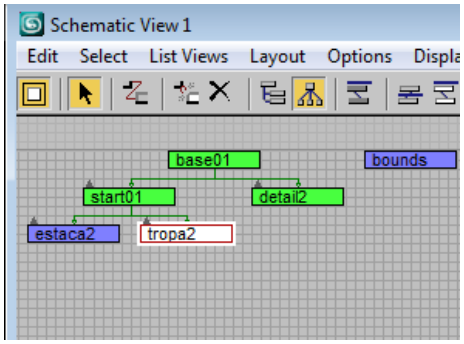


Figura 156. Jerarquías

Finalmente el objeto es exportado e importado dentro del motor gráfico (*Figura 157*).

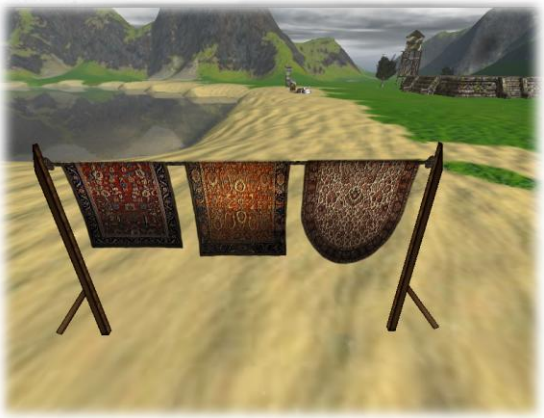


Figura 157. Figura terminada

4.1.6 Estabilización y Pruebas

En este punto se analizan minuciosamente, si los objetivos planteados en el *Sprint uno* se han cumplido. Por otro lado se verifica que las tareas realizadas sean funcionales y se encuentren operando correctamente en el sistema.

Dado que el *Sprint uno*, trata del manejo y uso del motor gráfico, es necesario familiarizarse con el motor y mantener pruebas constantes que mejoren el desempeño de la aplicación. Por lo cual se da prioridad a la ejecución de las siguientes tareas:

- Implementación de motor gráfico
- Carga de escenarios y personajes

Para la implementación del motor gráfico, es necesario hacer pruebas en cuanto al rendimiento tanto del motor como del computador. Es necesario diseñar un pequeño programa (*Figura 158*) en el cual se simula la generación de elementos en escena para tener una idea general de cómo se desenvuelve el motor bajo estrés.

Figura 158. Rendimiento

La figura *158*, presenta la aplicación gráfica, la cual instancia varios logos de la Escuela Politécnica del Ejército. En donde es posible:

- Cambiar la técnica de instanciación.
- Aumentar elementos en escena.
- Disminuir elementos en escena.

Las técnicas empleadas, son diseñadas en base a *Software* y a *Hardware*. El *Software* hace uso del procesador principal del computador para generar los múltiples objetos. Mientras que el *hardware*, emplea el uso de la tarjeta gráfica principal del ordenador. La técnica por Hardware no satura al computador ya que su memoria esta dedicada para este tipo de operaciones, haciendo de esta la técnica más implementada en aplicaciones 3D.

Con el aplicativo es posible medir el rendimiento a través de los frames por segundo o *Fps*. A mayor número de frames, mejor desempeño de la aplicación. Por lo cual, es recomendable que cualquier aplicación 3D no disminuya su rendimiento por debajo de los 30 frames, ya que la aplicación perderá fluidez y continuidad.

4.1.7 Toma de Resultados.

La toma de resultados, trata el cumplimiento de los objetivos planteados en el tiempo establecido al inicio del *Sprint* como se muestra en la tabla *7*.

Tareas	Tipo	Estado	Responsible	Fecha de Cumplimiento
Creación del Product Backlog	Análisis	Terminada	Natalia	2-dic
Concepto de Jugabilidad	Prototipado	Terminada	Natalia	13-nov
Implementación de Motor Grafico	Pruebas	Terminada	Christian	1-dic
Carga de escenarios y Personajes	Análisis	Terminada	Christian	26-nov
Ambientación	Codificación	Terminada	Natalia	19-nov
Skybox	Codificación	Terminada	Natalia	20-nov
Terreno	Codificación	Terminada	Christian	24-nov
Agua	Codificación	Terminada	Christian	24-nov
Animación	Codificación	Terminada	Christian	3-dic
Rigging	Codificación	Terminada	Christian	3-dic
Skinning	Codificación	Terminada	Christian	3-dic
Exportación	Codificación	Terminada	Christian	29-nov
Estabilización y Pruebas	Codificación	Terminada	Natalia	2-dic
Toma de Resultados	Codificación	Terminada	Natalia	3-dic

Tabla 7. Resultados

4.2. Sprint 2

4.2.1 Creación del Product Backlog.

Dentro del *Sprint dos* al igual que en el *Sprint uno* se establecen los parámetros iniciales como se muestra en la tabla *8*.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
2	3-dic-08	19	10

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis	Pendiente	Christian	23-dic
Codificación	En curso	Natalia	24-dic
Prototipado	Terminada		25-dic
Pruebas	Eliminada		
Reunión			

Tabla 8. Parámetros iniciales

La tabla *8*, describe el tipo de tarea así como la fecha de inicio del *Sprint dos*. Todos los detalles como días festivos son marcados como referencia para el desarrollo del proyecto.

La tabla *9* detalla cada tarea y especifica las horas de trabajo pendientes para cada día laborable.

SPRINT	INICIO	DURACIÓN
2	3-dic-08	19

					X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V
					3-dic	4-dic	5-dic	6-dic	7-dic	8-dic	9-dic	10-dic	11-dic	12-dic	13-dic	14-dic	15-dic	16-dic	17-dic	18-dic	19-dic	20-dic	21-dic	22-dic	23-dic	24-dic	25-dic	26-dic
				Tareas pendientes	13	13	13	12	10	12	12	11	10	9	9	7	8	8	8	7	6	6	5	6	4	4	5	3
				Horas de trabajo pendientes	190	180	170	160	135	150	140	130	120	110	100	89	90	80	70	60	50	40	37	30	26	26	18	6
PILA DEL SPRINT					ESFUERZO																							
Backlog	Tarea	Tipo	Estado	Responsable																								
	Menú Opciones	Análisis	Terminada	Christian	13	13	11	11	11	6	2																	
	Menú Créditos	Codificación	Terminada	Natalia	9	9	9	8	8	8	7	4																
	Menu Ayuda	Codificación	Terminada	Christian	9	9	9	8	8	8	7	5	2															
	Load Screen	Codificación	Terminada	Natalia	12	12	12	11	11	11	11	11	11	11	11	11	6	2	2									
	In Game GUI	Codificación	Terminada	Christian	14	14	14	13	13	12	12	12	12	10	6	6	6	6	2	2								
	Integración de Música y Fx	Codificación	Terminada	Christian	17	17	17	17	17	17	17	17	17	17	17	17	17	17	17	14	14	14	14	13	13	13	8	2
	Manejo de Partículas, luces y Cam	Codificación	En curso	Christian	21	21	21	21	21	21	21	21	21	21	21	21	21	21	21	20	15	10	10	8	8	8	6	3
	Integración de Shaders	Codificación	En curso	Christian	15	15	15	15	15	15	15	15	15	15	15	15	15	12	9	8	8	6	6	2				
	Estabilización y Pruebas	Codificación	Pendiente	Natalia	18	17	16	15	15	14	13	12	11	10	9	9	8	7	6	5	4	3	3	2	2	2	1	
	Toma de Resultados	Codificación	Pendiente	Natalia	19	18	17	16	16	15	14	13	12	11	10	10	9	8	7	6	5	4	4	3	3	3	2	1

Tabla 9. Tareas

	3-dic	4-dic	5-dic	6-dic	7-dic	8-dic	9-dic	10-dic	11-dic	12-dic	13-dic	14-dic	15-dic	16-dic	17-dic	18-dic	19-dic	20-dic	21-dic	22-dic	23-dic	24-dic	25-dic	26-dic
Christian	100	94	89	85	85	79	74	70	67	63	59	59	59	56	49	44	37	30	30	23	21	21	14	5
Natalia	90	86	81	75	50	71	66	60	53	47	41	30	31	24	21	16	13	10	7	7	5	5	4	1

Tabla 10. Horas Pendientes

Una vez terminadas las tareas, se analiza el esfuerzo realizado (*Figura 159*), el desarrollo de las tareas (*Figura 160*) y las horas empleadas por cada integrante del equipo de trabajo (*Gráfico 161*).

Proyecto
Evolution

SPRINT	INICIO	DÍAS
2	3-dic-08	19

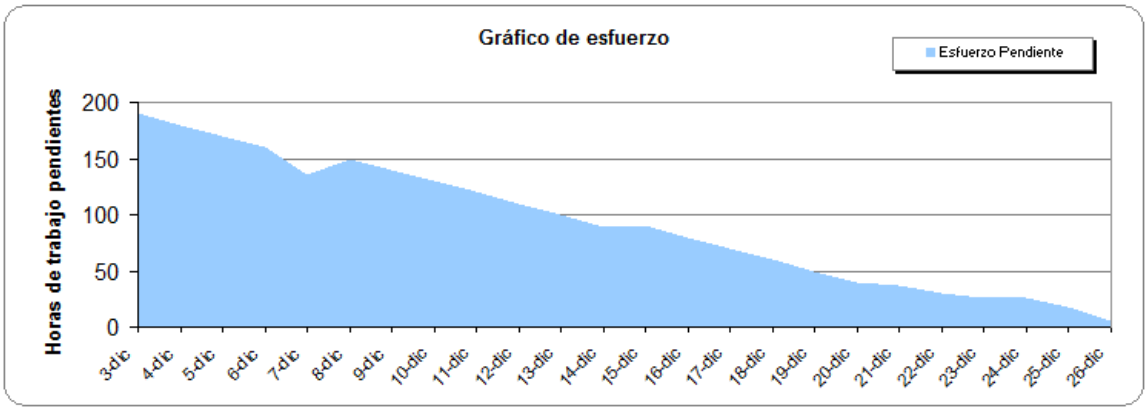


Figura 159. Esfuerzo

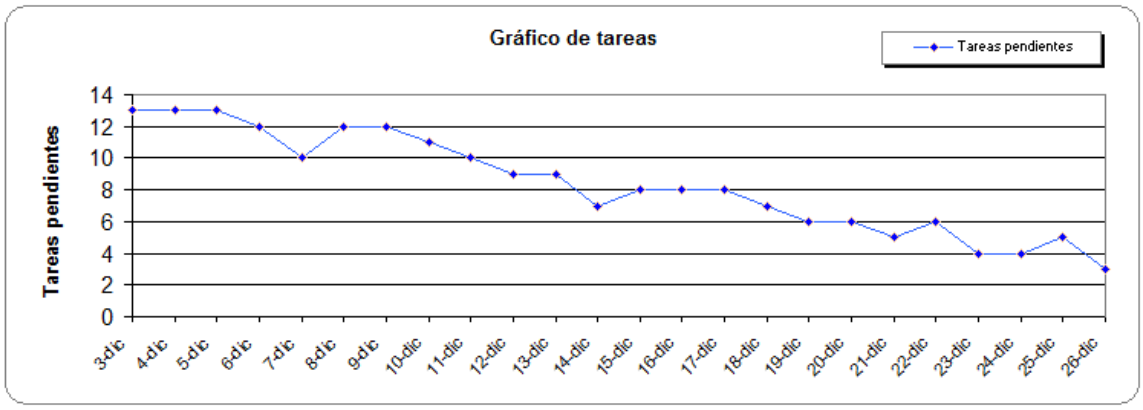


Figura 160. Tareas

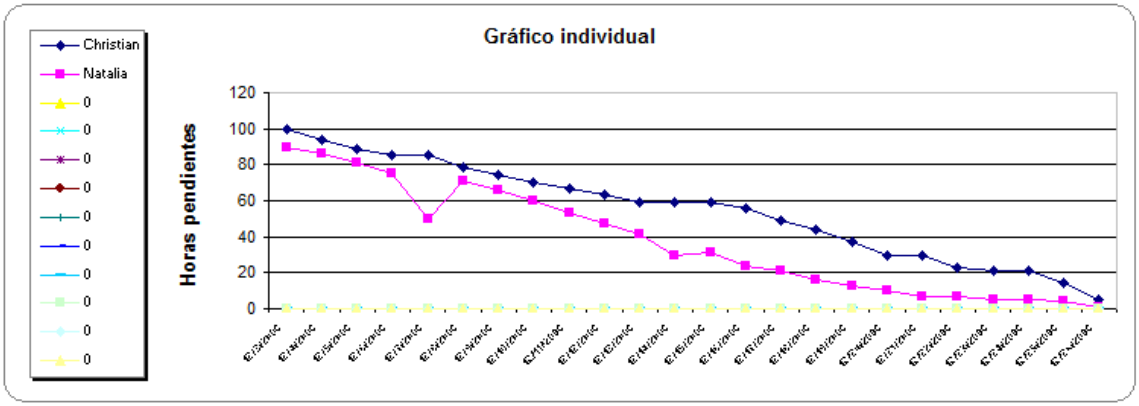


Figura 161. Horas de Trabajo

4.2.2 Desarrollo y Creación de Menús.

Esta etapa comprende la creación de pantallas de ayuda, configuración, selección de nivel que ayudan al usuario a guiarse dentro de la aplicación. Estos se detallan en orden de aparición dentro del juego:

- **Splash Screen:** Imágenes introductorias del Juego
- **Pantalla o menú principal**: Pantalla con las opciones principales, que son:
    - **Créditos**: Pantalla con los nombres de los creadores del juego.
    - **Ayuda**: Pantallas con ayudas de cómo funciona el juego.
    - **Configuración**: Pantalla que permite cambiar diversos parámetros en cuanto a la apariencia gráfica del computador.
- **Selección de juego:** Menú de selección de juego.
- **Load Screen:** Pantalla de espera para carga y ejecución del Juego.
- **In Game GUI**: Menú de controles dentro del Juego.

El desarrollo de cada una de las Pantallas o Menús se detalla a continuación:

Splash Screen

Básicamente el splash screen comprende un grupo de imágenes o videos que presentan el juego al Usuario. Estas imágenes o videos pueden introducir el tipo de juego, presentar auspiciantes o dar crédito a personas o empresas que brindaron su ayuda en el desarrollo del juego.

Evolution muestra una secuencia de imágenes del tema principal, del rating y de la institución a la cual el juego será presentado.

El rating (*Figura 162*) establece la orientación del juego al público donde:

Figura 162. Ratings

- *E (Everyone):* Todo público, no limita su uso ya que su contenido es apto para cualquier persona.
- *T (Teens):* Para Jóvenes y Adultos, limita su uso a personas con criterio formado, debido a violencia o lenguaje usado.
- *M (Mature):* Para público adulto. El contenido no es apto para menores de edad.
- *NR (Not Rated):* El juego no ha sido calificado o clasificado para un tipo de público en particular.

Evolution entra en la categoría E (everyone), ya que su contenido va orientado a público en general, sin ningún tipo de violencia.

Una vez que las imágenes han sido establecidas, el paso siguiente es mostrarlas en la aplicación. Para ello se crea una función que presente una a una las imágenes en un intervalo de tiempo.

```
Function loadStartup ()
{
GlobalActionMap.bind (keyboard, "escape", SkipIntro);
StartupGui.done = false;
  Canvas.setContent (StartupGui);
  Schedule (100, 0, checkStartupDone);
  SfxPlay (AudioStartup);
}
```

LoadStartup, verifica cuándo las imágenes han sido presentadas para pasar al menú principal y el sonido de fondo.

```
Function checkStartupDone ()
{
  If (! isObject (StartupGui))
  {
    Return;
  }

  If (StartupGui.done)
  {
    If ($index == 1) // change this value if the bitmap array increases
```

```
    {
      StartupGui.done = true;
      StartupGui.delete ();
      FlushTextureCache ();
      GlobalActionMap.unbind (keyboard, "escape");
      GlobalActionMap.bindCmd (keyboard, "escape", "", "handleEscape () ;");
      LoadMainMenu ();
    }
    Else
    {
      Next ();
      $musicHandle = SFXPlay (snd_evo);
      Schedule (1000, 0, checkStartupDone);
    }
  }
  Else
  {
    Schedule (1000, 0, checkStartupDone);
  }
}
```

La función *checkStartupDone*, pasa cada una de las imágenes en un intervalo de tiempo medido en milisegundos. También permite al usuario saltar la introducción al presionar cualquier tecla.

Menú Principal.

El menú principal contiene los accesos a las funciones principales de la aplicación como se ilustra en la figura *163*.

Figura 163. Menu Principal

Selección del Nivel del Juego.

Permite al usuario decidir el capitulo que desea jugar (*Figura 164*).

Figura 164. Capítulos

Los modelos previamente creados durante el *Sprint uno*, son usados en el menú simulando la rotación de la tierra alrededor del sol.

Opciones

Contiene configuraciones básicas para aumentar o disminuir las capacidades gráficas del computador. Esto va de acuerdo al poder de cómputo de un equipo en particular en cuanto al *CPU* y a la tarjeta gráfica (*Figura 165*).

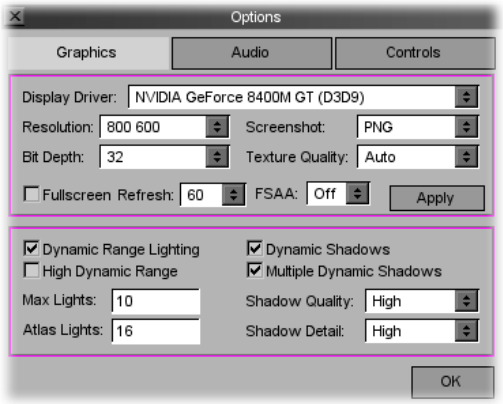


Figura 165. Opciones de Configuracion

También incluye opciones como:

- Volumen
- Dispositivo de Audio
- Controles

- Resolución

El control de las opciones de configuración del Juego viene incluido dentro del motor gráfico ya que son codificadas a nivel de hardware y para lo cual se necesita un conocimiento alto en cuanto al manejo de dispositivos y programación a bajo nivel.

Créditos

Los créditos presentan los nombres de los autores del juego así como de todas las personas involucradas en su desarrollo como muestra la figura *166*.

Figura 166. Créditos

Ayuda

Brinda una breve descripción del juego, como la historia, controles y modo de juego. La ayuda del juego debe ser clara para que no exista confusión por parte del jugador ya que esto puede causar frustración (*Figura 167*).

Figura 167. Ayuda

Load Screen

Load Screen, es una pantalla que se muestra mientras el usuario espera que el nivel se construya, cargando sonidos, objetos, etc. Muchas veces se muestra el proceso de carga con una barra de estado para tener constancia de que todo este marchando bien (*Figura 168*).

Figura 168. Load Screen

In Game GUI

Son los controles o información de estado durante el juego. Brindando información en tiempo real que mantiene al usuario al tanto del progreso del juego. Esta pantalla indica varios aspectos como: número de vidas, estado del juego, etc. (*Figura 169*).

Figura 169. Game GUI

También contiene controles como:

- **Generadores de eventos naturales:** Los cuales inician fenómenos naturales como:
  - Lluvia
  - Nevada
  - Terremoto
  - Explosión Volcánica
- **Estado del juego:** Permite al jugador mantener el control del juego en:
  - Número de habitantes
  - Mujeres
  - Hombres
  - Animales
- **Tiempo de Supervivencia:** El tiempo que el jugador logra mantener el control sobre la evolución de los habitantes.
- **Avisos:** mensajes que mantienen al tanto al jugador de los eventos generados o del estado de los habitantes en la isla.

4.2.3 Integración de Música y Efectos de Sonido

El sonido es parte fundamental en un juego ya que brinda la sensación al jugador de estar en medio de la acción. Muchas empresas tienden a contratar sonidistas y compositores ya que cada canción o sonido posee derechos de autor, por lo cual dentro de Evolution, se utilizan sonidos y fondos musicales de distribución libre para su reproducción.

Una vez seleccionados los sonidos a ser usados, es necesario tenerlos en un formato común ya que los motores gráficos soportan pocos tipos de frecuencias debido a que el código fuente de muchos *codecs* no son distribuidos por motivos de seguridad. El motor gráfico implementado en *Evolution* soporta los archivos de sonido:

- *Wav*: o Waveform, formato de audio de PC para almacenar y comprimir datos sonoros.

- *Ogg*: Formato libre que comprime audio en alta calidad reduciendo espacio en disco.

A pesar de que *Wav* y *Ogg* comprimen la información de archivos de audio, la compresión es mínima si se compara con el formato *Mp3,* el cual reduce significativamente el espacio en disco sin perder la calidad de audio*,* pero a diferencia de Wav y Ogg, Mp3 posee derechos de autor.

Teniendo todos los sonidos en el formato adecuado, el siguiente paso es integrarlos dentro de la aplicación. En primer lugar se instancia los sonidos dentro del motor aplicando el siguiente código:

```
Datablock SFXDescription (AudioDefault3d)
{
   Volume   = 1.0;
   IsLooping= false;

   is3D     = true;
   ReferenceDistance= 20.0;
   MaxDistance= 100.0;
   Channel = $SimAudioType;
};
```

Dentro del perfil de audio, el motor puede manejar los archivos de acuerdo a su uso, ya que por ejemplo, los sonidos ambientales siempre están activados, y otros sonidos como el sonido de un ave suenan siempre y cuando el jugador se encuentre cerca del objeto. Por lo cual existen dos tipos de emisores de audio:

- *Audio 2D*: audible todo el tiempo durante el juego.
- *Audio 3D*: audible dentro de un rango de aproximación.

Otro aspecto destacable es la propiedad *isLooping*, la cual permite repetir el sonido continuamente o reproducirlo una sola vez al ser iniciado.

Creado el perfil de audio, se crea el archivo de audio que será implementado con el código:

```
Datablock SFXProfile (bg_main)
{
```

```
Filename = "~/data/sound/main.wav";
Description = "AudioLooping2D";
Preload = true;
};
```

En donde:

- *Filename*: Es la ubicación del archivo en el computador.
- *Description*: Contiene el nombre del perfil previamente creado.
- *Preload*: Permite cargar el sonido al momento en el que el juego es iniciado.

Finalmente se instancia el sonido dependiendo del efecto que se desee conseguir mediante el código.

```
$musicHandle = SFXPlay (bg_main);
```

$musicHandle, permite manejar y almacenar en un buffer de memoria el sonido que se reproducirá al momento que este es llamado.

4.2.4 Manejo de Partículas, luces y cámaras

Partículas, luces y cámaras ocurren dentro del motor gráfico y en todos los casos es necesario codificar algunos parámetros para que cada uno tenga el efecto y secuencia esperado.

Partículas

Las partículas, son comúnmente usadas en simulación de explosiones, humo y polvo (*Figura 170*), pero realmente no existen barreras en su uso, todo depende del ingenio detrás de los creadores.

Figura 170. Uso de Partículas

Para *Evolution* el uso de partículas ayudarán a simular efectos como:

- Lluvia
- Humo
- Fuego
- Neblina
- Polvo

Es necesario codificar de acuerdo al comportamiento de cada partícula. Estas partículas son simplemente imágenes transparentes (*Figura 171*) que son multiplicadas por el motor gráfico mediante cálculos matemáticos que agrandan y encogen la imagen consiguiendo efectos de difuminación y desvanecimiento.

Figura 171. Partícula

Para su implementación dentro del motor gráfico, es creado el siguiente código:

```
Datablock ParticleData (ChimneySmoke)
{
   TextureName          = "~/data/shapes/particles/smoke";
   DragCoefficient      = 0.0;
   GravityCoefficient   = -0.2;
   SpinRandomMin = -30.0;
   SpinRandomMax = 30.0;
   Colors [0]     = "0.6 0.6 0.6 0.1";
   Colors [1]     = "0.6 0.6 0.6 0.1";
   Colors [2]     = "0.6 0.6 0.6 0.0";
   Sizes [0]      = 0.5;
   Sizes [1]      = 0.75;
   Sizes [2]      = 1.5;
   Times [0]      = 0.0;
   Times [1]      = 0.5;
   Times [2]      = 1.0;
};
```

En donde:

- **TextureName**: Indica la ubicación de la imagen transparente.
- **DragCoefficient**: Es el valor que retarda el movimiento de las partículas.
- **GravityCoefficient**: Es el valor de gravedad que afecta el desplazamiento de las partículas.
- **SpinRandomMin**: Es el valor mínimo que acelera la rotación de partículas.
- **SpinRandomMax**: Es el valor máximo que acelera la rotación de partículas.
- **Colors**: Altera el color de la partícula.
- **Sizes**: Transforma a la partícula en el eje X y Y.

El resultado final se muestra en la figura *172*.

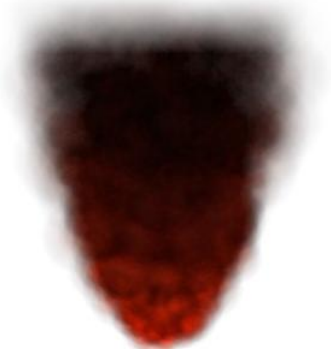


Figura 172. Generación de Partículas

Luces

La creación de luces es realizada mediante código ya que la iluminación puede darse tanto en objetos estáticos, como en el terreno. Existen dos tipos de iluminación:

- Iluminación Global.
- Iluminación Dirigida.

La iluminación global, es a nivel ambiental. Generalmente es usada para simular el sol en un entorno (*Figura 173*).

Figura 173. Iluminación Global

La iluminación dirigida, se genera en un punto en el plano, el cual irradia luz dentro de un rango limitado (*Figura 174*).

Figura 174. Iluminación Dirigida

El código empleado en ambos casos es el siguiente:

```
New Sun (SunObject) {
   CanSaveDynamicFields = "1";
   Enabled = "1";
   Azimuth = "300";
   Elevation = "14";
   Color = "0.85 0.83 0.73 1";
   Ambient = "0.5 0.47 0.43 1";
      Direction = "0.740866 0.370433 -0.560265";
      Position = "0 0 0";
      Scale = "1.4 1.4 1.4";
      ShadowColor = "0.200000 0.150000 0.100000 0.400000";
};
```

En el cual se especifican parámetros de**:**

- **Azimuth**: Es el ángulo de iluminación.
- **Elevation**: Representa la elevación del punto de origen del emisor.
- **Color**: Color de la Luz generada.

- **Ambient**: Color del ambiente, en el cual es posible simular el día y la noche en el escenario.
- **Direction**: Dirección de enfoque.
- **Position**: Posición del punto de origen.
- **Scale**: Scala del punto de origen.
- **ShadowColor**: Color de sombra que será producido por los objetos cercanos.

Cámaras

Para el manejo de cámaras, se emplean puntos guías que marcan el camino por el cual la cámara recorrerá al ser instanciada. Este mecanismo es usado para movimientos controlados (*Figura 175*).

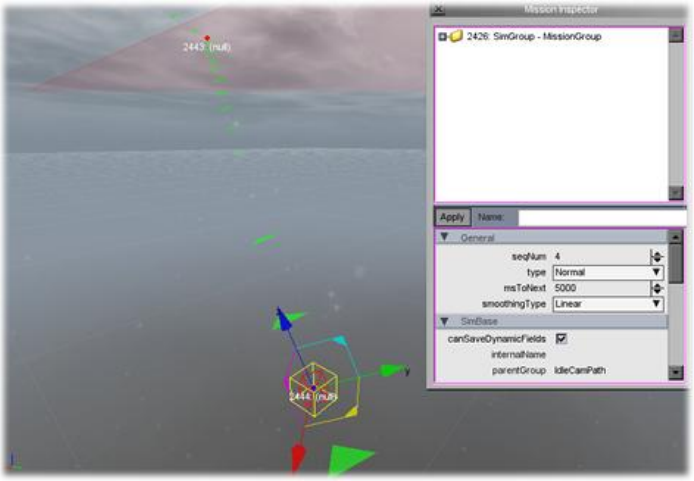


Figura 175. Cámara Guiada

El código implementado para controlar cámaras con rutas establecidas es:

```
New Marker () {
   CanSaveDynamicFields = "1";
   Position = "8511.9 1648.24 -38.8012";
   Rotation = "-0.813406 -0.365181 -0.452784 89.8499";
   Scale = "0.6 0.6 0.6";
   SeqNum = "1";
   MsToNext = "500";
   SmoothingType = "Linear";
      Speed = "15";
      Stop = "1";
};
```

Donde:

- **Position**: Posición del punto por el cual la cámara pasara.
- **Rotation**: Rotación de la cámara de un punto a otro.
- **Scale**: Escala del punto origen.
- **SeqNum**: Numero de secuencia, indica el ID del punto. Donde 1 es el punto en donde se origina la ruta.
- **MsToNext**: Milisegundos entre un punto y otro.
- **SmoothingType**: Movimiento de la cámara en relación a los puntos. Este puede ser lineal o suavizado.
- **Speed**: Velocidad de movimiento de la cámara.
- **Stop**: Indica si la cámara debe detenerse o proseguir al siguiente punto.

Para cámaras libres como las usadas sobre los personajes, se necesita especificar un punto de origen dentro del programa de edición 3D (*3DMax*), realizado durante el *Sprint uno* (*Figura 176*).

Figura 176. Cámara Libre

4.2.5 Integración de Shaders

Shaders, trata el uso de técnicas de iluminación en objetos estáticos o animados. Básicamente crean un efecto de brillo, iluminación, reflejo o desplazamiento de las texturas existentes para un objeto en particular. El uso de *shaders,* aumenta el procesamiento en el CPU del computador, por lo cual es

recomendado su uso exclusivamente para computadores o consolas que posean una calidad de procesamiento *media – alta*.

En la figura *177*, se muestra un ejemplo de uso de shaders aplicados en el juego, el cual esta dado en el agua que simula la existencia de olas y calcula el reflejo del entorno.

Figura 177. Shaders

Esto es posible en elementos estáticos gracias al uso de imágenes especiales denominadas *Normal Maps* (*Figura 178*) que simulan el brillo recibido.

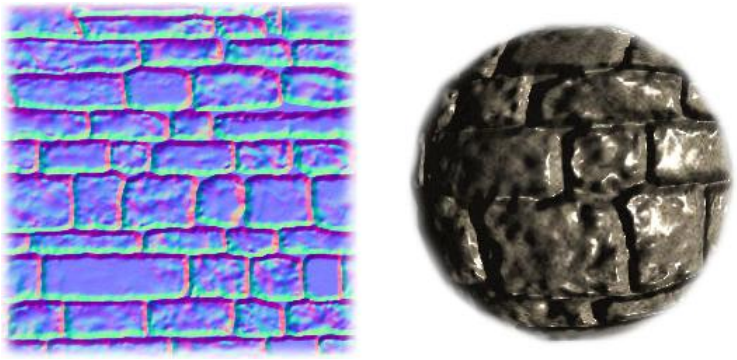


Figura 178. Normal Maps

Normal map, es una técnica denominada *Bump Mapping* o mapa de desplazamiento que ayuda a añadir detalles sin tener que crear un objeto que posea un número elevado de polígonos. Como muestra la figura *178*, a pesar de ser una esfera llana la figura tridimensional, la imagen *Normal Map*, crea un efecto de ladrillo con hendiduras y profundidad sin afectar el performance dentro del motor gráfico.

Para la generación de *Normal Maps*, se implementa la herramienta *Gimp* especializada en la manipulación de texturas. El programa posee filtros especiales de edición 2D (*Figura 179*).

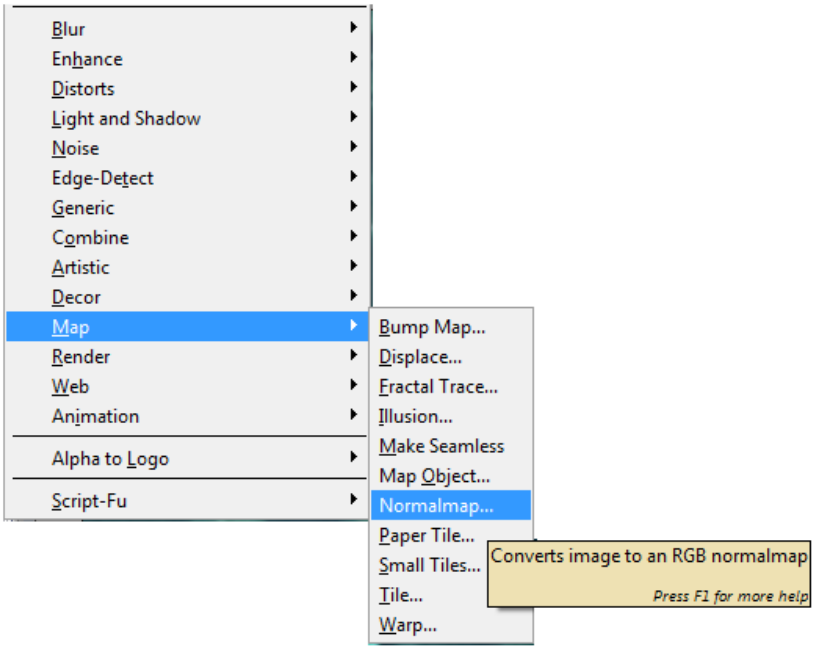


Figura 179. Filtros

El filtro permite simular profundidad mediante la propiedad *Scale* como se muestra en la figura *180*.

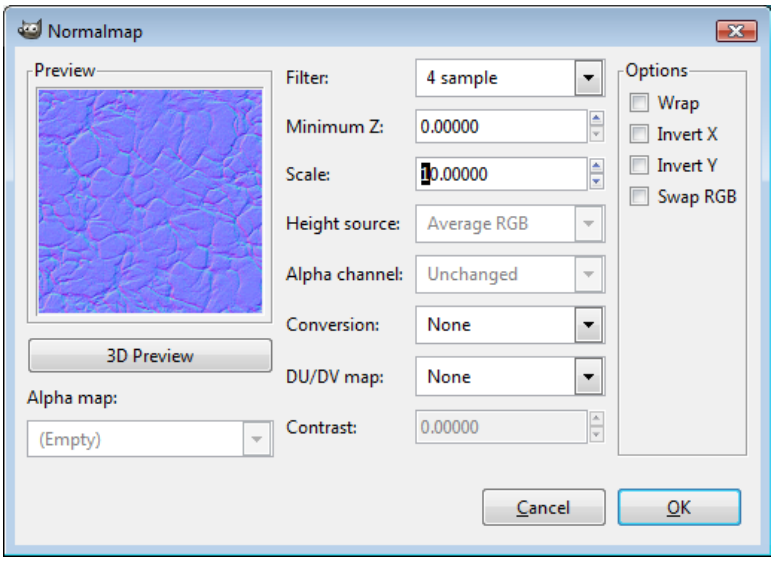


Figura 180. Propiedades

Otros efectos tomados en consideración para el desarrollo de *Evolution* son:

Brillo: Efecto de volumen lumínico en un material. En el caso de la figura *181* el objetivo es simular lava en un volcán.

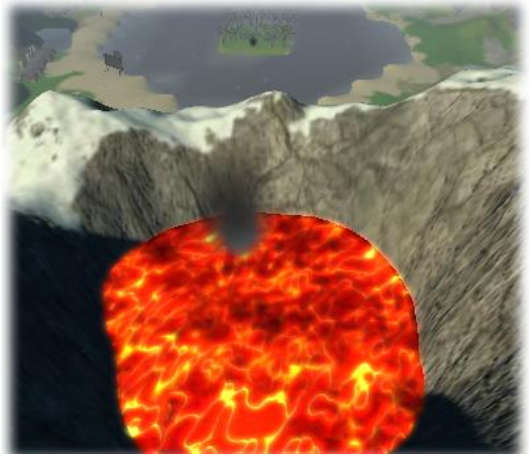


Figura 181. Emisión de Luz

Desplazamiento: Efecto de desplazamiento o profundidad de un objeto. La figura *182*, muestra el efecto de profundidad simulado en el logo de la *Escuela Politécnica del Ejército*.

Figura 182. Profundidad

El proceso de implementación, requiere del uso de un archivo externo en el cual se codifica y se especifican los parámetros:

```
New Material (Logo_Espe)
{
   BaseTex [0] = "~/data/Nature/espe";
   BumpTex [0] = "~/data/Nature/espeN";
   Specular [0] = "0.8 0.8 0.8 0.1";
   SpecularPower [0] = 32;
```

```
   PixelSpecular [0] = true;
};
```

En donde:

- **BaseTex**: Textura base del objeto
- **BumpTex**: Textura (Normal Map) que regula los niveles de brillo del objeto
- **Specular**: Valor de cubrimiento del brillo en el objeto
- **SpecularPower**: Valor de intensidad de brillo
- **PixelSpecular**: Activar brillo en un objeto estático

4.2.6 Manejo de Detección de Colisiones

El manejo y detección de colisiones dentro del juego permite que tanto el jugador como los objetos dentro de la escena, no atraviesen otros objetos y respeten aspectos de gravedad y física.

Existen dos mecanismos que permiten lograr el mismo efecto, implementados de distinta forma y en distintos procesos. El primero es aplicado dentro del motor gráfico, en donde el modelo es encerrado dentro de una figura básica, como un cubo o una esfera la cual no es visible para el usuario (*Figura 183*).

Figura 183. Detección de Colisiones.

Otro método es implementado durante el proceso de exportación dentro de la herramienta de edición 3D, en donde se encierra al objeto en una figura geométrica básica, limitando y estableciendo una colisión como se muestra en la figura *184.*

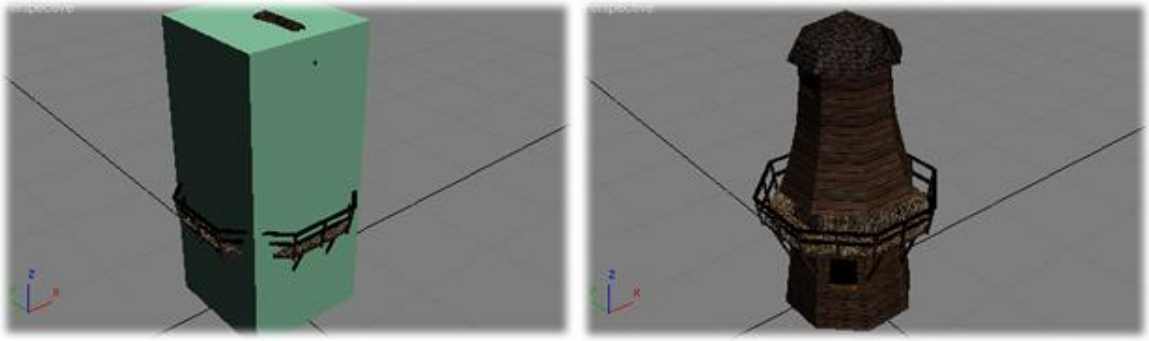


Figura 184. Detección de Colisiones

Es recomendable utilizar cubos para encerrar a los objetos ya que al utilizar otro tipo de figuras como esferas o pirámides, se disminuye el performance del objeto en el juego. Adicionalmente se crea una caja del mismo tamaño y posición, con los nombres *Collision-1* y *Col-1* respectivamente.

Es posible crear más de un elemento de colisión en un objeto, pero este puede tener un máximo de siete elementos, los cuales son interpretados por el motor gráfico de forma automática.

Al igual que en el *Sprint uno* durante la exportación de objetos, se define una jerarquía en la cual se adjuntan las cajas de colisión como se muestra en la figura *185.*

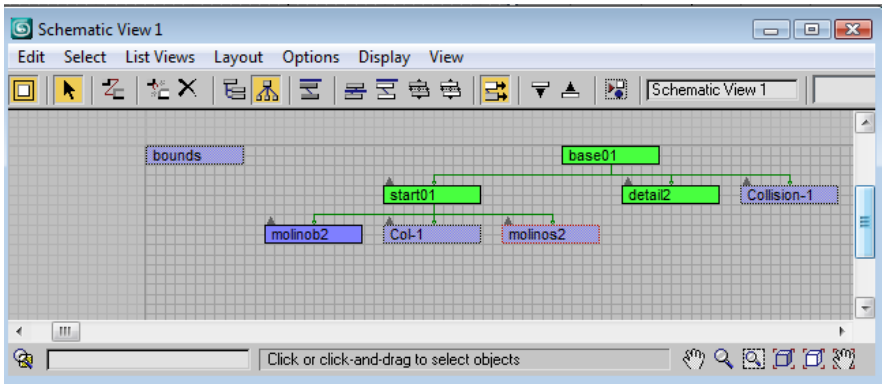


Figura 185. Jerarquías y Colisiones.

Finalmente, el objeto es exportado e importado por el motor gráfico sin necesidad de codificar sus colisiones.

4.2.7 Estabilización y Pruebas

La estabilización y la necesidad de pruebas dentro del *Sprint dos*, son fundamentales ya que se pone a prueba el código implementado para los elementos principalmente visuales del juego. Una vez analizado el desempeño del motor en cuanto al performance en el *Sprint uno*, se da paso al análisis de:

- Manejo de partículas, luces y cámaras.
- Integración de shaders.

Al igual que en el *Sprint uno*, son necesarias mini aplicaciones de prueba para verificar que durante el desarrollo del juego no existan problemas de eficiencia. La aplicación, tiene por objetivo implementar el uso de efectos visuales como shaders, luces y cámaras como se muestra en las figuras *186* y *187*.

Figura 186. Shaders

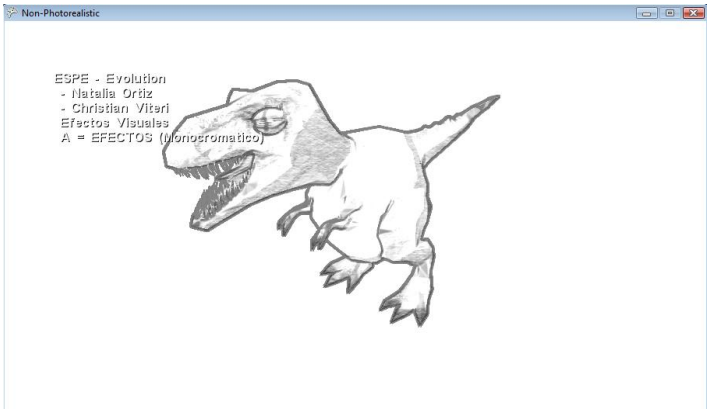


Figura 187. Luz

4.2.8 Toma de Resultados

La toma de resultados, en el *Sprint dos* analiza el cumplimiento de los objetivos planteados de acuerdo al tiempo establecido como se muestra en la tabla *11*.

Tareas	Tipo	Estado	Responsible	Fecha de Cumplimiento
Creación del Product Backlog	Análisis	Terminada	Natalia	26-dic
Menú Principal	Prototipado	Terminada	Christian	6-dic
Menú Selección de Juego	Pruebas	Terminada	Natalia	14-dic
Menú Opciones	Análisis	Terminada	Christian	10-dic
Menú Créditos	Codificación	Terminada	Natalia	11-dic
Menu Ayuda	Codificación	Terminada	Christian	12-dic
Load Screen	Codificación	Terminada	Natalia	18-dic
In Game GUI	Codificación	Terminada	Christian	19-dic
Integración de Música y Fx	Codificación	Terminada	Christian	26-dic
Manejo de Partículas, luces y Cam	Codificación	Terminada	Christian	26-dic
Integración de Shaders	Codificación	Terminada	Christian	23-dic
Estabilización y Pruebas	Codificación	Terminada	Natalia	26-dic
Toma de Resultados	Codificación	Terminada	Natalia	26-dic

Tabla 11. Resultados

4.3. Sprint 3

4.3.1. Creación del Product Backlog

Según la metodología Scrum, es necesario diseñar el Product Backlog, el cual contendrá todas las tareas correspondientes al sprint tres. Para cada tarea se define una persona responsable de su desarrollo. A continuación se muestra la tabla 12, la cual indica el tipo de tarea a desarrollarse, el nombre de la tarea, el estado de esta tarea, la persona responsable de cumplirla y los días festivos que existe durante este tiempo.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
3	10-nov-08	20	3

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis	Pendiente	Christian	6-dic
Prototipado	En curso	Natalia	24-dic
Prototipado	Terminada	Christian	1-ene
Codificación	Terminada	Natalia	
Codificación	Terminada	Christian	
Codificación	Terminada	Natalia	
Codificación	Terminada	Natalia	
Pruebas	Terminada	Christian	
Reunión	Terminada	Natalia	

Tabla 12. Tabla de tareas para el Sprint 3

Para poder visualizar la evolución de cada sprint, es necesario definir tiempos para cada una de las tareas, con lo cual se podrá elaborar gráficos estadísticos que permitan observar el esfuerzo y el avance que se va dando durante la duración del sprint.

El siguiente cuadro (tabla 13) muestra las tareas por fecha de terminación y el porcentaje avanzado por cada día de tarea.

SPRINT	INICIO	DURACIÓN
3	10-nov-08	20

	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X
	10-nov	11-nov	12-nov	13-nov	14-nov	15-nov	16-nov	17-nov	18-nov	19-nov	20-nov	21-nov	22-nov	23-nov	24-nov	25-nov	26-nov	27-nov	28-nov	29-nov	30-nov	1-dic	2-dic	3-dic
Tareas pendientes	14	14	14	13	13	13		13	13	13	13	13	13		13	13	13	13	13	13		7	6	
Horas de trabajo pendientes	203	190	183	175	164	150		140	130	128	123	111	101		97	83	73	63	53	44		24	14	

Backlog ID	Tarea	Tipo	Estado	Responsable	ESFUERZO																					
	Creación del Product Backlog	Análisis	En curso	Natalia	19	18	17	16	15	14		13	12	11	10	9	8		7	6	5	4	3	2		1		
	Concepto de Jugabilidad	Prototipado	Terminada	Natalia	8	3	1																					
	Implementación de Motor Grafico	Pruebas	Terminada	Christian	19	19	16	12	11	8		7	7	6	6	5	5		5	5	3	1	1	1				
	Carga de escenarios y Personajes	Análisis	Terminada	Christian	7	7	7	6	6	4		4	4	4	3	3	3		1	1	1	1	1	1				
	Ambientación	Codificación	Terminada	Natalia	11	9	7	6	5	5		4	2	9	7	5	5		4	3	3	3	3	3				
	Skybox	Codificación	Terminada	Natalia	9	9	9	9	7	5		2	2	1	7	5	5		4	3	3	3	3	3				
	Terreno	Codificación	Terminada	Christian	16	16	16	16	15	15		14	9	6	6	3	3		4	3	3	3	3	3				
	Agua	Codificación	Terminada	Christian	10	7	10	7	10	5		4	4	4	2	2	2		4	3	3	3	3	3				
	Animación	Codificación	Terminada	Christian	24	24	24	24	24	24		24	24	24	23	22	17		17	14	10	10	10	8		5	1	
	Rigging	Codificación	En curso	Christian	18	18	18	18	18	18		18	18	18	18	18	16		16	14	14	14	10	6		6	4	
	Skinning	Codificación	En curso	Christian	13	13	13	13	12	13		13	13	12	12	13	13		13	11	11	11	8	8		5	4	
	Exportación	Codificación	Pendiente	Christian	10	10	10	15	10	10		10	10	10	8	7	7		7	7	6	1	1	1		4	3	
	Estabilización y Pruebas	Codificación	Pendiente	Natalia	19	18	17	16	15	14		13	12	11	10	9	8		7	6	5	4	3	2		1	1	
	Toma de Resultados	Codificación	Pendiente	Natalia	20	19	18	17	16	15		14	13	12	11	10	9		8	7	6	5	4	3		2	1	

Tabla 13. Esfuerzo en el Sprint 3.

A continuación los gráficos 188, 189 y 190 muestran los avances en el sprint 3 de Evolution, así como también el esfuerzo realizado en cada tarea.
Se define además las fechas de inicio y fin de las tareas y la duración total del sprint.

Proyecto
Evolution

SPRINT	INICIO	DÍAS
3	10-nov-08	20

Figura 188. Gráfico del Sprint 3.

El siguiente gráfico muestra el esfuerzo empleado por cada una de las tareas establecidas para el sprint 3.

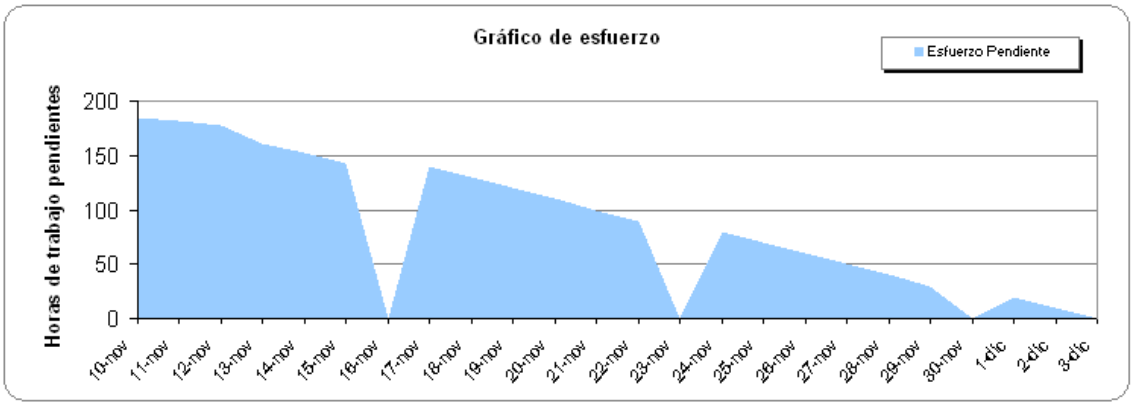


Figura 189. Gráfico del Esfuerzo en el Sprint 3.

El gráfico184 define las tareas pendientes y el avance de cada una de las tareas según las fechas definidas.

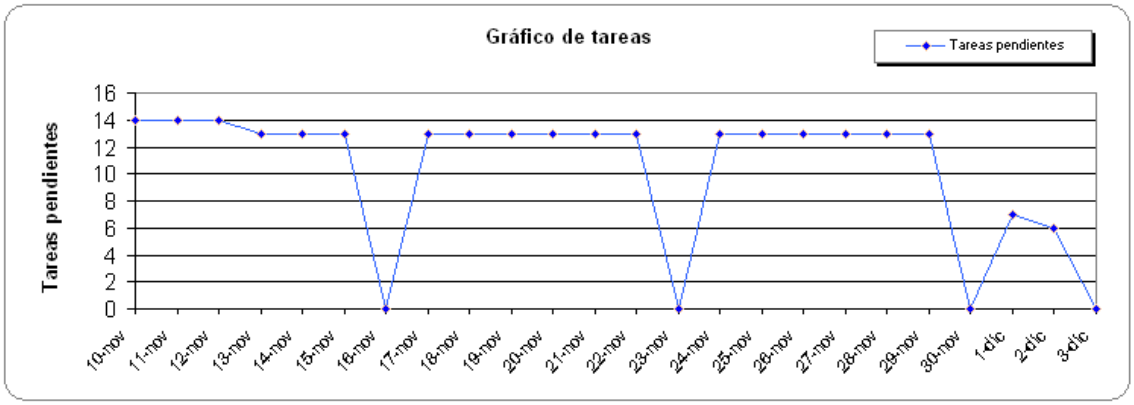


Figura 190. Gráfico que define el avance en las tareas del Sprint 3.

El gráfico 191 define el estado de las tareas según se va avanzando en el cronograma definido. Para realizar este gráfico es necesario realizar un cuadro en el que se muestre el estado de cada tarea. Cada número 1 indica el estado de la tarea en ese día.

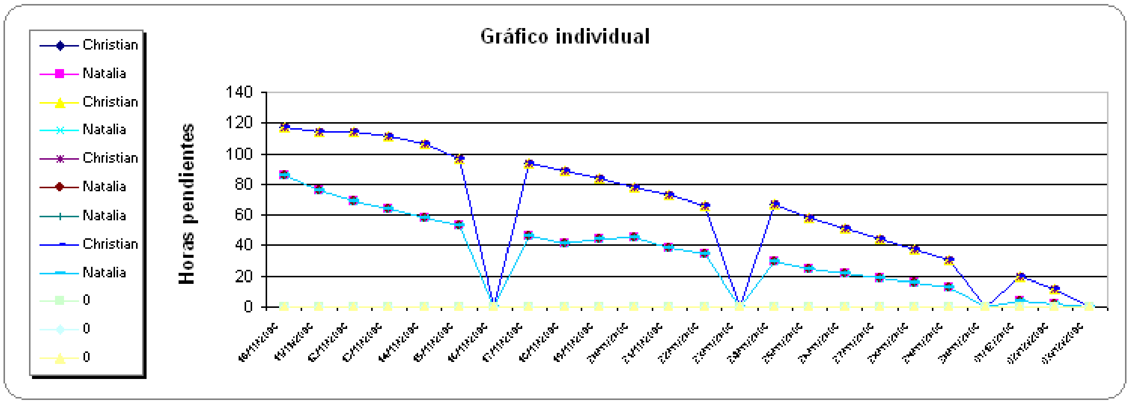


Gráfico 191. Gráfico Individual del Avance de las tareas en el Sprint 3.

	10-nov	11-nov	12-nov	13-nov	14-nov	15-nov	16-nov	17-nov	18-nov	19-nov	20-nov	21-nov	22-nov	23-nov	24-nov	25-nov	26-nov	27-nov	28-nov	29-nov	30-nov	1-dic	2-dic	3-dic
Christian	117	114	114	111	106	97		94	89	84	78	73	66		67	58	51	44	37	31		20	12	
Natalia	86	76	69	64	58	53		46	41	44	45	38	35		30	25	22	19	16	13		4	2	
Christian	117	114	114	111	106	97		94	89	84	78	73	66		67	58	51	44	37	31		20	12	
Natalia	86	76	69	64	58	53		46	41	44	45	38	35		30	25	22	19	16	13		4	2	
Christian	117	114	114	111	106	97		94	89	84	78	73	66		67	58	51	44	37	31		20	12	
Natalia	86	76	69	64	58	53		46	41	44	45	38	35		30	25	22	19	16	13		4	2	
Natalia	86	76	69	64	58	53		46	41	44	45	38	35		30	25	22	19	16	13		4	2	
Christian	117	114	114	111	106	97		94	89	84	78	73	66		67	58	51	44	37	31		20	12	
Natalia	86	76	69	64	58	53		46	41	44	45	38	35		30	25	22	19	16	13		4	2	

Gráfico 192. Gráfico de estado de las tareas en el Sprint 3.

4.3.2. Implantación de Modelos Matemáticos a la Estructura del Juego.

Con base en la teoría del Equilibrio de Nash y la Estrategia Maximin, Evolution aplica algunos modelos conocidos en videojuegos, principalmente para gráficas y para el realismo físico del juego, en donde se aplican conocimientos de física matemática, inteligencia artificial y geometría, en base a operaciones aritméticas, ecuaciones, trigonometría y vectores.

Tanto las matemáticas como la física, permiten hacer más realista el juego ya que ayudan a mejorar el desempeño del motor gráfico logrando mayor exactitud en la creación y movimiento de gráficos.

A continuación se va a detallar lo empleado en el desarrollo de Evolution tanto para el motor gráfico, el diseño de personajes y entorno del juego:

Matemática 3D

La matemática 3D abarca todo lo referente a lugares, distancias, ángulos precisos y matemática en espacios 3D. El uso más frecuente se encuentra dentro del denominado Sistema Cartesiano Coordenado aplicado en el diseño de personajes en 3D como se muestran en la figura 193, en la cual se observa un espacio 3D para el diseño de habitantes de Evolution.

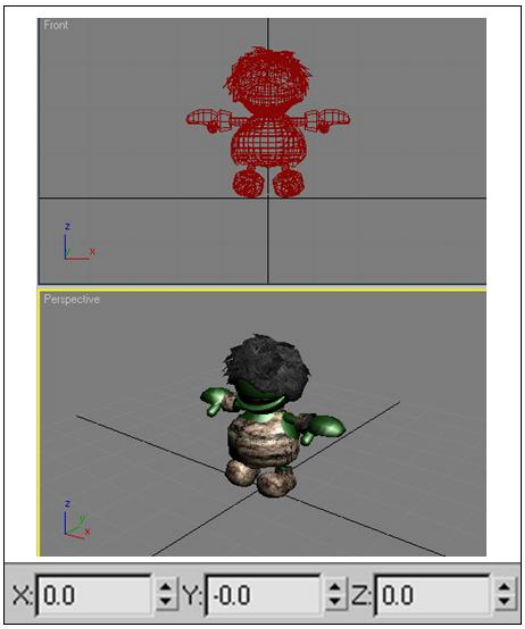


Figura 193. Espacio 3D.

Sistema Cartesiano de Coordenadas 3D

El sistema cartesiano coordenado constituye la base para poder realizar un diseño, este se debe ubicar dentro de los ejes coordenados, tal como es el típico ejemplo de un plano cartesiano en tercera dimensión (figura 194).

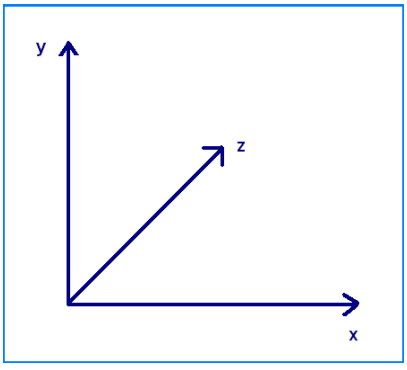


Figura 194. Plano 3D.

El plano cartesiano se lo utiliza dentro de la herramienta 3D Max. Éste se extiende sin limites, por lo cual, tiene un rango y dominio infinito.

En Evolution el plano en tercera dimensión (figura 195), es usado para graficar cada caracter que forma parte del juego. El plano cartesiano constituye el referente para el diseño de un carácter, ya que permite ubicar correctamente al personaje dentro del entorno donde se va a desarrollar, y además definir distancias y movimientos.

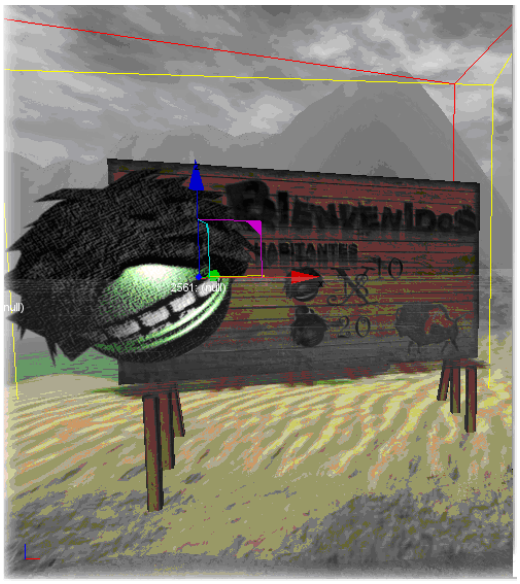


Figura 195. Vectores en Evolution

Vectores

Un vector es representado por una flecha dirigida hacia un punto en el espacio. Si es un vector de tres dimensiones se necesitarán valores para las coordenadas x, y, y z.

En Evolution el uso de vectores permite medir distancias y velocidades. Un vector permite obtener el cambio de posición de un carácter como se observa en la figura 196.

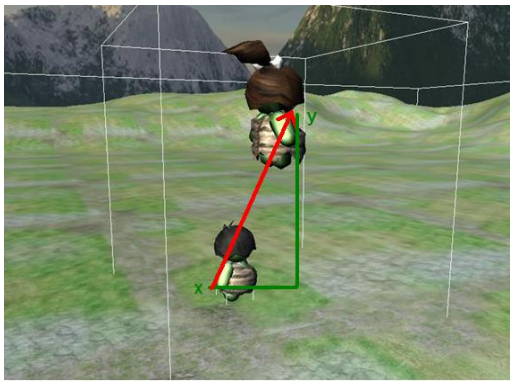


Figura 196. Cálculo de Distancias con Vectores

Los siguientes métodos permiten calcular la distacia entre un punto y otro por medio de vectores. La clase Vector3 contiene los datos del ángulo y largo del vector. Los dos métodos realizan lo mismo y únicamente difieren en la forma en que es retornado el resultado. El siguiente código realizado en C# usando la herramienta XNA muestra lo expuesto anteriormente:

```
public static float PointPointDistance(Vector3 pt1, Vector3 pt2)
{
   float num3 = pt1.X - pt2.X;
   float num2 = pt1.Y - pt2.Y;
   float num0 = pt1.Z - pt2.Z;
   float num4 = ((num3 * num3) + (num2 * num2)) + (num0 * num0);
   return (float)System.Math.Sqrt((double)num4);
}

public static void PointPointDistance(ref Vector3 pt1, ref Vector3 pt2, out float result)
{
   float num3 = pt1.X - pt2.X;
   float num2 = pt1.Y - pt2.Y;
   float num0 = pt1.Z - pt2.Z;
   float num4 = ((num3 * num3) + (num2 * num2)) + (num0 * num0);
   result = (float)System.Math.Sqrt((double)num4);
}
```

Únicamente con las coordenadas cartesianas se puede calcular la nueva posición del caracter en cada instante, usando las coordenadas (X,Y) del vector velocidad en las variables 'vel_x' y 'vel_y', y la posición del mismo en las variables 'pos_x' y 'pos_y'. se obtiene la nueva posición que tomará el personaje, para esto se usa el siguiente código realizado en la herramienta de desarrollo XNA en lenguaje C#:

```
pos_x += vel_x;
pos_y += vel_y;
```

Por otro lado, se utilizan también coordenadas polares, en el caso de que se tenga el ángulo y el largo del vector velocidad, con lo cual, se tiene la ventaja de controlar de mejor manera el movimiento que realice un caracter.

En Evolution, si el jugador se desplaza hacia la izquierda, el caracter se movilizará un espacio a esa dirección. Si se obtiene el dato del ángulo entero que forma el caracter en su desplazamiento, 'character_angle', se aplica el siguiente código en C# (usando XNA):

```
if (key[KEY_LEFT])
{
	character _angle -= 1; // dobla un grado hacia la izquierda
}
if (key[KEY_RIGHT])
{
	character _angle += 1; // dobla un grado hacia la derecha
}
```

Si es necesario incrementar la velocidad del carácter se puede incrementar el vector generado del personaje para que sus movimientos sean más rápidos.

Funciones Seno y Coseno

Las funciones seno y coseno permiten convertir las coordenadas polares en cartesianas.

En la figura 197, se puede observar que las dos funciones tienen la misma forma, la única diferencia es que la función coseno esta desplazada 90 grados regulares.

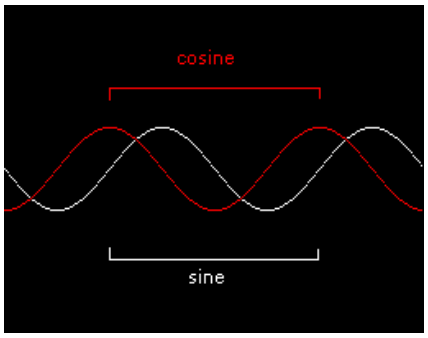


Figura 197. Gráfica de las funciones Seno y Coseno.

El seno se utiliza para calcular la coordenada Y del vector, y el coseno se utiliza para calcular la coordenada X. Las funciones seno (sin) y coseno (cos) solo admiten un parámetro: el ángulo.

Cada una de estas funciones retornan números entre -1 y 1, si se multiplica este número por el largo del vector, se obtienen las coordenadas cartesianas exactas

del vector. El código realizado en C# (usando XNA), que permite realizar esta conversión es el siguiente:

```
speed_x = speed_length * cos (speed_angle);
speed_y = speed_length * sin (speed_angle);
```

En resumen para calcular velocidades y movimientos en Evolution, únicamente se guarda el ángulo y el largo del vector velocidad. Se ajusta esto en función de la entrada que el jugador ingrese, es decir el movimiento que éste realice y se calculan las coordenadas para actualizar la posición del caracter.

Radianes

Los radianes permiten calcular el ángulo que limita un arco de circunferencia cuya longitud es igual a la del radio de la circunferencia. En Evolution los radianes permiten calcular el grado de un ángulo cuando un carácter realiza una vuelta o un movimiento en círculo.

El siguiente código permite calcular el número de grados con el número de radianes y viceversa:

```
degrees = radians * 180 / PI;
radians = degrees * PI / 180;
```

En la tabla siguiente se observan algunos valores claves de las funciones seno y coseno, en la cual las dos funciones llegan a su máximo o su mínimo en los múltiplos de 90 grados regulares.

Ángulos	**Radianes**	**Sin**	**Cos**
0	0	0	1
90	1/2 pi	1	0
180	pi	0	-1
270	3/2 pi	-1	0
360	2 pi	0	1

En Evolution un caracter es un círculo con una línea representando la dirección del caracter.

Para calcular la posición de un caracter es necesario conocer el vector y la velocidad a la que se desplaza. El siguiente método permite conocer la posición en la que se encuentra, para lo cual realiza llamadas a los métodos *CalculatePosibleLocation* y *MoveTo.*

```
internal void GoToPosition(Vector3 Location, inhabitantVelocity iVelocity, EvolutionRandom rand,
float gameTime)
    {
        if (isActive)
        {
            CalculatePosibleLocation(Location, rand);
            MoveTo(disasterPosition, iVelocity, 10, gameTime, rand);
        }
    }
```

El método *CalculatePosibleLocation* permite ubicar el sitio donde se está produciendo un desastre y puede haber disminución de la población del juego como se muestra en el siguiente código.

```
internal void CalculatePosibleLocation(Vector3 possiblePoint, EvolutionRandom rand)
 {
     if (!isOnDisaster)
     {
        const float radius = 5;
        double angle = (rand.NextDouble()) * Math.PI * 2;
        float x = (float)Math.Cos(angle);
        float z = (float)Math.Sin(angle);
        disasterPosition = new Vector3((x * radius), 7.5f, ((possiblePoint.Z + z) * radius));
         disasterPosition = disasterPosition - possiblePoint;
         disasterPosition = new Vector3(disasterPosition.X, 10.5f, disasterPosition.Z);
          isOnDisaster = true;
          isOnChangeVelocity = true;
      }
 }
```

Una vez conocida la ubicación de desastre el personaje en peligro tiene que cambiar de movimiento para evitar ser alcanzado por el desastre, para lo cual utiliza el método *MoveTo* en el siguiente código.

```
public void MoveTo(Vector3 destination, inhabitantVelocity iVelocity, float radius, float gameTime,
EvolutionRandom rand)
    {
        Vector3 difference = destination - body.Position;
        if (isActive)
```

```
            if ((body.Position != destination) && difference.Length() >=
radius)//(Math.Abs(difference.X) >= radius * 2) && (Math.Abs(difference.Z) >= radius * 2))
            {
                body.Immovable = false;
                float speedScaleGlobal = 0.1f;
                if (isOnChangeVelocity)
                {
                    speedVal = rand.Next((int)iVelocity);
                    isOnChangeVelocity = false;
                }
                switch (speedVal)
                {
                    case (int)inhabitantVelocity.stunned:
                        speedScaleGlobal = 0f;
                        break;
                    case (int)inhabitantVelocity.normal:
                        speedScaleGlobal = 1;
                        break;
                    case (int)inhabitantVelocity.affraid:
                        speedScaleGlobal = 0.3f;
                        break;
                    case (int)inhabitantVelocity.horrorized:
                        speedScaleGlobal = 0.5f;
                        break;
                    case (int)inhabitantVelocity.almostdeath:
                        speedScaleGlobal = 0.6f;
                        break;
                    default:
                        break;
                }
                if (difference.Length() != 0)
                    difference.Normalize();
                difference = difference * gameTime * walkSpeed * speedScaleGlobal;
                body.Velocity = new Vector3(difference.X, body.Velocity.Y, difference.Z);
            }
            else
            {
                body.Velocity = Vector3.Zero;
                canRoam = true;
                isFindingFood = true;
```

```
            isGoingHome = false;
        }
    }
```

Este método es usado también para movilizar al habitante cuando ha encontrado su pareja como se muestra en el siguiente código:

```
/// <param name="oppositeSex">An inhabitant in the radius</param>
/// <param name="gameTime">The time passed since last call</param>
/// <param name="returnTo">Where to go now?</param>
public void SnuSnu(Inhabitant oppositeSex, float gameTime, Vector3 returnTo)
 {
   this.MoveTo(returnTo, 10, gameTime);
   oppositeSex.MoveTo(returnTo, 10, gameTime);
   if (OnSoundEvent != null)
      OnSoundEvent("snusnus");
 }
```

La velocidad del caracter está representada por el ángulo y el largo. Si el jugador se desplaza hacia arriba, el largo del vector velocidad se incrementa; si el jugador se desplaza hacia abajo, el largo del vector velocidad se decrementa, el ángulo cambia si el jugador se desplaza a la izquierda o derecha.

Después de que la dirección y la velocidad han sido ajustadas, las coordenadas cartesianas 'vel_x' y 'vel_y' son calculadas con sin () y cos (). En cada iteración del bucle, estas coordenadas son sumadas a las coordenadas del carácter, como se muestra en el siguiente código:

```
public void FaceDirection(Vector3 target, float gameTime)
 {
    body.Orientation = Matrix.CreateRotationY(-gameTime *
(float)Math.Acos((double)MathHelper.ToRadians(Vector3.Dot(body.Orientation.Right, target))));
 }
```

En Evolution se emplea el uso de estas funciones además para mover los planetas de la pantalla para seleccionar los niveles que tiene el juego. El planeta va a ser representado por un pequeño punto, que se moverá alrededor de un círculo. A continuación se tiene el siguiente código:

```
    Vector3 RandomPointOnCircle()
    {
```

```
        #region Implementation
        const float radius = 38;
        const float height = 45;
        double angle = random.NextDouble() * Math.PI * 2;

        float length_x = (float)Math.Cos(angle);
        float length_y = (float)Math.Sin(angle);

        return new Vector3(length_x * radius, length_y * radius + height, 0);
        #endregion
    }
```

Al variar los valores de 'length_x' y 'length_y', el planeta no se moverá formando un círculo, sino que se moverá trazando una elipse.

Para calcular las coordenadas en la parte superior de un círculo, por ejemplo una cabeza, se tiene que la coordenada X es 0 y la coordenada Y es igual que al radio del círculo, tomando en cuenta el valor negativo en función de las coordenadas de la pantalla.

Para dibujar un píxel en esa coordenada, se debe calcular la distancia al centro con el teorema de Pitágoras, y se dibuja el píxel cuya distancia desde el centro se aproxime cada vez más al radio del círculo.

- Teorema de Pitágoras (figura 198): a^2 + b^2 = c^2, es decir:

```
length = sqrt (x * x + y * y)
```

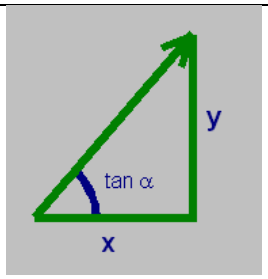


Figura 198. Teorema de Pitágoras.

Para calcular el ángulo, se tiene que calcular la proporción entre Y y X como se muestra a continuación:

```
tan (angle) = y / x
```

Para el próximo píxel se puede ir un píxel a la derecha, o un píxel abajo y luego un píxel a la derecha, como muestra la figura 199.

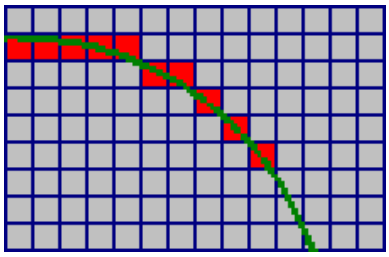


Figura 199. Graficando Píxeles.

Mapas de Bit y Rotación

Para la rotación de *sprites*, se realiza una función para iterar a través de todos los píxeles del bitmap, para evitar que un píxel se pinte en pantalla dos veces o más. En la posición (0,0) de la pantalla se ubica la posición (0,0) del *sprite*. Luego, en la posición (1,0) el siguiente píxel que se pinte depende del ángulo que se va a rotar. Si se rota un ángulo a 0 grados, se ubica el píxel (1,0) del *sprite* en esa posición. Si se rota 270 grados, el píxel (0,1) debe estar en esa posición. Para cualquier ángulo se utiliza el siguiente código:

```
sprite_x = cos (angle);
sprite_y = sin (angle);
```

La siguiente posición será:

```
sprite_x = 2 * cos (angle);
sprite_y = 2 * sin (angle);
```

Este procedimiento se repite sucesivamente cuando se mueva un píxel a la derecha en el destino.

Un ejemplo de lo expuesto anteriormente se observa en la figura 200.

Figura 200. Pixelado isla del juego

Para poder rotar un *sprite* se utiliza el siguiente método que permite movilizarse a traves de los píxeles de un bitmap.

```
void my_rotate_sprite (BITMAP *dest_bmp, BITMAP *src_bmp,
   fixed angle, fixed scale)
{
   // current position in the source bitmap
   fixed src_x, src_y;
    // current position in the destination bitmap
   int dest_x, dest_y;
    // src_x and src_y will change each time by dx and dy
   fixed dx, dy;
    // src_x and src_y will be initialized to start_x and start_y  at the beginning of each new line
   fixed start_x = 0, start_y = 0;
    // We create a bit mask to make sure x and y are in bounds.
   // Unexpected things will happen
   // if the width or height are not powers of 2.
   int x_mask = src_bmp->w - 1;
   int y_mask = src_bmp->h - 1;

   // calculate increments for the coordinates in the source bitmap
   // for when we move right one pixel on the destination bitmap
   dx = fmul (fcos (angle), scale);
   dy = fmul (fsin (angle), scale);

   for (dest_y = 0; dest_y < dest_bmp->h; dest_y++)
   {
      // set the position in the source bitmap to the
      // beginning of this line
      src_x = start_x;
      src_y = start_y;


      for (dest_x = 0; dest_x < dest_bmp->w; dest_x++)
      {
         // Copy a pixel.
         // This can be optimized a lot by using
         // direct bitmap access.
         putpixel (dest_bmp, dest_x, dest_y,
            getpixel (src_bmp,
```

```
                fixtoi (src_x) & x_mask,
                fixtoi (src_y) & y_mask));

            // advance the position in the source bitmap
            src_x += dx;
            src_y += dy;
        }

        // for the next line we have a different starting position
        start_x -= dy;
        start_y += dx;
    }
}
```

Rotación de un Carácter

Para la rotación de un caracter se crea una matriz de rotación. Las Matrices de rotación se utilizan frecuentemente en el mundo de gráficos 3D, ya que proveen una manera de rotar un vector sin convertirlo a coordenadas polares. El código, realizado en C# (utilizando XNA) es el siguiente:

```
new_x = x * cos (angle) - y * sin (angle)
new_y = x * sin (angle) + y * cos (angle)
```

Siendo 'angle' el ángulo con el que se va a rotar el vector, 'x' e 'y' son las coordenadas antiguas del vector, y 'new_x' y 'new_y' son las nuevas coordenadas del vector.

Para la rotación con ángulos de 90 grados el procedimiento es diferente. Por ejemplo, si se tiene un punto de coordenadas (4,8) y se lo va a rotar a 90 grados alrededor del origen se emplean las siguientes fórmulas:

```
new_x = x * cos (90) - y * sin (90)
new_y = x * sin (90) + y * cos (90)
```

Donde:

- cos (90) = 0
- sin (90) = 1

Con lo cual las nuevas coordenadas son (-8, 4). Si se desea rotar el mismo punto (4, 8) a 180 grados alrededor del origen, se emplean las siguientes fórmulas:

```
new_x = x * cos (180) - y * sin (180);
new_y = x * sin (180) + y * cos (180);
```

O en forma simplificada:

```
new_x = -x
new_y = -y
```

Donde se obtiene la nueva coordenada (-4, -8).

En la siguiente tabla se observan las rotaciones para los ángulos de 90, 180 y 270 grados.

	90 Degrees	**180 Degrees**	**270 Degrees**	**360 Degrees**
new x value	-y	-x	y	x
new y value	x	-y	-x	y

Colisiones

En Evolution los habitantes del juego ejecutan algunos movimientos en su desplazamiento y sobre la plataforma del juego puede colisionar con las paredes del nivel.

Para aplicar colisiones entre un personaje y la estructura del terreno del nivel se determina la distancia entre el personaje y la próxima plataforma o pared.

La figura 201 muestra estas distancias. Si el personaje está caminando hacia otro objeto es necesario que este reconozca dos aspectos importantes: primero, si el objeto es peligroso para éste, tratar de huir en dirección contraria a este objeto, caso contrario detenerse antes de que pueda colisionar con el objeto y cambiar de dirección. De igual manera si el personaje camina sobre una superficie, se debe detener en el suelo e indicarle que no puede avanzar en cuanto a profundidad.

Figura 201. Colisiones de Personajes

Todo avance del personaje puede superar el píxel, es decir, en un instante dado el personaje podría necesitar avanzar unos 20 pixeles a la vez. Si en ese trayecto se encuentra una plataforma, se debe detener el movimiento y avanzar menos de lo que se esperaba recorrer.

Para implementar ese control se realizó una función que determine, dada una coordenada y rango, la distancia hasta la próxima plataforma:

```
public int GetDistSuelo (int x, int y, int rango)
{
        int cont;
         for (cont = 0; cont < rango; cont ++)
        {
                if (es_suelo(x, y + cont))
                        return cont;
        }
         return rango; // no encuentra un suelo
}
```

El objetivo de este método es buscar, punto a punto, la existencia de un suelo sin exceder el rango (tercer parámetro). Si encuentra el suelo retorna la distancia al mismo, en caso contrario retorna el valor de rango. Los dos primeros parámetros del método corresponden a una coordenada de la pantalla.

La función *es_suelo* simplemente verifica si un bloque es sólido o no.

```
public bool es_suelo (int x, int y)
{
         if (y == 0 && es_solido(x, y))
                return true;
        else
                return false;
}
```

```
public bool es_solido (int x, int y)
{
        return tiles[y / 16][x / 16].solido;
}
```

Para utilizar estos métodos es necesario primero tener una referencia al nivel donde se encuentra el personaje.

Formas Geométricas

El dibujo de formas geométricas es una de las utilidades mas básicas del motor, que para esta versión del proyecto únicamente se ha implementado el dibujo de cuadrados con bordes redondeados.

La implementación debe soportar el tamaño, el color de fondo, el color de línea y la posición de dibujo.

- **Frames por segundo (FPS):** Esta característica permite habilitar de manera automática la impresión en pantalla de la cantidad de cuadros por segundo, lo cual es un dato importante y muy útil al momento de realizar mediciones de rendimiento.

El código que permite evaluar los frames por segundo es:

```
int frameRate = 0;
int frameCounter = 0;
TimeSpan elapsedTime = TimeSpan.Zero;
public override void Update(GameTime gameTime)
{
    elapsedTime += gameTime.ElapsedGameTime;

    if (elapsedTime > TimeSpan.FromSeconds(1))
    {
        elapsedTime -= TimeSpan.FromSeconds(1);
        frameRate = frameCounter;
        frameCounter = 0;
    }
}
```

Física

Evolution implementa un motor de física, el cual permite crear simulaciones que se asemejen a la realidad y crear efectos especiales empleando los principios fundamentales de la dinámica.

Mientras el juego se está ejecutando, el motor de física consulta la posición, velocidad y la aceleración de un caracter, al mismo tiempo, se calcula la distancia entre el caracter y los obstáculos de camino. Cada movimiento o acción es calculado precisamente por el motor. Estos cálculos son basados en la aplicación de los principios del impulso, al instante en que se presenta una fuerza en función de la velocidad. Como en una colisión entre el caracter y otro objeto, se obtiene la aceleración, la velocidad y la posición de los objetos involucrados en base a los valores previos de la aceleración, velocidad y posición, respectivamente, de modo que todo cálculo se basa en el antecedente previo.

En el siguiente código se muestra la forma en que se actualiza la posición y orientación de un caracter con la actual velocidad y la conservación del impulso.

```
/// <summary>
/// implementation updates the position/orientation with the
/// current velocties.
/// </summary>
/// <param name="dt"></param>
public void UpdatePosition(float dt)
{
  if (immovable || !IsActive)
    return;
  Vector3 angMomBefore = Vector3.Transform(transformRate.AngularVelocity, worldInertia);
  transform.ApplyTransformRate(transformRate, dt);
  // Matrix Check
  invOrientation = Matrix.Transpose(transform.Orientation);
  // recalculate the world inertia
  worldInvInertia = invOrientation * bodyInvInertia * transform.Orientation;
  worldInertia = invOrientation * bodyInertia * transform.Orientation;
  // conservation of momentum
  transformRate.AngularVelocity = Vector3.Transform(angMomBefore, worldInvInertia);
  if (this.CollisionSkin != null)
    CollisionSkin.SetTransform(oldTransform, transform);
 }
```

Además utilizando fórmulas físicas el motor calcula la gravedad de las partículas, como se muestra en el siguiente código:

```
public Vector3 Gravity
```

```
    {
        get { return this.gravity; }
        set
        {
            // Have a look here
            this.gravity = value;
            this.gravityMagnitude = value.Length();
            if (value.X == value.Y && value.Y == value.Z)
                gravityAxis = -1;
               gravityAxis = 0;
            if (System.Math.Abs(value.Y) > System.Math.Abs(value.X))
                gravityAxis = 1;
            float[] gravity = new float[3] { value.X, value.Y,value.Z};

            if (System.Math.Abs(value.Z) > System.Math.Abs(gravity[gravityAxis]))
                gravityAxis = 2;
        }
    }
```

Caracteres, Stages e Interacción

- **Personajes**

Un *sprite* es una imagen o animación que se adjunta a una escena o stage, como se muestra en la figura 202, donde como *sprite* se tiene al dinosaurio y la escena del juego lo muestra caminando sobre la isla.

Figura 202. Sprite y Stage en Evolution

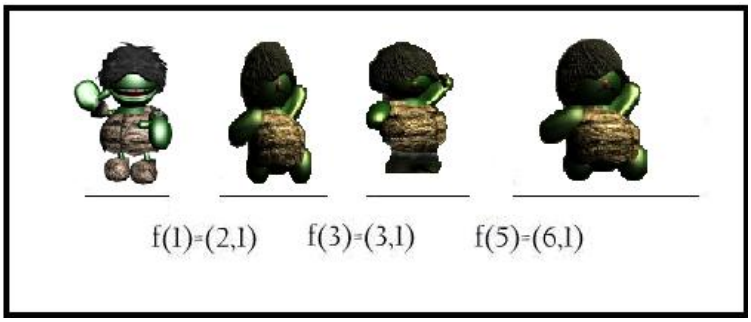
Un *sprite* tiene asignado un punto central, el cual le permite desplazarse por una curva, como se muestra en la figura 203.

Figura 203. Asignación del punto central al carácter

Aplicando las fórmulas de desplazamiento rectilíneo se obtiene el movimiento del sprite hacia la derecha y hacia la izquierda respectivamente, como se muestra en la figura 204.

$$F(t) = (t + \alpha, \beta)$$

$$F^*(t) = (-t + \alpha, \beta)$$



$$f(t) = (t + 1, 1)$$

Figura 204. Movimiento hacia la derecha

Modelado de Stage

Los Stages o Escenas en Evolution están conformados en su totalidad por caracteres, los cuales se clasifican de 2 formas:

1) De Ornato: Como son las plantas, los arbustos y los rótulos (figura 205).

Figura 205. Stages de Ornato.

2) De Interacción: Como la cueva en la que los personajes ingresan (figura 206).

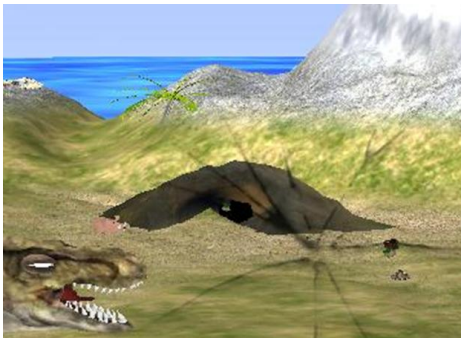


Figura 206. Stage de Iteración.

La forma de integrar los caracteres al escenario se realiza seleccionando en el plano cartesiano un punto central, donde después se adjunta la imagen, con la diferencia de que estos puntos no se mueven a lo largo de una curva.

Posteriormente se utiliza el concepto de aureola de acción que son los puntos que rodean al punto central de un personaje (figura 207).

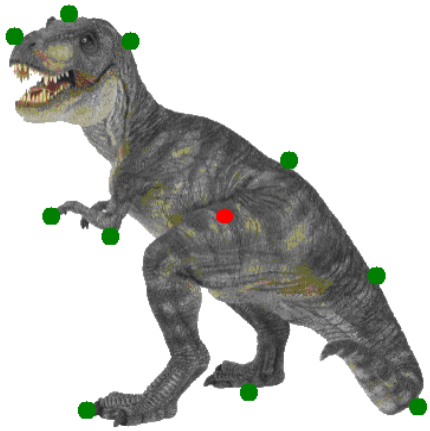


Figura 207. Personaje con aureola de acción

Usualmente el punto central se designa como M_5 y la aureola de acción junto a este punto se denota con una matriz de la forma (figura 208).

$$A_{Personaje} = \begin{pmatrix} P1 & P2 & P3 \\ P4 & P5 & P6 \\ P7 & P8 & P9 \end{pmatrix}$$

Figura 208. Matriz para definir la Aureola de Acción.

La aureola permite darle la posibilidad al personaje de que se relacione con el escenario (stage) y con otros personajes (enemigos). Para realizar esto se

utilizan sentencias como if, else, and, or, etc., y el concepto de distancia entre dos puntos.

La distancia entre dos puntos a= (x_0,y_0) y b=(x_1,y_1), esta dada por:

$$d(a,b) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Con esto se definirán los eventos como la muerte del un habitante de Evolution, la captura de un animal por parte de un personaje como alimento y la captura de un habitante por parte de su enemigo, con lo cual se tendrá la siguiente estructura: ***If*** Condicion de Distancia ***then*** evento 1 ***and/or*** evento 2 ***else*** evento 3.

En Evolution se define el evento muerte de un personaje de la siguiente manera:

- Se tienen las matrices aureolas de los dos personajes: habitante y dinosaurio.

Matriz de Habitante **Matriz de Dinosaurio**

$$M_{Habitante} = \begin{pmatrix} h1 & h2 & h3 \\ h4 & h5 & h6 \\ h7 & h8 & h9 \end{pmatrix} \qquad M_{Dinosaurio} = \begin{pmatrix} d1 & d2 & d3 \\ d4 & d5 & d6 \\ d7 & d8 & d9 \end{pmatrix}$$

Entonces:

Si $D(d_i,h_i) = 0/i = \{1,2,3,4,5,6,7,8,9\}$ then

$V_{(x,y)} = (x,10t)$ (for) hasta que $t = -5$ and $H_5=0$.

En la siguiente figura 209, se muestra el resultado de este bucle:

Figura 209. Muerte del habitante.

Interpolación de un Personaje

Evolution emplea la técnica de interpolación lineal en la que se modelará a los personajes a partir de un dibujo (figura 210).

Figura 210. Interpolación de un personaje.

Como primer paso se selecciona una cantidad de puntos conocidos como nodos de interpolación y se usa Interpolantes del tipo:

$$f(x) = \begin{cases} \frac{y_2-y_1}{x_2-x_1}(x-x_1)+y_1 & \text{en } [x_1,x_2] \\ \frac{y_3-y_2}{x_3-x_2}(x-x_2)+y_2 & \text{en } [x_2,x_3] \\ \vdots & \vdots \\ \frac{y_n-y_{n-1}}{x_n-x_{n-1}}(x-x_{n-1})+y_{n-1} & \text{en } [x_{n-1},x_n] \end{cases}$$

La imagen a formarse es la siguiente:

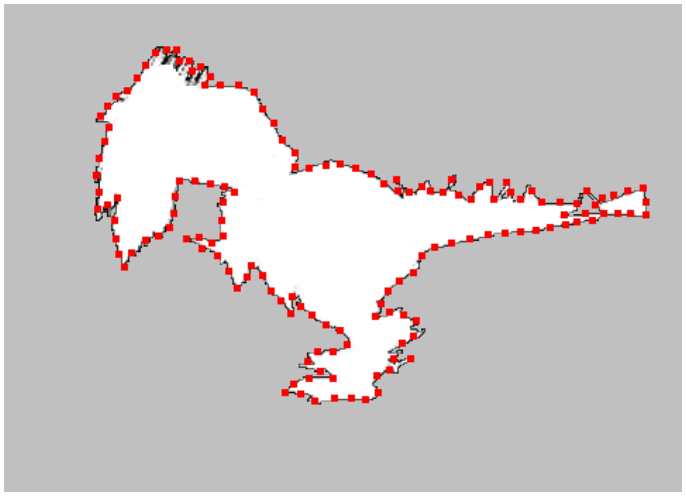


Figura 211. Imagen interpolada.

El siguiente paso consiste en rellenar al personaje con los colores correspondientes y anexarle ciertos detalles ornamentales, obteniendo con esto el resultado final de un dinosaurio digital.

Figura 212. Relleno de imagen interpolada.

Equilibrio de Nash y Estrategia Maximin

La teoría del Equilibrio de Nash se aplica en Evolution para establecer un conjunto de estrategias por el habitante del juego frente a un peligro inminente. Mientras que la estrategia maximin se utiliza en Evolution para definir una matriz de movimientos de cada habitante según los peligros a los que se vea afectado.

En el concepto de equilibrio de Nash es fundamental que los personajes tengan racionalidad, es decir, si un habitante sospechara que su adversario no se comporta racionalmente, podría tener sentido que adoptara una estrategia *maximin*, esto es, aquella en la que se maximiza la ganancia mínima que puede obtenerse; que aplicado al juego indica que el habitante tiene más posibilidades de huir de su enemigo.

En Evolution cada habitante dispone de tres estrategias posibles para huir de los peligros como: el ataque de un dinosaurio, una erupción volcánica, una inundación o un terremoto.

Las posibles estrategias que el jugador puede tomar son:

A= Huir hacia su derecha, en dirección contraria a su atacante.

B= Huir hacia su izquierda, en dirección contraria a su atacante.

C=Huir hacia atrás, en dirección contraria a su atacante.

Los premios o pagos consisten en la asignación de los movimientos según estén disponibles, los cuales se diseñan en una tabla llamada matriz de pagos.

Matriz De Pagos:

Matriz de Habitantes:

A	B	C
Huir derecha	Huir Izquierda	Huir Atrás
10mtrs.	5 mtrs.	15mtrs.

Matriz de Atacantes:

A	B	C
Atacar derecha	Atacar Izquierda	Atacar al frente
6mtrs.	15mtrs.	2mtrs.

Para descubrir qué estrategia es más conveniente se analiza la matriz de pagos y la acción que ejecutará el atacante.

Una forma de analizar el juego para tomar una decisión consiste en mirar cuál es el mínimo resultado que puede obtener con cada una de las estrategias.

Por ejemplo. Si el atacante juega con la estrategia C, entonces el habitante decidirá la estrategia A o C, las mismas que indican la distancia máxima a la que se encontrará si toma esa decisión. Por tanto la ganancia es para el habitante ya que éste no será alcanzado por su atacante.

La estrategia *maximin* consiste en elegir la tarjeta C ya que esa estrategia garantiza que, el atacante estará más distante que seleccionando otra estrategia.

Algoritmo de Línea de Visión

Evolution implementa el algoritmo de línea de visión haciendo uso de un mapa de bloques accesible vía filas y columnas y con dos tipos de bloques (colisionables y no colisionables).

Mapa.- El mapa (figura 207) se representa por medio de un array de enteros de dos dimensiones, una columna y una fila.

```
int map[COL][ROW];
```

Donde *col* es la cantidad total de columnas y *row* es la cantidad total de filas. Dentro del mapa se representan dos posiciones que corresponden a la posición del objeto A y el objeto B. El algoritmo permite verificar si el camino de A hacia B está libre de obstáculos, es decir libre de bloques colisionables.

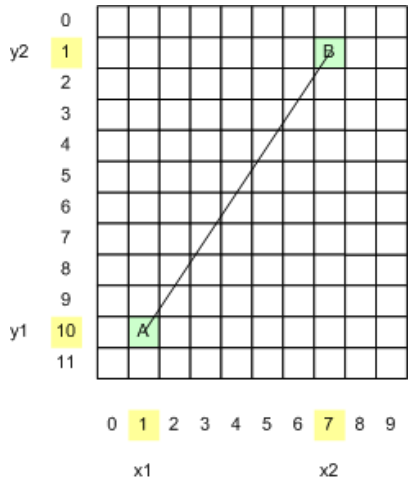


Figura 207. Mapa de Bloques.

La función que implementa el algoritmo requerido para determinar si el objeto A puede visualizar al objeto B está determinada por el siguiente método:

```
public bool LineOfSight (int x1, int x2, int y1, int y2)
{
  float col = x1;
  float row = y1;
  float slopeRow = GetSlopeRow(x1, x2, y1, y2);
  float slopeCol = GetSlopeCol(x1, x2, y1, y2);
  // Voy desde (x1, y1) a (x2, y2)
  if(fabs(slopeRow) == 1)
   {
          while (row ¡= y2)
          {
      // Si algún bloque que esté en camino es colisión, los objetos no se ven.
      if (map[floor(col)][floor(row)] == BLOCK_COLLISION)
        return false;
           // Incremento la fila y la columna
        row += slopeRow;
```

```
                col += slopeCol;
      }
   }
  else
   {
      while (col != x2)
       {
         // Si algún bloque que esté en camino es colisión los objetos no se ven
         if (map[floor(col)][floor(row)] == BLOCK_COLLISION)
                        return false;
          // Incremento la fila y la columna
          row += slopeRow;
          col += slopeCol;
       }
 }
return true;
}
```

Para desarrollar este código, la idea es partir del punto (x1,y1) y desplazarse hasta el punto (x2,y2). Por lo tanto es necesario utilizar una sentencia de iteración condicional hasta llegar al punto B. En cada iteración se verifica si la posición calculada de fila y columna dentro del mapa es un bloque de colisión; de ser así el método retorna false.

Evolution implementa además dos métodos llamados *GetSlopeRow* y *GetSlopeCol* que permiten calcular el incremento se deberá acumular en la fila y la columna.

```
float GetSlopeRow(int x1, int x2, int y1, int y2
{
            if (fabs((float) y2-y1) > fabs((float) x2-x1))
                        return (y2>y1 ? 1 : -1);
            else
            {
                        float slope = fabs((y2-y1) / (float) (x2-x1));
                        slope *= (y2>y1 ? 1 : -1);
                        return slope;
            }
}
```

```
float GetSlopeCol(int x1, int x2, int y1, int y2)
{
        if (fabs((float) x2-x1) > fabs((float) y2-y1))
                return (x2>x1 ? 1 : -1);
        else
        {
           float slope = fabs((x2-x1) / (float) (y2-y1));
           slope *= (x2>x1  ? 1 : -1);
           return slope;
        }
}
```

4.3.3. Inteligencia Artificial de Personajes

Un personaje del videojuego realiza varios movimientos simples como: caminar, cazar su alimento y acelerar su paso para huir de un atacante.

Mediante la Inteligencia Artificial se puede gestionar un conjunto de movimientos mediante autómatas finitos.

Estados

Un estado representa el comportamiento del personaje en un momento dado. Cada estado reacciona de manera diferente a los eventos (como la pulsación de una tecla) y está asociado a una animación diferente como se muestra en la figura 213.

Figura 213. Estados de un Personaje.

En este caso, el personaje de la figura cuenta con el estado “parado”, ya que tiene una animación asociada y reacciona de manera diferente ante un evento como “caminar”, en comparación a otro estado como “capturacomida”.

Autómata

Un autómata finito es un modelo lógico asociado a una serie de propiedades, entre ellas, un conjunto de estados, un alfabeto y una relación de transición.

Interpretación del Autómata

Un autómata constituye una máquina que se encuentra en un estado particular, y que ante determinados eventos cambia de un estado a otro.

En Evolution cada personaje actuará como un autómata como se muestra en la figura 214.

Figura 214. Autómata

Cuando comienza el juego, cada uno de los habitantes se encuentra en posición caminando (un estado), su estado se modificará de acuerdo a la acción que realice el jugador. Por ejemplo si es necesario que la tribu crezca en población, el habitante llevará a otra habitante a la cueva para que nazca un nuevo habitante en la tribu, así el nuevo estado será “llevarhabitante”, y en él se tendrá una serie de posibilidades nuevas como realizar una acción “caminarderecha”.

Este criterio se puede apreciar fácilmente mediante un diagrama de transiciones:

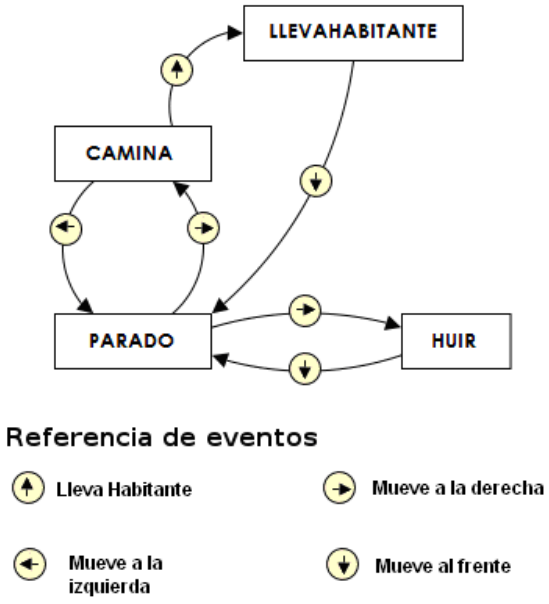


Figura 215. Estado de los habitantes.

De la figura se pueden deducir algunas características:

1.- El autómata no puede estar en más de un estado al mismo tiempo.

2.- Los eventos relacionan a los estados entre sí.

3.- No todos los eventos implican un cambio de estado.

Implementando un Gestor de Estados

Existen varias formas de implementar un gestor de estados partiendo de un autómata. Si bien el modelo original será idéntico, cada implementación difiere en eficiencia y complejidad.

En Evolution, se puede manejar al personaje con los direccionales del teclado. En la pantalla principal se muestran mensajes al jugador si el habitante ha realizado acciones como:

- Ingresar en la cueva para crear un nuevo habitante.
- Un habitante ha muerto.
- Provisiones de alimento.
- Número de habitantes actuales.

El personaje almacena su estado actual junto con el resto de sus propiedades dentro de la estructura habitante:

```
enum estado { parado, camina, llevahabitante [...] };
 class Habitant
{
       enum estado;
       int x;
       int y;
}
```

El manejo general del personaje consiste en llamar de manera periódica al método:

```
void Update_Habitant (class actor obj)
```

Este método selecciona, en base al estado del actor, que función se debe procesar para representar el comportamiento en ese instante.

Así, cada estado se encuentra asociado a un método independiente que recibe los datos del personaje y las teclas pulsadas:

```
void Habitant_State_StandUp (Habitant obj, int key);
```

```
void Habitant_State_Walk (Habitant obj, int key);
void Habitant_State_CarryHabitant (Habitant obj, int key);
```

Diagrama de Transiciones

El siguiente gráfico muestra el diagrama de transiciones para cada estado en el que pueda hallarse el habitante.

El objetivo principal de los estados es diseñar un autómata que cubra los requerimientos; a partir de ahí, la implementación posterior es muy sencilla y fácil de modificar.

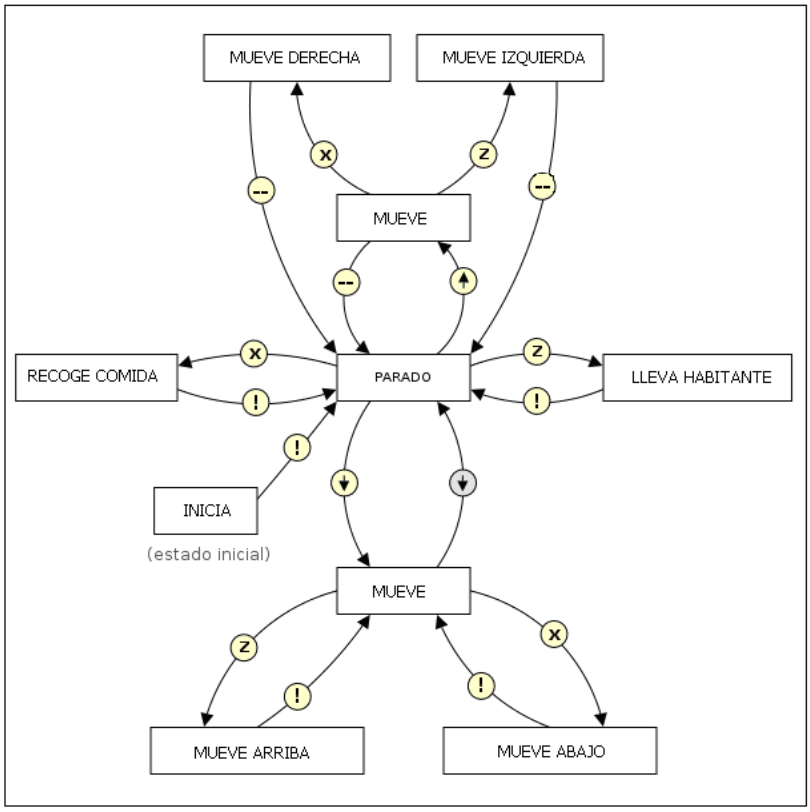


Figura 216. Estados de un habitante de Evolution

Figura 217. Significado de los estados.

Flocking

Permite el movimiento coordinado de grupos (*flocks*) de entidades. Este movimiento coordinado puede apreciarse en acciones que se realiza en la naturaleza.

El modelo posee ciertas reglas que hay que respetar en la implementación, es decir, flocking no es sólo intentar simular el movimiento de entidades coordinadas sino además la aplicación de ciertas reglas que aseguren las condiciones reales de este fenómeno. No puede existir una inteligencia superior que gobierne todas las entidades, cada entidad se mueve de modo independiente según las reglas del modelo y su ponderación respectiva.

De este modo la simulación que Evolution aplica se asemejará al comportamiento de las entidades que se observen en la naturaleza, las cuales implementan mediante un patrón de comportamiento.

Las entidades mantienen una cohesión pero al mismo tiempo mantienen una distancia mínima entre ellas, no se chocan entre sí ni contra obstáculos y mantienen una dirección común.

En Evolution los comportamientos se obtienen de la aplicación de reglas, a la que se sea sometido el caracter y que varíen su velocidad, aceleración y dirección, lo cual provoca en el personaje la falta de predictibilidad en su comportamiento, durante un lapso de tiempo.

Las reglas son aplicadas para definir el comportamiento de los enemigos de los habitantes del juego, y estas son:

Regla de Separación: Es la distancia mínima que mantienen los caracteres con sus vecinos. Esta regla se la implementa usando el código de distancias entre dos puntos.

Alineamiento: Los habitantes mantienen una dirección común entre ellos. Evolution implementa el siguiente método para obtener la dirección del grupo y asignar el estado obtenido al actual.

```
public void StoreState()
{
  storedTransform = transform;
  storedTransformRate = transformRate;
}
```

```
public void RestoreState()
{
  transform = storedTransform;
  transformRate = storedTransformRate;
  Matrix.Transpose(ref transform.Orientation, out invOrientation);
  Matrix.Multiply(ref invOrientation, ref bodyInvInertia, out worldInvInertia);
  Matrix.Multiply(ref worldInvInertia, ref transform.Orientation, out worldInvInertia);
  Matrix.Multiply(ref invOrientation, ref bodyInertia, out worldInertia);
  Matrix.Multiply(ref worldInertia, ref transform.Orientation, out worldInertia);
}
```

Cohesión: Las entidades se mantienen juntas.

Esquive: Las entidades evaden obstáculos.

Persecución de Metas: Permiten cazar un alimento.

Tiempo de Vida: Se especifica el tiempo de vida en el que un personaje puede morir.

El siguiente método implementa las tres reglas anteriores, al actualizar el estado de un habitante cuando ocurre un quebrantamiento de la tierra en el juego.

Para implementar la regla de cohesión se utiliza el método *GoToPosition*, dentro del cual se alerta a la población a mantenerse alejada del desastre, tomando la posición del habitante que más lejano se encuentre de la base del desastre, como la posición base para que los demás habitantes se dirijan y puedan salvar su vida.

La regla persecución de metas se aplica en el método *FindFood,* el cual permite cazar alimento.

Si el habitante no está vivo se realizan los cálculos correspondientes para disminuir la población.

```
public void Update(GameTime gameTime, bool isEarthQuakeOn, disasterEvent disaster,
inhabitantVelocity iVelocity, Vector3 disasterLocation, ref List<Pig> pigList, float waterHeight, ref
TRex objTrex, EvolutionRandom rand)
{
  int newInhabitants = 0;
  foreach (Inhabitant hb in inhabitantList)
  {
    if (hb.isAlive)
      {
        if (isEarthQuakeOn)
          {
            switch (disaster)
            {
              case disasterEvent.volcano:
              hb.GoToPosition(disasterLocation, iVelocity, rand,
gameTime.ElapsedGameTime.Milliseconds);
              break;
              case disasterEvent.flood:
              hb.GoToPosition(disasterLocation, iVelocity, rand,
gameTime.ElapsedGameTime.Milliseconds);
              break;
              case disasterEvent.snow:
              hb.GoToHome(houseLocation, iVelocity, 5,
gameTime.ElapsedGameTime.Milliseconds, rand);
              break;
              case disasterEvent.earthquake:
              hb.GoToPosition(disasterLocation, iVelocity, rand,
gameTime.ElapsedGameTime.Milliseconds);
              break;
              default:
              break;
            }
            KillInhabitants(disaster, iVelocity);
          }
          else
```

```
          {
            isOnKillingTime = true;
            hb.IsOnDisaster = false;
            hb.Roam(gameTime.ElapsedGameTime.Milliseconds);
            if (isHuntingEnable)
            hb.FindFood(gameTime.ElapsedGameTime.Milliseconds, ref pigList, houseLocation);
            if (isSnusnusEnable)
              newInhabitants += MultiplyInhabitants(hb, gameTime, rand);
          }
          hb.Update(gameTime, waterHeight, ref objTrex);
        }
        if (!hb.isAlive && hb.isActive)
          {
            if (hb.isMale)
              maleCount -= 1;
            else
              femaleCount -= 1;
            hb.isActive = false;
            this.game.Components.Remove(hb);
          }
      }
        if (newInhabitants > 0)
          {
            newInhabitant = true;
          }
        else
          {
            newInhabitant = false;
          }
          for (int i = 0; i < newInhabitants; i++)
          {
            AddInhabitant(game, null, rand, houseLocation, 3);//add random sex inhabitant
          }
}
```

4.3.4. Comportamientos Esperados

El jugador de Evolution tiene como objetivo lograr pasar a la siguiente fase del juego con la mayor parte de la población en la isla.

Sea cual sea la configuración inicial, la evolución conducirá a la población a uno de los tres estados siguientes:

- **Extinción**: Al cabo de un número finito de generaciones desaparecen todos los miembros de la población.
- **Estabilización**: Al cabo de un número finito de generaciones la población queda estabilizada, bien de forma rígida e inamovible, bien de forma oscilante entre dos o más formas.
- **Variación Constante**: En esta situación la población crece indefinidamente.

En resumen, las reglas deben ser tales que la conducta de la población sea a la vez interesante e impredecible.

En cuanto a cada personaje, deben cumplir con ciertas reglas establecidas para el juego, estas son:

1. **Supervivencia**: Cada jugador que tenga 2 ó 3 habitantes en la isla sobrevive y pasa a la generación siguiente.
2. **Fallecimiento**: Cada individuo que tenga menos de 1 habitante fallece por falta de población en la isla.
3. **Nacimiento**: Cada vez que el habitante ingrese en la cueva con su pareja, la población aumentará en uno.

Mientras se avanza en el juego se descubre que la población va constantemente experimentando cambios insólitos e inesperados.

En cuanto al animal enemigo de los habitantes de Evolution, en este caso el dinosaurio, también sigue reglas preestablecidas para cumplir con su misión. Estas son:

1. **Supervivencia:** El dinosaurio tiene que sobrevivir igual que los animales a los peligros que la naturaleza le presente, como son terremotos, inundaciones y erupciones volcánicas.
2. **Ataque:** El dinosaurio tiene la ventaja de ser un predador y cazar su alimento sin que éste pueda ser atacado por otra criatura dentro de Evolution.

Personajes

En la medida en que el juego logre cautivar la atención del jugador más efectivos serán sus resultados, así pues se espera que Evolution cumpla las expectativas en cuando al dibujo, modelado, animación, música, edición de video, guión, scripter, programación, debugging y testing.

La idea de esto es principalmente cautivar, deslumbrar e impactar al jugador, el hecho de encontrarse con sorpresas a nivel gráfico o de comportamiento, acompañados de un excelente fondo musical.

En Evolution se trata de que un habitante se comporte como una entidad que perciba su entorno y actúe en consecuencia decidiendo que hacer por medio de reglas de lógica propias del agente como se muestra en la figura 218.

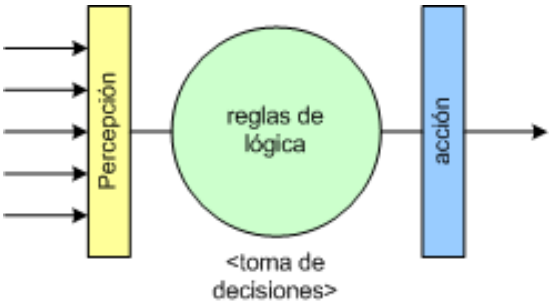


Figura 218. Comportamiento de Habitantes.

Cada entidad en Evolution se comportará como un agente inteligente. De este modo, cada entidad percibirá su entorno mediante cierta información disponible para él; las entidades simulan seres vivos con organismos de percepción no ideales, con ojos que poseen un cierto campo de visión, con oídos, etc., para que el comportamiento final sea lo más fiel posible al del ser vivo que se intente copiar. En la siguiente figura (figura 219.) se muestra el comportamiento de un habitante cuando se ha provocado una acción sobre este. En este caso el habitante muestra sus datos al jugador.

Figura 219. Habitantes de Evolution.

La aplicación de cada una de las reglas indica al caracter hacia donde moverse (figura 220). Luego la toma de decisión que tendrá que realizar la entidad será la promediación ponderada de cada una de las acciones sugeridas por las reglas.

La dirección a tomar puede ser alterada por una regla de alta prioridad que indique la persecución de una meta, como por ejemplo el avistamiento de una presa. Cuando se da esta situación, todas las entidades modifican su dirección para cazar la presa, después de esto, la regla se desactiva y deja de tener influencia sobre la decisión tomada.

Figura 220. Población de Evolution.

4.3.5. Integración de Niveles

Evolution cuenta con un menú de tres niveles, como se muestra en la figura 221, de los cuales solo en primer nivel cubre el alcance de este proyecto.

Figura 221. Niveles de Evolution.

La carga de cada nivel se realiza con la herramienta XNA, la cual permite cargar las pantallas del juego.

El siguiente código muestra la carga de pantallas del juego, dentro de las cuales incluye la pantalla de niveles de Evolution.

```
public static void Load(ScreenManager screenManager, bool loadingIsSlow, params
GameScreen[] screensToLoad)
{
  // Tell all the current screens to transition off.
  foreach (GameScreen screen in screenManager.GetScreens())
        screen.ExitScreen();
        // Create and activate the loading screen.
        SplashScreen splashScreen = new SplashScreen(screenManager,
loadingIsSlow, screensToLoad);
        screenManager.AddScreen(splashScreen);
}
```

La pantalla siguiente muestra el menú principal de los niveles del juego en la figura 222.

Figura 222. Niveles de Evolution

Una vez cargada la pantalla el jugador selecciona uno de los niveles para jugar, esto se lo realiza mediante el siguiente código:

```
public override void HandleInput(InputState input)
    {
        #region Implementation
        // Move to the previous menu entry?
        if (input.MenuUp)
        {
            selectedEntry--;
            if (selectedEntry < 0)
                selectedEntry = menuEntries.Count - 1;
        }

        // Move to the next menu entry?
        if (input.MenuDown)
        {
            selectedEntry++;
            if (selectedEntry >= menuEntries.Count)
                selectedEntry = 0;
        }
        // Accept or cancel the menu?
        if (input.MenuSelect)
        {
            OnSelectEntry(selectedEntry);
```

```
        }
        else if (input.MenuCancel)
        {
            OnCancel();
        }
        #endregion
    }

    /// <summary>
    /// Handler for when the user has chosen a menu entry.
    /// </summary>
    protected virtual void OnSelectEntry(int entryIndex)
    {
        #region Implementation
        menuEntries[selectedEntry].OnSelectEntry();
        #endregion
    }
```

Cuando ya se ha seleccionado el nivel de juego, se carga el contenido de éste mediante el siguiente método, donde cada *sprit* del nivel es cargado mediante el método Load que permite cargar cada frame por segundo.

```
public override void LoadContent()
{
  #region Implementation
  if (content == null)
    content = new ContentManager(ScreenManager.Game.Services, "Content");
    gameFont = content.Load<SpriteFont>("Fonts\\gamefont");
    for (int frameIndex = 0; frameIndex < NUMBER_OF_FRAMES + 1; frameIndex++)
    {
          animationFrameCollection.Add(content.Load<Texture2D>("Media\\Intro\\" +
frameIndex.ToString()));
    }
    spriteAnimator.Load(animationFrameCollection, NUMBER_OF_FRAMES,
FRAMES_PER_SECOND);
    ScreenManager.Game.ResetElapsedTime();
  #endregion
}
```

Cada frame es dibujado mediante el siguiente código:

```
/// <summary>
    /// Draws a frame on the screen
```

```
        /// </summary>
        /// <param name="batch">SpriteBatch object</param>
        /// <param name="frame">index of the actual frame</param>
        /// <param name="screenPos">Screen Position</param>
        public void DrawFrame(SpriteBatch batch, int frame, Vector2 screenPos)
        {
            #region Implementation

            Rectangle sourcerect = new Rectangle(0, 0,
                  animationFrameCollection[frame].Width,animationFrameCollection[Frame].Height);
            batch.Draw(animationFrameCollection[frame], screenPos, sourcerect, Color.White,
               rotation, new Vector2(animationFrameCollection[frame].Width / 2,
animationFrameCollection[frame].Height / 2), scale, SpriteEffects.None, depth);
            #endregion
        }
```

Finalmente se obtiene el nivel que se ha seleccionado como se muestra en la figura 223.

Figura 223. Primer Nivel de Evolution

4.3.6. Estabilización y pruebas

Al final de la implementación de los modelos matemáticos y fórmulas físicas y geométricas se obtiene un mejor funcionamiento del juego con lo cual se logra un producto para ser sometido a pruebas.

Evolution es sometido a una etapa de pruebas, detección y corrección de errores hasta alcanzar un juego estable, eficiente, creíble y divertido.

Para probar la inteligencia artificial en los personajes del juego se los debe someter a pruebas que verifiquen que los personajes se comporten como agentes inteligentes dentro del juego, capaces de responder ante un evento que suceda en su entorno, para lo cual se procede a modificar las reglas dadas a cada personaje.

Cada regla aplicada debe tomar en cuenta aspectos como:

- Conocimiento de entidades vecinas: Como su radio de proximidad y ángulo.

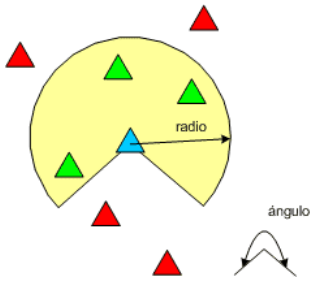


Figura 224. Primer Nivel de Evolution

- Analizar el campo de visión de una entidad, analizando la posición actual de la misma.
- Verificar la distancia contra toda entidad vecina. Si alguna distancia es menor a la mínima estipulada efectuar un movimiento que revierta esta situación.
- Promediar la aceleración de todas las entidades vecinas.

Es importante también, realizar pruebas que corroboren los datos que se obtiene en fórmulas para tener una veracidad en los datos al momento de su utilización en Evolution.

Por lo general, el comportamiento adecuado se obtiene mediante la generación de la prueba y el error reportado, para lo cual se puede agregar y disminuir reglas.

4.3.7. Toma de Resultados

Los resultados obtenidos en este sprint se reflejan en el product backlog donde se definieron cada una de las tareas y los estados que cada una de ellas presenta al final del periodo de desarrollo.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
3	10-nov-08	20	3

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis	Pendiente	Christian	6-dic
Prototipado	En curso	Natalia	24-dic
Prototipado	Terminada	Christian	1-ene
Codificación	Terminada	Natalia	
Codificación	Terminada	Christian	
Codificación	Terminada	Natalia	
Codificación	Terminada	Natalia	
Pruebas	Terminada	Christian	
Reunión	Terminada	Natalia	

Tabla 14. Product Backlog.

4.4. Sprint 4

4.4.1. Creación del Product Backlog

El Product Backlog del sprint 4 contiene las tareas correspondientes a este sprint desarrollado. A continuación se muestra la tabla 15, la cual indica el tipo de tarea a desarrollarse, el nombre de la tarea, el estado de esta tarea, la persona responsable de cumplirla y los días festivos si existen durante este tiempo.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
4	11-feb-09	17	2

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis	Pendiente	Christian	
Pruebas	Terminada	Natalia	
Pruebas	Terminada	Christian	
Reunión	Terminada	Natalia	

Tabla 15. Tabla de tareas para el Sprint 4.

El siguiente cuadro (tabla 16) muestra las tareas por fecha de terminación y el porcentaje avanzado por cada día de tarea.

SPRINT	INICIO	DURACIÓN
4	11-feb-09	17

	X	J	V	S	D	L	M	X	J	V	S	D	L	M	X	J	V	S	D	L
	11-feb	12-feb	13-feb	14-feb	15-feb	16-feb	17-feb	18-feb	19-feb	20-feb	21-feb	22-feb	23-feb	24-feb	25-feb	26-feb	27-feb	28-feb	1-mar	2-mar
Tareas pendientes	4	4	4	4		3	3	3	3	3	3		2	2	2	2	1	1		1
Horas de trabajo pendientes	53	47	44	38		30	36	36	37	35	29		26	24	21	19	15	11		8

PILA DEL SPRINT						ESFUERZO														
Backlog ID	**Tarea**	**Tipo**	**Estado**	**Responsable**																
1	Creación del Product Backlog	Análisis	En curso	Natalia	19	18	17	16		14	14	13	12	11	10		9	8	7	6	5	4		2
2	Integración con la consola Xbox 360	Pruebas	Terminada	Christian	8	3	4	4		4	10	12	14	14	10		9	8	7	6	5	4		4
3	Estabilización y Pruebas	Pruebas	Terminada	Natalia	19	19	16	12		8	8	7	7	6	6		5	5	5	5	3	1		1
4	Toma de Resultados	Reunión	Terminada	Christian	7	7	7	6		4	4	4	4	4	3		3	3	2	2	2	2		1

Tabla 16. Esfuerzo en el Sprint 4.

El gráfico 225 muestra el esfuerzo empleado por cada una de las tareas establecidas para el sprint 4.

Proyecto
Evolution

SPRINT	INICIO	DÍAS
4	11-feb-09	17

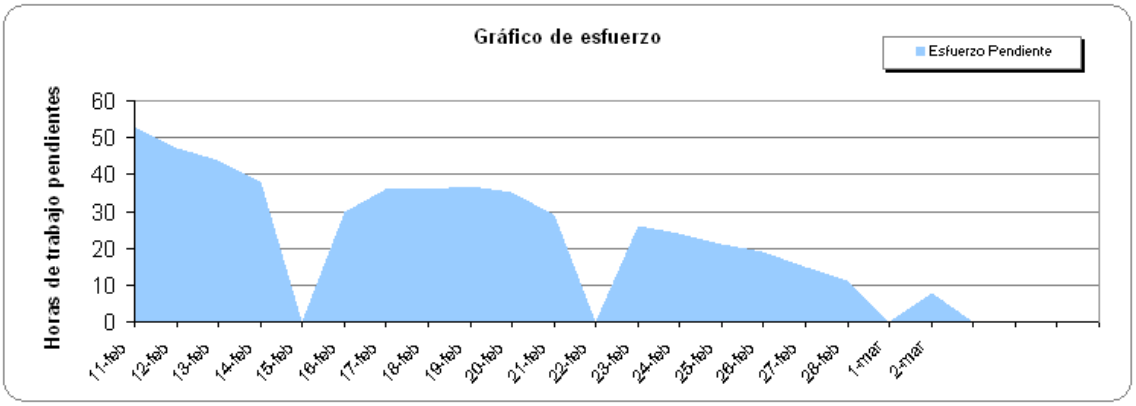


Figura 225. Gráfico de Esfuerzo

El gráfico 226 define las tareas pendientes y el avance de cada una de las tareas según las fechas definidas.

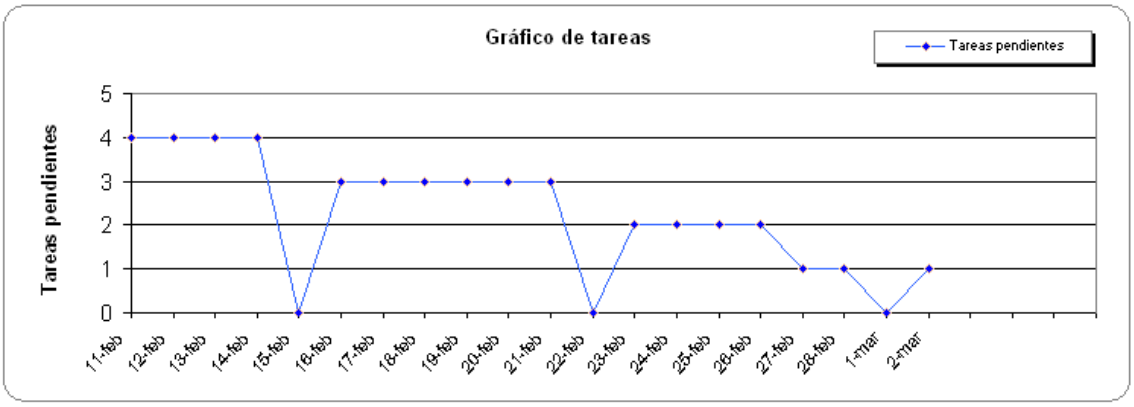


Figura 226. Gráfico de Tareas

El gráfico 227 muestra el estado de las tareas según se va avanzando en el cronograma definido. Para realizar este gráfico es necesario realizar un cuadro en el que se indique el estado de cada tarea. Cada número uno señala el estado de la tarea en ese día.

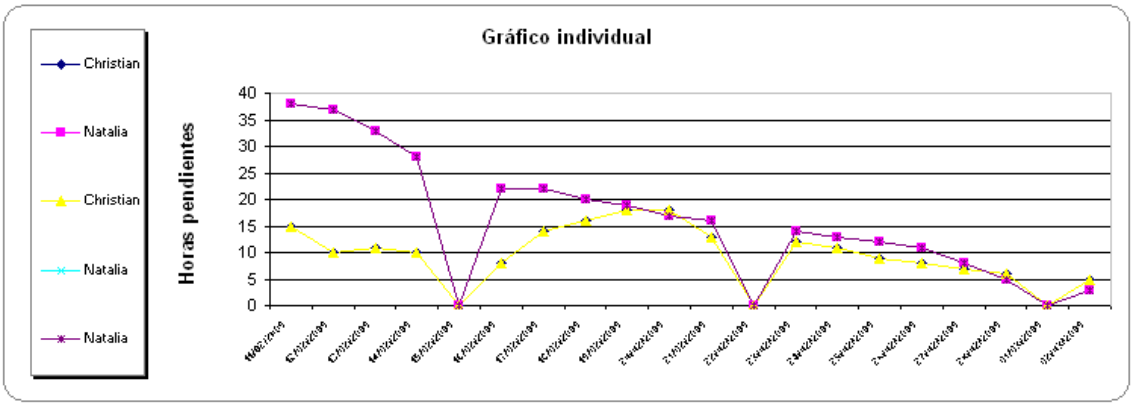


Figura 227. Gráfico Individual de Esfuerzo

	11-feb	12-feb	13-feb	14-feb	15-feb	16-feb	17-feb	18-feb	19-feb	20-feb	21-feb	22-feb	23-feb	24-feb	25-feb	26-feb	27-feb	28-feb	1-mar	2-mar
Christian	15	10	11	10		8	14	16	18	18	13		12	11	9	8	7	6		5
Natalia	38	37	33	28		22	22	20	19	17	16		14	13	12	11	8	5		3
Christian	15	10	11	10		8	14	16	18	18	13		12	11	9	8	7	6		5
Natalia	38	37	33	28		22	22	20	19	17	16		14	13	12	11	8	5		3
Natalia	38	37	33	28		22	22	20	19	17	16		14	13	12	11	8	5		3

4.4.2. Integración Xbox360

XNA Game Studio permite crear juegos basados en Windows y para ser instalados en la consola de Xbox 360.

Para conectar la consola XBox 360 al computador, es necesario instalar el Framework XNA en la consola por medio de Xbox Live Service, previamente se debe suscribirse a al XNA Creators Club, la cual permite crear y depurar los juegos en la consola Xbox 360.

Dentro de XNA existe la opción para el enlace con XNA Game Studio Device Center como se muestra en la figura 228.

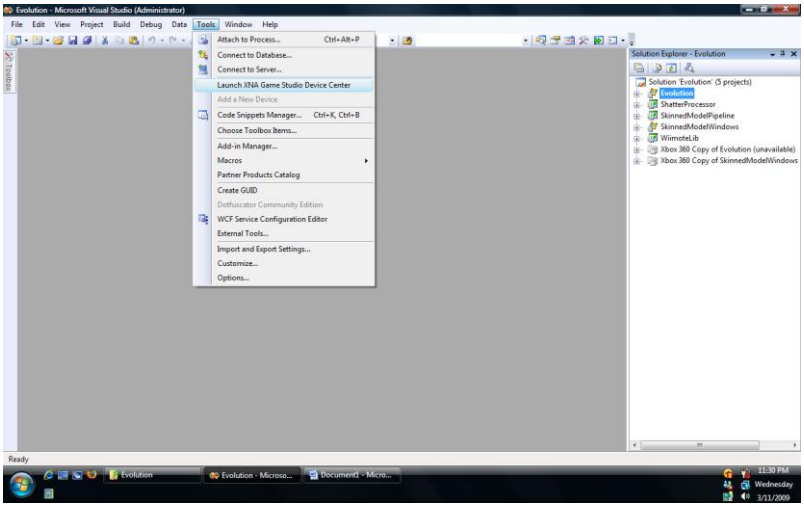


Figura 228. XNA Game Studio Device Center

La siguiente pantalla (figura 229) muestra la ventana principal que permite agregar el dispositivo al cual se va a conectar XNA.

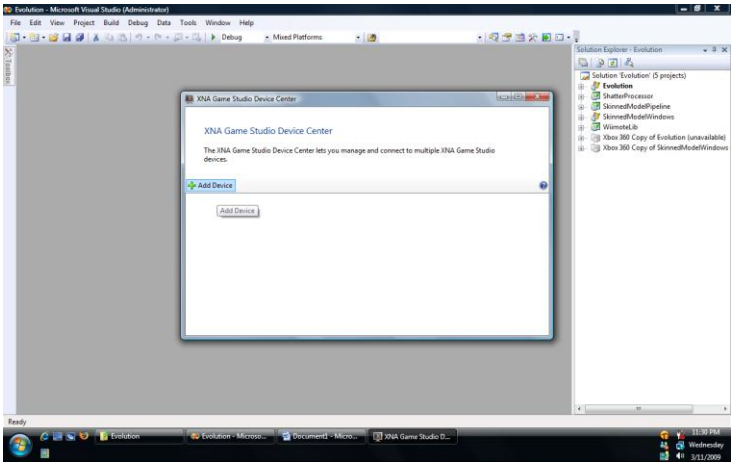


Figura 229. Conexión con Game Studio Device Center.

Se ingresa un nombre para la consola que se va a usar, como se muestra en la figura 230.

Figura 230. Nombre de la consola Xbox 360.

Una vez que se haya realizado la suscripción al XNA Creators Club, se procederá a configurar la consola Xbox 360 para poder transferir los juegos desde el equipo, para lo cual es necesario seguir los siguientes pasos:

- En la consola Xbox 360, se debe iniciar sesión en Xbox LIVE con el gamertag asociado a la suscripción especial.
- Cuando se haya iniciado sesión, hay que desplazarse a *Bazar de juegos.*
- En el área *Bazar de juegos*, hay que descargar *XNA Game Studio Connect.* Esta descarga conecta el equipo de Windows que ejecuta XNA Game Studio con la consola Xbox 360 para implementar y depurar los juegos en la consola Xbox 360.
- Una vez concluida la descarga, hay que desplazarse hasta el canal *My Xbox.*
- Seleccionar *Biblioteca de juegos, Community Games,* y, a continuación, *XNA Game Studio Connect*, como se muestra en la figura 231.

Figura 231. Ventana de Conexión.

La primera vez que se ejecute XNA Game Studio Connect, aparecerá un código en la pantalla. Este código será necesario para conectarse al equipo.
Se ingresa la clave de subscripción para el XNA Game Studio Connect, como se muestra en la figura 232.

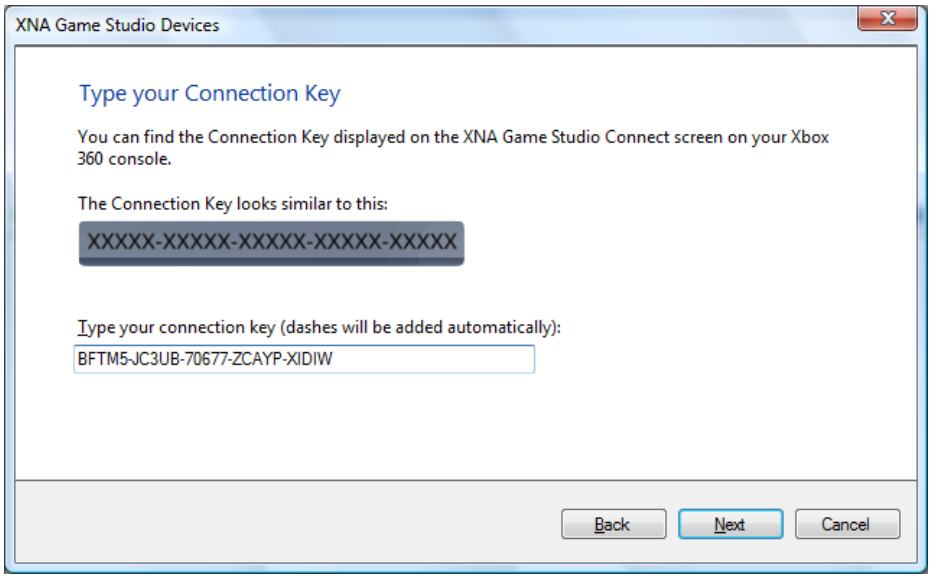


Figura 232. Clave de conexión al XNA Creators club.

Se prueba la conexión entre la computadora y la consola de Xbox 360, como se muestra en la figura 233. Tanto el equipo como la consola Xbox 360 deben mostrar un mensaje que indique que dispone de una conexión correcta.

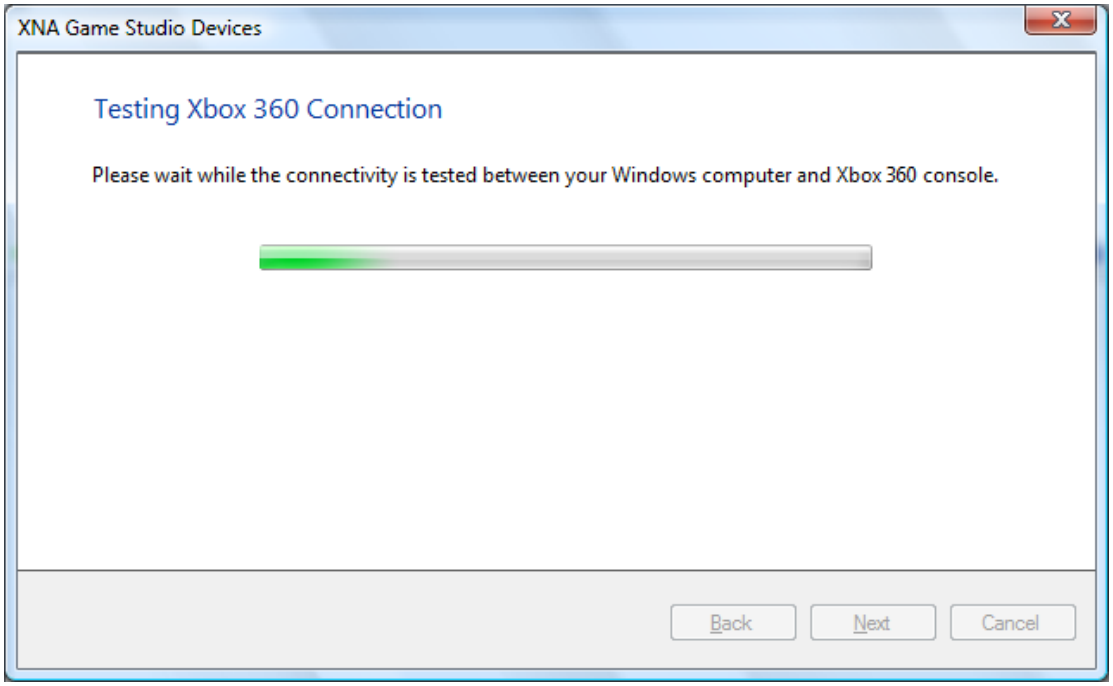


Figura 233. Prueba de Conexión entre la computadora y la consola Xbox 360.

A continuación se visualiza una conexión exitosa entre el computador y la consola Xbox 360 como se muestra en la figura 234.

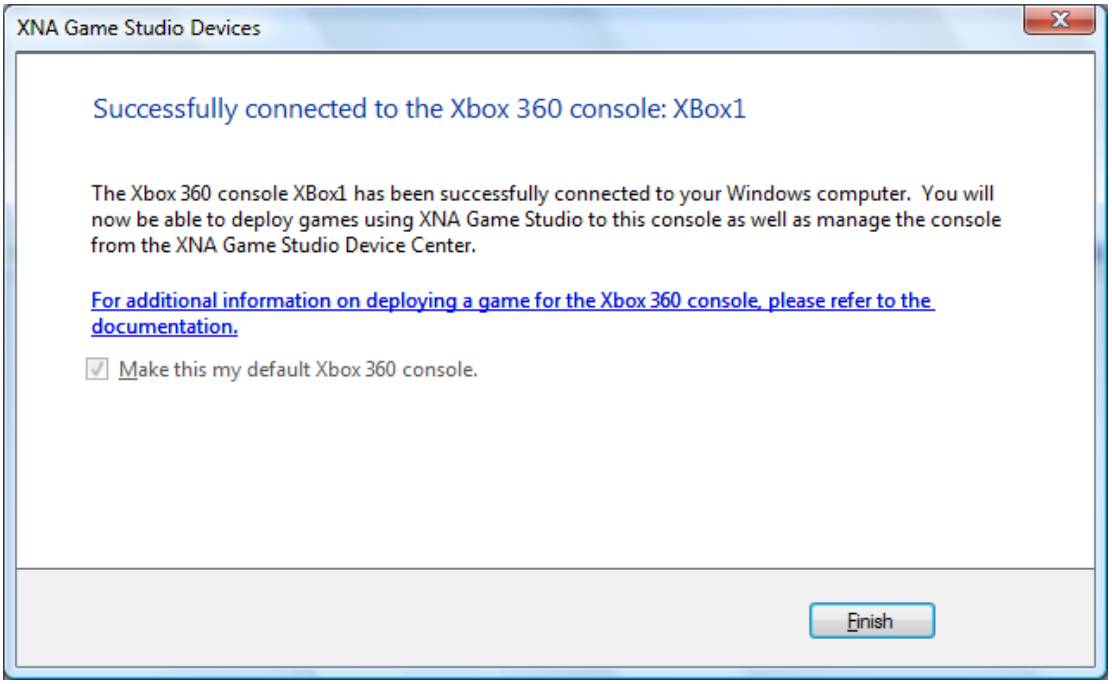


Figura 234. Conexión exitosa al Xbox 360.

Una vez probada la conexión aparece una ventana de espera de conexión entre el computador y la consola, como se muestra la figura 235.

Figura 235. Conexión exitosa al Xbox 360.

Una vez realizada la conexión, se muestra el dispositivo agregado en la figura 236.

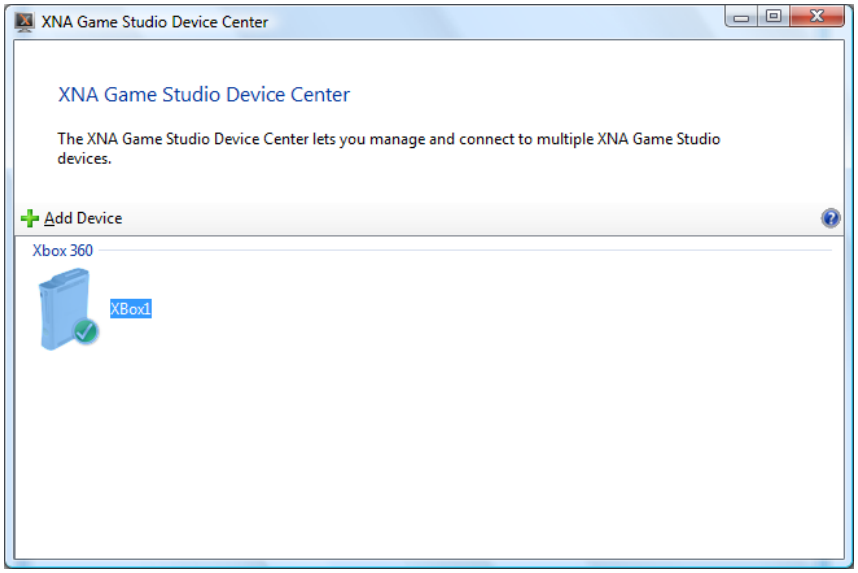


Figura 236. Pantalla de Dispositivos conectados.

Finalmente se podrá implementar el proyecto Evolution para la Xbox 360 desde el XNA Game Studio de la consola Xbox 360.

Para implementar el juego en la consola Xbox 360, se debe abrir el proyecto utilizando XNA Game Studio, para lo cual se debe asegurar que XNA Game Studio Connect está ejecutándose en la consola Xbox 360.

A continuación se crea una copia del proyecto para la consola Xbox 360 como se muestra en la figura 237.

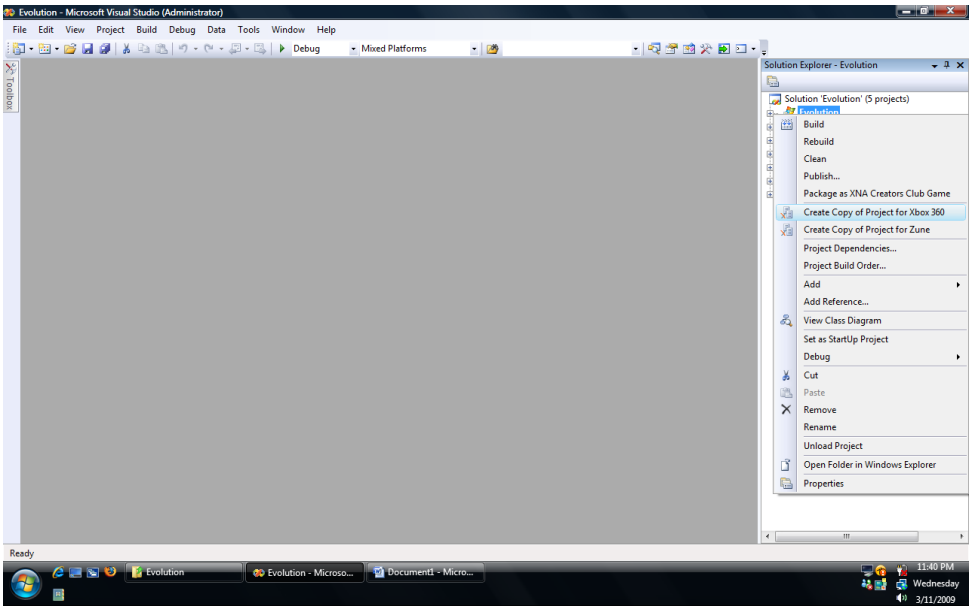


Figura 237. Pantalla de Dispositivos conectados.

Posteriormente se establece el proyecto de Xbox 360 como el proyecto de inicio, como se muestra en la figura 238.

La copia del proyecto es creada en la misma solución del proyecto en XNA.

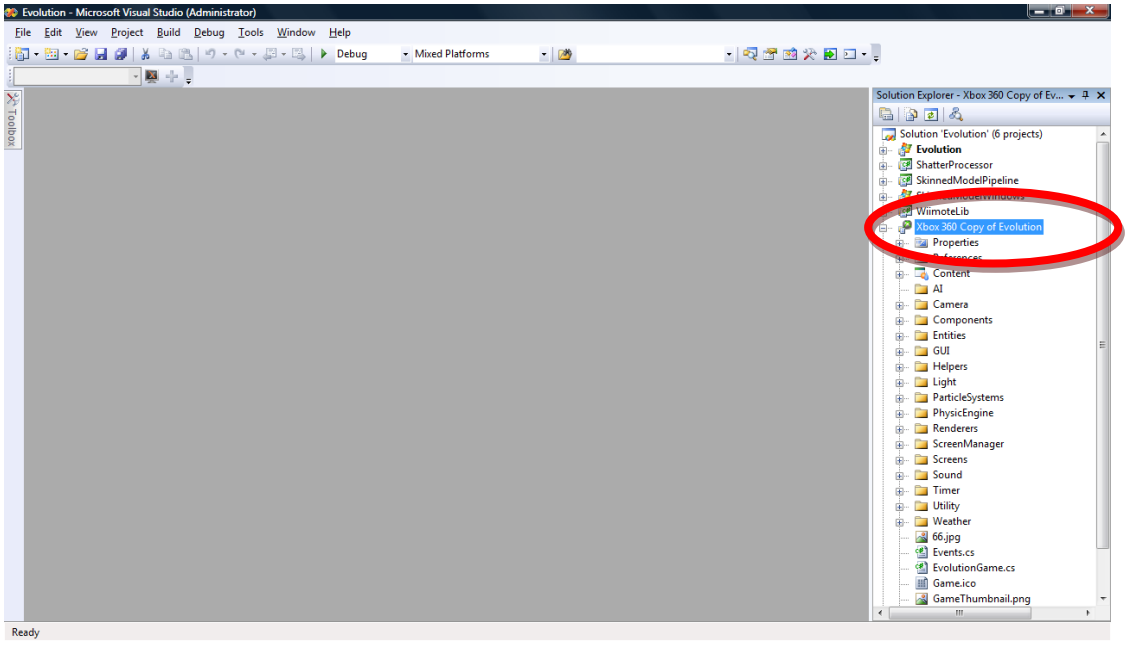


Figura 238. Copia del proyecto Evolution.

A continuación, se construye el proyecto y éste se ejecutará de forma automática en la consola Xbox 360 como se muestra en la figura 239. y 240.

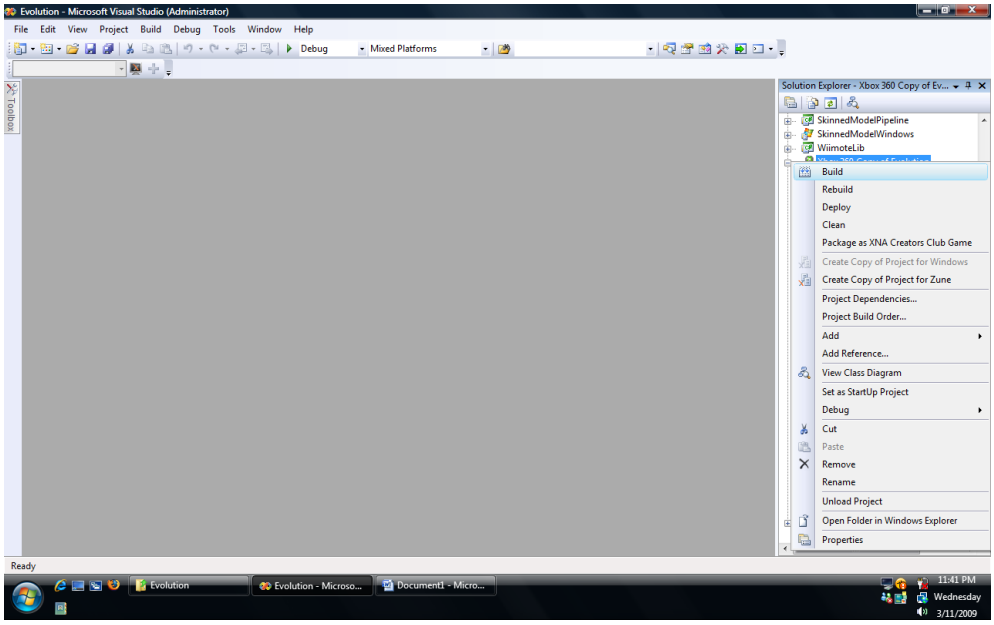


Figura 239. Construcción del proyecto Evolution.

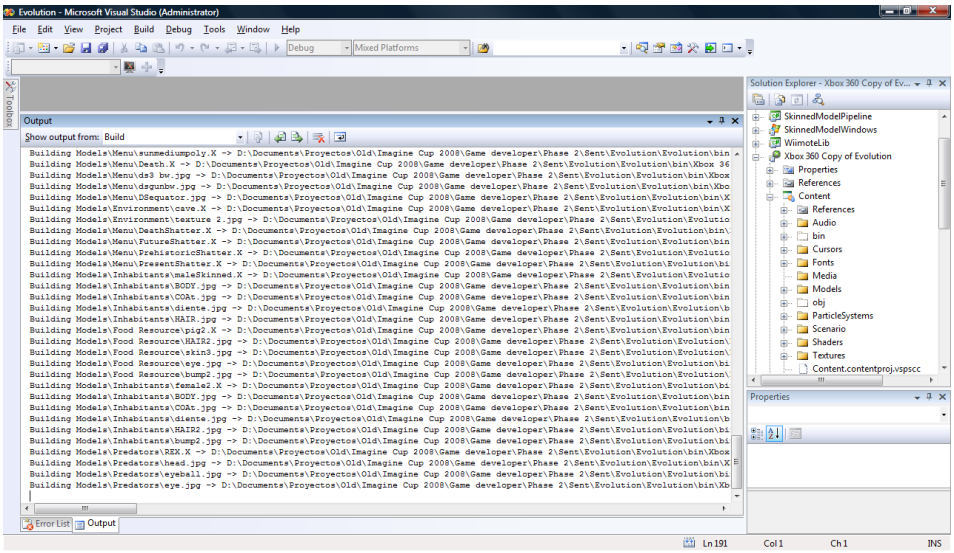


Figura 240. Ejecutar un proyecto XNA en la consola XBox 360.

4.4.3. Estabilización y pruebas

Para cumplir con este sprint fue necesario pasar el juego a la consola Xbox 360. Según los pasos dados anteriormente, se obtuvo conexión entre la consola y el computador y se comprobó que el juego corre perfectamente en la consola del Xbox 360.

4.4.4. Toma de Resultados

La toma de resultados, en el sprint 4 analiza el cumplimiento de los objetivos planteados de acuerdo al tiempo establecido como se muestra en la tabla 17.

Proyecto
Evolution

Nº de sprint	Inicio	Días	Jornada
1	7-oct-08	22	4

TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Codificación	Terminada	Christian	
Pruebas	Terminada	Natalia	
Pruebas	Terminada	Christian	
Reunion	Terminada	Natalia	

Tabla 17. Resultados Obtenidos

CAPITULO V

5.1. Informe de Resultados. (Estadísticas de resultados finales de los sprints)

Sprint 1

El tiempo para el desarrollo del sprint 1 fue de 20 días. En este sprint se realizaron las siguientes tareas:

SPRINT	INICIO	DURACIÓN
1	10-nov-08	20

PILA DEL SPRINT				
Backlog	Tarea	Tipo	Estado	Responsal
	Creación del Product Backlog	Análisis	En curso	Natalia
	Concepto de Jugabilidad	Prototipado	Terminada	Natalia
	Implementación de Motor Grafico	Pruebas	Terminada	Christian
	Carga de escenarios y Personajes	Análisis	Terminada	Christian
	Ambientación	Codificación	Terminada	Natalia
	Skybox	Codificación	Terminada	Natalia
	Terreno	Codificación	Terminada	Christian
	Agua	Codificación	Terminada	Christian
	Animación	Codificación	Terminada	Christian
	Rigging	Codificación	En curso	Christian
	Skinning	Codificación	En curso	Christian
	Exportación	Codificación	Pendiente	Christian
	Estabilización y Pruebas	Codificación	Pendiente	Natalia
	Toma de Resultados	Codificación	Pendiente	Natalia

Figura 241. Sprint 1

Sprint 2

El tiempo para el desarrollo del sprint 2 fue de 19 días. En este sprint se realizaron las siguientes tareas:

SPRINT	INICIO	DURACIÓN
2	3-dic-08	19

PILA DEL SPRINT				
Backlog	Tarea	Tipo	Estado	Responsal
	Menú Opciones	Análisis	Terminada	Christian
	Menú Créditos	Codificación	Terminada	Natalia
	Menu Ayuda	Codificación	Terminada	Christian
	Load Screen	Codificación	Terminada	Natalia
	In Game GUI	Codificación	Terminada	Christian
	Integración de Música y Fx	Codificación	Terminada	Christian
	Manejo de Partículas, luces y Cam	Codificación	En curso	Christian
	Integración de Shaders	Codificación	En curso	Christian
	Estabilización y Pruebas	Codificación	Pendiente	Natalia
	Toma de Resultados	Codificación	Pendiente	Natalia

Figura 242. Sprint 2

Sprint 3

El tiempo para el desarrollo del sprint 3 fue de 23 días. En este sprint se realizaron las siguientes tareas:

SPRINT	INICIO	DURACIÓN
1	9-sep-08	23

PILA DEL SPRINT

Backlog	Tarea	Tipo	Estado	Responsable
	Creación del Product Backlog	Análisis	En curso	Christian
	Implantación de Modelos Matemátic	Prototipado	Terminada	Natalia
	Inteligencia Artificial de Personajes	Prototipado	Terminada	Christian
	Comportamientos Esperados	Codificación	Terminada	Natalia
	Integración de Niveles	Codificación	Terminada	Christian
	Manejo de Menús	Codificación	Terminada	Natalia
	Manejabilidad	Codificación	Terminada	Natalia
	Estabilización y Pruebas	Pruebas	Terminada	Christian
	Toma de Resultados	Reunión	Terminada	Natalia

Figura 243. Sprint 3

Sprint 4

El tiempo para el desarrollo del sprint 4 fue de 17 días. En este sprint se realizaron las siguientes tareas:

SPRINT	INICIO	DURACIÓN
4	11-feb-09	17

PILA DEL SPRINT

Backlog	Tarea	Tipo	Estado	Responsable
	Product Backlog	Codificación	Terminada	Natalia
	Integración Xbox 360	Codificación	Terminada	Christian
	Estabilización y Pruebas	Codificación	Terminada	Christian
	Toma de Resultados	Codificación	Terminada	Christian

Figura 244. Sprint 4

5.2. Conclusiones

El desarrollo de videojuegos es un proceso complejo que requiere de un amplio conocimiento en diversas áreas así como de tiempo y esfuerzo pero esto no quiere decir que sea imposible de realizar.

Al concluir este proyecto, se ha logrado conocer y aplicar más a fondo el proceso para desarrollar un videojuego, para lo cual se ha conseguido implementar un motor gráfico comercial para el funcionamiento de Evolution, el cual constituye la base para posteriormente crear un proyecto más robusto y completo en cuanto a sus niveles de dificultad.

El desarrollo de Evolution, requirió de un análisis profundo de los aspectos fundamentales sobre la teoría de juegos, algoritmos, fórmulas matemáticas a ser implementadas así como del Framework XNA y los patrones de comportamiento relacionados con la inteligencia artificial a ser aplicada para la construcción del videojuego.

Se implementó la teoría de juegos como un referente para definir estrategias y objetivos del juego que conjuntamente con fórmulas matemáticas y físicas permitieron formar el núcleo principal del juego, dándole una mayor productividad a la aplicación.

Además se aplicaron varios conceptos que permitieron comprender de mejor manera la programación gráfica aplicada en videojuegos 3D. Dentro del proyecto, fueron implementados conceptos como skybox, ambientación, biped, rigging, transformaciones como rotación, escalado, traslación, entre otras que ayudaron a dinamizar el comportamiento del juego de tal manera que luzca interesante y realista, facilitando de esta manera su aprendizaje.

De igual manera la inteligencia artificial ha jugado un papel muy importante dentro del desarrollo del videojuego ya que le otorgó un mayor realismo y a su

vez permitió evitar un comportamiento monótono en cuanto a la interacción de los personajes.

Como parte del desarrollo de Evolution se contemplaron todos los pasos previos relacionados con el diseño como lo son: la creación de bocetos gráficos y guiones para posteriormente realizar el modelado, diseño de personajes y escenarios que comprenden el juego, los cuales ayudaron a complementar el enfoque o visión del juego.

Para el proyecto, se aplicó la metodología Scrum, la cual permitió obtener resultados visibles en cada iteración con lo cual se controló de mejor manera la gestión de calidad para evitar futuros fallos en el producto final.

Además, la historia creada para el videojuego conduce al jugador al aprendizaje de formas de cuidar y preservar el medio en el que se desenvuelve, generando una historia animada y a la vez educativa con los personajes y escenarios necesarios para representarla.

Finalmente se logró culminar con éxito el desarrollo completo de la primera fase del juego así como de los elementos artísticos involucrados en su creación sobrepasando las expectativas iniciales del proyecto.

5.3. Recomendaciones

Para crear un videojuego se recomienda tener sólidos conocimientos en áreas como matemática, física e inteligencia artificial, puesto que estas ciencias constituyen la base fundamental para crear el motor gráfico que realice los procesos dinámicos del juego, o en su defecto para comprender la lógica de un motor de videojuego existente y poder modificarla según la necesidad.

En cuanto al diseño se recomienda tener claro el concepto del juego a la hora de empezar el proyecto, analizando detenidamente y cuidadosamente su enfoque, género y jugabilidad, ya que esto puede ocasionar problemas que pueden ocasionar retrasos.

Para el desarrollo del videojuego se recomienda estar familiarizado con la herramienta de desarrollo ya que de esta manera la curva de aprendizaje durante el desarrollo del proyecto será mínima y así se dará mayor prioridad al desarrollo como tal. De la misma manera, la interfaz de comandos debe ser intuitiva y amigable, lo cual constituye una gran ventaja para el jugador.

Antes de desarrollar un videojuego se recomienda realizar una análisis de mercado para tomar en cuenta las diferentes tendencias y con base en esto priorizar en el desarrollo de juegos novedosos y dirigidos a distintos usuarios, creando productos especializados para mercados poco explotados pero con una gran cantidad de consumidores.

Para la comercialización del videojuego se recomienda tomar en cuenta el costo de realización y la competitividad que presente el juego para salir al mercado, además es importante escoger una consola en la que se pueda instalar en juego sin problemas y además ésta debe estar entre las más seleccionadas por los jugadores con lo cual se aumenta significativamente la ganancia para el desarrollador del videojuego ya que el producto será adquirido por más personas.

Se recomienda utilizar una metodología ágil para este tipo de proyectos ya que éstos son considerados de alto riesgo por la inestabilidad al momento de su construcción y por la incompatibilidad que puede surgir entre herramientas utilizadas y tecnologías aplicadas. Además este tipo de metodologías permiten obtener versiones del proyecto en un corto plazo con lo cual se puede visualizar los resultados de manera más rápida y sin requerir demasiada documentación como lo exigen otras metodologías aplicadas en el desarrollo de un proyecto de software común.

Como herramienta de apoyo para la realización de este tipo de tesis se recomienda aumentar textos en la biblioteca de la universidad para orientar de mejor manera el desarrollo y ayudar al estudiante a que culmine en un menor tiempo su tesis.

5.4. Blibliografía

- Tomado de: http://www.fuzzygamedev.com/2007/02/10/tutorial-linea-de-vision-line-of-sight/#more-99
- Tomado de:http://www.fuzzygamedev.com/2007/05/01/flocking/#more-234
- Tomado de: http://www.fuzzygamedev.com/category/tutoriales/inteligencia-artificial/
- Tomado de: http://aigamedev.com/
- Tomado de: http://www.fuzzygamedev.com/category/tutoriales/xna
- Tomado de: http://creators.xna.com/es-ES/create_detail
- Tomado de: http://dragonworld.mforos.com/1411961/6659864-como-crear-un-videojuego/
- Tomado de: http://www.monografias.com/trabajos12/moma/moma.shtml
- Tomado de: http://www.material_simulacion.ucv.cl/tipos_de_modelos_matematicos.htm
- Tomado de: http://www.gaugeus.com/ramblings/2007/9/9/modelos-matematicos
- Tomado de: http://www.zonaeconomica.com/teoriadejuegos/tiposdejuego
- Tomada de: http://es.wikipedia.org/wiki/Teor%C3%ADa_de_juegos
- Tomado de: http://www.cmirg.com/SitioCMIRG/pantallas/LibroDirectX.aspx
- Tomado de: http://www.desarrolloweb.com/articulos/1806.php
- Tomado de: http://www.alegsa.com.ar/Dic/grafico%20vectorial.php
- Tomado de: http://www.aulaclic.es/flashMX/b_10_1_1.htm
- Tomado de: http://www.tufuncion.com/graficos-vectoriales
- Tomado de: http://adobeperson.com/3ds-max/3d-model-mathematical-compass-tutorial-in-3d-max

- Tomado de: http://www.nodo50.org/arevolucionaria/masarticulos/marzo2003/comportamientoideal.htm
- Tomado de: http://es.wikipedia.org/wiki/Frustum
- Tomado de: http://trevinca.ei.uvigo.es/~formella/doc/ig02/ig_proy.pdf
- Tomado de: http://idam.ladei.org.ar/Tutoriales/Direct3D/Entendiendo_TR_Fija_Proy.html
- Tomado de: http://www2.ravensoft.com/users/ggribb/plane%20extraction.pdf
- Tomado de: A Modular Framework for Artificial Intelligence Based on Stimulus Response Directives. Autor: Charles Guy Año Edición: 2000.
- Tomado de: Programming Game AI by Example. Autor: Mat Buckland.

BIOGRAFÍA

Natalia Verenice Ortiz Duque

DATOS PERSONALES

- Fecha de Nacimiento: 10 de Octubre de 1984
- Lugar de Nacimiento: Quito – Ecuador

EDUCACIÓN

- Escuela Santa María Eufrasia
- Colegio Simón Bolívar
- Bachiller en Ciencias, Especialidad Informática. Año: Julio 2007

Biografía

Christian Andrés Viteri Proaño

DATOS PERSONALES

- Fecha de Nacimiento: 14 de Abril de 1984
- Lugar de Nacimiento: Quito - Ecuador

EDUCACIÓN

- Escuela: Pensionado Borja 3
- Colegio: San Gabriel
- Especialidad Físico Matemático.

HOJA DE LEGALIZACION DE FIRMAS

ELABORADO POR:

_____

Srta. Natalia Verenice Ortiz Duque

_____

Sr. Christian Andrés Viteri Proaño

COORDINADOR DE LA CARRERA

_____

Ing. Danilo Martinez.

Sangolquí, 22 de Mayo del 2009.