MATHEW: MATHEMATICS ENTERTAINMENT WORLD

CARLOS ALFREDO LIÉVANO MARTÍNEZ-VILLALBA
JUAN GABRIEL LIÉVANO MARTÍNEZ.VILLALBA

Proyecto de grado

Director:
Fernando Antonio Rojas Morales
Ingeniero de Sistemas

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA DE SISTEMAS
BUCARAMANGA
2006

Nota de aceptación:

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Bucaramanga, 19 de Enero de 2007

CONTENIDO

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

INTRODUCCIÓN

Como resultado de la dificultad observada en estudiantes para la apropiación de conocimientos matemáticos, en el curso de la experiencia docente de los autores, se establece una propuesta pedagógica para lograr este objetivo, basada en el uso de un juego computacional. Esta propuesta ha encontrado un mayor sustento en la teoria del desarrollo cognitivo y en ciertos puntos de los métodos construccionistas. En este informe se hace un detalle de estos puntos y su relación con el producto final del proyecto, así como algunos elementos considerados para lograr el atráctivo lúdico del mismo, obtenidos posteriormente a la presentación del anteproyecto.

La contextualización del mismo dentro de la realidad del país y de las necesidades locales, llevó a hacer algunos cambios en materia tecnológica con respecto a las soluciones inicialmente planteadas. Sin embargo, las consideraciones que llevaron a esta modificación se presentan en el contenido del presente documento y en el transcurso del mismo se verá como estas desiciones verdaderamente contribuyen al alcance de objetivos que superaron las expectativas de los autores en cuanto al alcance del prototipo.

En el diseño original se mostraba de manera muy sencilla los conceptos implícitos en la elaboración del juego y la construcción de conceptos matemáticos. Claramente el prototipo terminado da muestra de varias de las características permitidas por la tecnología, llegando a cubrir la implementación de conceptos avanzados en el desarrollo de juegos y procesamiento de gráficos.

Existe aún camino por recorrer por cuanto las posibilidades tecnológicas han llevado a que las expectativas de los usuarios en cuanto a calidad artística sean altas, y por lo tanto que proyectos comercializables requieran del trabajo y colaboración de un equipo amplio de personas expertas en varias disciplinas. No obstante, el prototipo cumple con su objetivo de ilustrar las posibilidades y el potencial a los cuales está sujeto el proyecto.

1. EL VALOR DEL JUEGO EN LA EDUCACIÓN

Tal y como se hubo expresado en anterior ocasión, el origen de este proyecto se dá en virtud de las posibilidades percibidas en el atractivo de los juegos de rol como estímulo para el aprendizaje de ciertas competencias matemáticas, basados en el auto-descubrimiento de la fantasia simulada en este tipo de juegos en conjunción con el mismo proceso sobre el mundo y lenguaje de las matemáticas.

No obstante este tema ya ha sido tratado en el anteproyecto, se considera importante agregar en esta oportunidad algunos elementos adicionales obtenidos a partir de investigaciones recientes en el campo de la educación, particularmente en el uso que se puede dar a los computadores en este área que ha sido un constante referente de investigación para la humanidad.

1.1 ELEMENTOS QUE HACEN DIVERTIDO UN JUEGO

Para conocer estos elementos se va a seguir la taxonomia propuesta por Malone[1]. No obstante su enfoque se hace con orientación a conocer que hace un juego divertido, hoy en día todavía no se ve claramente un seguimiento de estos parámetros, incluso en las modernas propuestas comerciales que ciertamente podrían beneficiarse de agregar más valor a los consumidores en cuanto al factor diversión se refiere.

El críterio fundamental con que Malone define su taxonomía es el de motivación intrínseca, definida por Lepper y Greene[2] en contraposición a la motivación extrínseca. En términos generales, esta teoria consiste en la diferencia existente en lo que motiva a un individuo a llevar a cabo una actividad por la actividad en si misma, en lugar de un conjunto de elementos motivacionales externos. Una ampliación de este concepto ha surgido en el estudio de agentes económicos para análizar el verdadero impacto de la compensación económica que resulta evidentemente en un factor motivacional extrínseco y por lo tanto no está intimamente relacionado con las actividades inherentes al trabajo que dicho agente llega a desempeñar[3].

---

[1] MALONE, Thomas W. What makes things fun to learn? : Heuristics for designing instructional computer games. En : ACM SIGSMALL SYMPOSIUM (3° : 1980 : Palo Alto) Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems. Palo Alto : ACM, 1980. p. 162-169.

[2] LEPPER, Mark R. and GREENE, David. The hidden costs of reward : New perspectives on the psychology of human motivation. Hillsdale, New Jersey : Lawrence Erlbaum, 1979. 276 p.

[3] SLIWKA, Dirk. On the hidden costs of incentive schemes. En : WORKSHOP EMPIRICAL PERSONNEL ECONOMICS (1° : 2004 : Bonn) IZA conferences program. Bonn : IZA, 2004. 38 p.

Para un ejemplo en el caso educativo, un estudiante puede estar motivado a aprender por un interés adecuado sobre la materia, la satisfacción obtenida del logro de los objetivos, una nota, el premio de sus padres y el reconocimiento de sus compañeros y docentes. Los dos primeros son claramente aspectos intrínsecos, mientras los últimos son todos aspectos extrínsicos.

Teniendo esto en mente, Malone divide los elementos motivacionales de los juegos, con especial enfasís a los juegos educativos, aun cuando no excluyente de otras situaciones y contextos, en tres: *Desafio*, *Fantasía* y *Curiosidad*.

1.1.1 Desafio.

Para lograr que un juego sea desafiante, Malone considera que es necesario de una *meta* cuyo resultado sea *incierto*. Obviamente el mejor diseño gráfico, la inclusión de una banda sonora cautivante y el uso de efectos visuales sorprendentes, no logrará más que un gran impacto inicial pero con el tiempo no garantiza que la calidad de la experiencia del jugador retenga su atención y compenetración con el juego. Un objetivo práctico, claro, y de dificultad apropiada, junto con un contexto que permita al jugador conocer su avance hacía el logro del mismo, puede ayudar a garantizar que la calidad de la experiencia sea envolvente.

No obstante, una meta que siga estos críterios no garantiza por si misma que el juego sea desafiante según la definición de Malone. La *incertidumbre* es el otro elemento que ayuda a artícular este propósito. Cuando el resultado de alcanzar una meta es obvio o evidente del contexto en que se desarrolla el juego, aún antes de alcanzarla, el nivel de motivación decae. Esto está intimamente relacionado con el concepto de *Curiosidad* que se tratará más adelante.

En primera instancia, esta incertidumbre se puede aproximar dando la alternativa de diferentes niveles de dificultad o una dificultad adaptativa de acuerdo a la capacidad del jugador. Esto tiene la ventaja que en caso que el jugador perciba como fácil el alcanzar sus objetivos, no sabe a ciencia cierta su capacidad para lograrlos en el siguiente nivel.

Otra manera de lograr incertidumbre consiste en crear metas u objetivos en diferentes niveles. Para enteder esto se puede recordar el juego *Memoria*. Este juego consiste en encontrar parejas correspondientes en un conjunto de tarjetas dispuestas boca abajo, destapando un par a la vez. En un primer nivel el objetivo es claro y consiste en encontrar todas las parejas disponibles. En un segundo nivel se podría establecer como meta encontrarlas todas en el menor tiempo posible o en el menor número de pares destapados. A este nivel se podría agregar el elemento de un competidor.

Como se puede observar este mecanismo permite que un jugador que pueda pasar facilmente el primer nivel de objetivos, pueda tener un desafio adecuado en los objetivos de niveles más altos.

Otras alternativas incluyen ocultar información e incluir eventos aleatorios durante la ejecución del juego. La primera de estas consiste en ir revelando porciones de la información, de tal manera que se vaya generando mayor desafio al no contar con la totalidad de elementos necesarios, pero dejando los suficientes para resolver el obstáculo. Adicionalmente, genera *curiosidad*, lo cual será discutido más adelante. Del mismo modo, los eventos aleatorios son un claro mecanismo para permitir generar el elemento de *incertidumbre*.

1.1.2 Fantasia.

Al involucrar fantasias en propuestas lúdicas, se pueden clasificar entre *extrínsecas* e *intrínsecas*. Una fantasia *extrínsica* es aquella cuyo desarrollo está ligado al cumplimiento de una habilidad, pero la habilidad no está involucrada dentro de la fantasia como tal. Tal es el caso del popular juego ahorcado, que relaciona el adivinar una palabra una letra a la vez (habilidad lexical), con el ahorcamiento de un individuo una parte corporal a la vez (fantasia *extrínseca*).

En el caso de las fantasias *intrínsecas*, no solo el desarrollo de la fantasia depende del desempeño en la habilidad por parte del jugador, sino que la habilidad esta relacionada con la fantasia planteada. Tal es el caso de simuladores como SimCity, en el cual la habilidad de planeación esta relacionado con el desarrollo de la supuesta ciudad administrada.

Las fantasías *intrínsecas* tienen ventaja sobre su contraparte en el sentido que sugieren posibles usos de la habilidad usada en otras situaciones del mundo real, o en el caso de fantasias intimamente relacionadas con la habilidad que se pretende enseñar, otorgan al jugador de una base sobre la cual puede construir por analogía inferencias en el dominio del nuevo conocimiento que está adquiriendo.

La fantasia adicionalmente involucra un componente emocional a la experiencia del jugador, al llenar algunas de sus necesidades de diversión, satisfacción, autoestima, y otras de acuerdo a cada individuo. Esto nos lleva a pensar que cada individuo puede tener una necesidad emocional diferente que busca cubrir con un juego determinado, y de ahi, que este sea un factor determinante en el gusto que sienta por el juego como tal.

Una de las ventajas de los juegos de rol es que permite cierto nivel de adaptación con respecto a estas necesidades emocionales, al permitirle en muchos casos que sea el jugador quien escoja las características que van a definir a su personaje. Si bien el prototipo no llega a este nivel de adaptación,

claramente las alternativas como género, apariencia y habilidades del personaje pueden resultar en una mayor aceptación del juego en una implementación comercial.

Prueba de esto se puede observar en el gran exito que ha tenido el juego de simulación Los Sims, el cual permite al jugador escoger elementos como género, apariencia, personalidad, profesión, amigos, pareja, y muchos más. Estos elementos han garantizado que el juego tenga gran aceptación incluso dentro de la población femenina que históricamente son menos atraidas hacía los juegos de video. Probablemente, la posibilidad del mismo a ajustarse a una imagén con la que la mujer se pueda identificar y las alternativas que se ajustan a diferentes fantasias explorables, y por lo tanto a distintas necesidades emocionales, han garantizado el exito comercial y la participación del género femenino en la comunidad global de Sims.

1.1.3 Curiosidad.

La curiosidad consiste del deseo de aprender, independientemente de cualquier objetivo, meta o satisfacción de necesidades. Según Malone, esta se estimula en entornos donde el ambiente presenta suficiente complejidad de información disponible para que el individuo aprenda, pero sin que sea demasiado sencilla como para que pierda todo sentido de novedad o sorpresa. En terminos de juegos esto quiere decir que un ambiente optimo sería aquel en que el jugador genera expectativas con respecto a lo que va a acontecer, pero en el cual estas expectativas algunas veces no son realizadas.

La curiosidad también puede dividirse en dos categorias: Sensorial y Cognitiva. La primera claramente esta relacionada con la atracción que pueden significar los cambios de estimulos sensoriales como la luz, sonido y tacto. La curiosidad sensorial puede despertarse con el uso de estimulos decorativos, propiciando el ambiente o estado de animo que se quiere transmitir, como recompensa o para representar situaciones.

La curiosidad cognitiva por otro lado consiste en la busqueda de mejorar las estructuras de conocimiento que se poseen. Estas se mejoran dandoles tres caracterísiticas importantes: *Completitud*, *consistencia* y *simplicidad*. Un conocimiento incompleto del entorno, la historia o de las habilidades requeridas pueden estimular al jugador a seguir adelante. La inconsistencia entre dos posibles realidades puede causar curiosidad para conocer aquella que es verdadera. La falta de simplicidad en los motivos de un evento o suceso puede impulsar cierta curiosidad por conocer las causas subyacentes y simplificar de esta manera lo aprehendido.

Es claro que en el caso del proyecto, el juego ha sido diseñado siguiendo la estructura de un juego de rol, que por su narrativa inherente es totalmente susceptible de estimular la curiosidad tanto sensorial como cognitiva. La parte

sensorial evidentemente estará propiciada por medio de los elementos artísticos, visuales y musicales que le den mayor atráctivo al juego en su implementación comercial, pero que evidentemente no se han desarrollado a profundidad para efectos de prototipo.

La curiosidad cognitiva, que evidentemente es la que ofrece un factor motivacional más perdurable, es estimulada facilmente desde la construcción de una historia principal que es el eje estructural del juego como tal y el que le dá mayor cohesión, sin que necesariamente esta se encuentre completa, consistente y simple en un comienzo. El diseño del prototipo finalmente se justifica en la posibilidad de establecer un acompañamiento entre la curiosidad cognitiva que se deriva de la busqueda de aprender la historia y la que es parte integral del aprendizaje de las matemáticas, y que pueden hacer de este un proceso divertido.

1.2 DEL CONSTRUCTIVISMO AL CONSTRUCCIONISMO

El constructivismo como teoria del aprendizaje fue formulada y desarrollada por Jean Piaget a mediados del siglo pasado. Como tal está enmarcada en la teoria del desarrollo cognitivo, la cual formula la existencia de cuatro estados o niveles de desarrollo cognitivo que afectan la evolución del pensamiento del niño desde sus primeros años hasta su adolecencia. Estos estados incluyen:

- Sensomotriz (Experimenta a través del movimiento y los sentidos)
- Preoperacional (Adquiere habilidades motrices)
- Operacional Concreto (Pensamiento lógico con relación a eventos concretos)
- Operacional Formal (Capacidad de razonamiento abstracto)

Para Piaget, el paso de un nivel al siguiente es resultado de la acumulación de errores conceptuales que catalizan la generación de nuevas estructuras mentales en el niño. Es decir que cada etapa es un requisito necesario para la construcción de la siguiente, y por lo tanto, la evolución del aprendizaje y el pensamiento del niño es resultado de un proceso al interior del mismo y no como resultado de tomar lo que le aporta el entorno.

Es aqui donde el constructivismo cobra importancia. Como resultado de la mencionada teoria, el constructivismo promueve el aprendizaje orientado al desarrollo de estas etapas, iniciado y dirigido por el estudiante, pero apoyado por un adulto o maestro que le ayuda en la revisión de la validez de sus construcciones.

Existen algunas concepciones erradas en cuanto a esta definición. La primera y más común es que esta es una teoria de como enseñar, cuando en realidad es, como se ha definido acá, una teoria sobre el aprendizaje. Es decir, sobre la manera en que el estudiante se apropia del conocimiento y del mundo que lo rodea. Esta concepción ha conducido a que muchos asocien al contructivismo

con la metodología de aprender haciendo. Lo cierto es que este proceso de aprendizaje no necesariamente está relacionado con algún “hacer”, sino con la manera en que se construyen las estructuras mentales involucradas en dicho proceso.

El construccionismo igualmente está asociado a esta idea de aprender haciendo. Esta teoria fue promovida por Seymour Papert, quien trabajó con Piaget durante los sesentas, y esta basada en las ideas de este último con respecto al constructivismo y la teoria del desarrollo cognitivo. Tan íntima es la relación entre las teorias de ambos que el principio que explica los resultados de las investigaciones de Piaget es conocido como el *Principio de Papert*.

El *Principio de Papert* establece que algunos de los pasos cruciales en el crecimiento mental están basados no solo en adquirir nuevas habilidades, sino en adquirir nuevas maneras administrativas de usar lo que uno ya conoce. Esto claramente se encuentra incorporado en el corazón de la teoria del desarrollo cognitivo al hacer alusión al paso entre los distintos estados a través de la construcción progresiva sobre los estados precedentes. Pero igualmente guarda íntima relación con la posición más pragmática que asume Papert al buscar maneras de lograr la enseñanza adecuada, más no tanto en el desarrollo del proceso de aprendizaje.

La mayor parte del trabajo de Papert está enfocado a la enseñanza de las matemáticas y las ciencias, e involucra el uso de la tecnología, siendo este el creador del lenguaje de programación *Logo*, orientado principalmente para facilitar la programación de computadores por parte de los niños. Este esfuerzo ha sido continuado por la *Logo Foundation.*

El construccionismo por el contrario si es una teoría de como se puede enseñar. Sin embargo, siguiendo la propia naturaleza metodológica, el mismo Papert se abstiene de definirlo, esperando que alguien que se interese por el mismo participe de la discusión en torno a lo que es y supone el concepto y entre todos los participantes deben llegar a converger en una visión común o aproximada[4]. Como tal, hace homenaje a la virtud de la diversidad y en su confianza por dicha convergencia se puede llegar a hacer alusión a un socialismo intelectual implícito, donde todos aportan sus puntos de vista para esa construcción común de la cual todos obtienen provecho.

Como consecuencia de esta carencia de una definición explícita, su evolución ha permitido su aplicación a un sinnúmero de campos y situaciones. La investigación actual intenta involucrar el uso de tecnología computacional, aunque se han tenido experiencias donde la aplicación carece de este uso. Lo que si es común a estas experiencias es que en todas ellas los estudiantes de encuentran comprometidos en el proceso de construcción del conocimiento y el maestro actua como un guia que es partícipe del proceso.

[4] PAPERT, Seymour. Contructivism. Westport, Conneticut : Ablex Publishing, 1991. 518 p.

Claramente el juego propuesto no guarda relación con este tema común de las experiencias derivadas de la teoria de Papert por cuanto no presupone la participación de un guía para evaluar la calidad de la construcción de conocimiento generada ni para participar en la construcción misma. Sin embargo, si cuenta con la propiedad de permitir que sea el jugador quien se haga responsable de la construcción del conocimiento matemático relevante, con el valor agregado de involucrar elementos motivacionales adicionales a los que permite la propia construcción al complementarla con la actividad de jugar.

Se podría llegar a caer en la tentación de involucrar el juego en un proceso de enseñanza tradicional, donde el maestro formaliza las construcciones obtenidas por el estudiante en la solución de los retos enfrentados. No obstante, este es un riesgo que es recomendable no correr por cuanto perdería tanto los beneficios motivacionales inherentes al juego, y perdería también los elementos construccionistas en los cuales el estudiante ya se había involucrado y que la formalización del profesor podría llegar a desbocar.

Por el contrario, podría ser recomendable si se usara en un entorno un poco más formal, que una sesión de juego sea seguida por una sesión de discusión en la que sean los estudiantes quienes aporten puntos de vista y perspectivas obtenidas durante el logro de los retos establecidos. De esta manera es el mismo quien dará una estructura formal al conocimiento adquirido y la participación del profesor habría de estar orientada hacía la moderación de esta discusión y solo intervendría en los casos que la construcción obtenida se desvíe de la pretensión original. Dicha intervención sería preferible desde un enfoque dialéctico que permita que sean los estudiantes quienes corrijan la construcción a la que hubiesen llegado y busquen la construcción comun que la teoria construccionista pregona.

2. TECNOLOGÍA USADA

En el desarrollo del presente proyecto se ha hecho la selección de diversas alternativas ofrecidas para el desarrollo de los elementos que componen el producto final. Se considera importante entender cuales fueron los criterios que perfilaron las decisiones tomadas y las consecuencias que estas pueden tener para futuros desarrollos del **software**.

2.1 LENGUAJE DE PROGRAMACIÓN

El lenguaje de programación elegido desde un principio fue *C++*, teniendo en cuenta su alto rendimiento, su portabilidad y su capacidad para la implementación de un diseño orientado a objetos. La decisión resultó acertada por cuanto las ventajas establecidas se materializaron durante el desarrollo del proyecto, facilitando no solo en la implementación inicial, sino también las distintas iteraciones de diseño, que buscaban darle nuevas dimensiones al alcance del proyecto.

La elección de un diseño orientado a objetos fue un factor clave para permitir el redimensionamiento del proyecto, agregando nuevos atributos y metodos de acuerdo a la complejidad que se iba requiriendo. La sutil interrelación entre las clases que conforman el mundo creado era más fácil de monitorear con este diseño, y por lo tanto facilitaba el mantenimiento del código con cada actualización.

El impacto en el resultado final es claro: El proyecto evolucionó de un entorno con materiales solidos, enemigos sin ningún rumbo partícular y sin ninguna animación, y un solo reto que requería estructuras matemáticas, pasando a un proyecto que incluía diferentes instanciaciones de las clases para terceras personas, uso de materiales más elaboradas en los fondos, incorporación de conceptos de iluminación, sombras y efectos, integración de inteligencia artificial para permitir la persecución del personaje principal, la incorporación de un segundo reto, y animación para distintas articulaciones de cada uno de los personajes.

2.2 API GRÁFICO

La interface escogida para el desarrollo del proyecto es *OpenGL*, que representa **Open Standard Graphic Library**. *OpenGL* es una interface para programación de aplicaciones, o API por su sigla en inglés, que permite al programador, por medio de un conjunto de funciones, hacer uso del **hardware** disponible para la generación de gráficos por computador.

Para entender los motivos que llevaron a escoger esta API sobre su más fuerte competidor, *Direct3D*, se debe entender un poco mejor la historía y la evolución de ambas alternativas.

2.2.1 Historia y origen.

El desarrollo de la industria de los gráficos digítales ha visto la evolución de las dos APIs mencionadas en la introducción de este título: *OpenGL* y *Direct3D*.

OpenGL, fue originalmente desarrollado por Silicon Graphics en 1992 como remplazo a otro API conocido como Iris GL, que se basaba en UNIX. Como ya se había expresado, **Open** representa que es un estandar abierto, en lugar de un código abierto. Esto quiere decir que un usuario cualquiera puede tener acceso al estandar e implementarlo, sin perjuicio de que existan regalías aplicables, y en algunos casos, como lo es el de *OpenGL*, que este sea aprobado y modificado por el consenso de un comité formal abierto a todos los interesados. *OpenGL* es supervisado por la *Junta de Revisión Arquitectónica* o *ARB* por su sigla en inglés, la cual se reune cada tres meses y está integrada con derecho a voto por 3DLabs, Apple, ATI, Dell, IBM, Intel, nVidia, SGI, y Sun Microsystems. Entre reuniones las compañias y organizaciones miembro suelen participar activamente a través de correo-e y teleconferencias.

En su nacimiento, *OpenGL* fue diseñada siguiendo los parametros de lo último en tecnología de punta para estaciones de trabajo de alto nivel, con una clara visión de lo que el futuro traería. Como resultado, este *API* ha sido siempre estable y consistente. No obstante, el funcionamiento consensual del *ARB* ha hecho que el **core** de la especificación haya evolucionado lentamente, llegando tan solo a su versión 1.3 en el transcurso de más de 10 años. Esta demora ha sido superada por parte de los desarrolladores haciendo uso del mecanismo de extensión, permitiendose estar al tanto de los últimos adelantos en **hardware**, aun cuando no siempre tenga los mejores resultados en cuanto a la simplicidad del código. El *ARB* ha reconocido este problema y ha acelerado el proceso de actualización del **core**.

OpenGL es una especificación basada en *C*. Por esto, su estructura está orientada hacía un paradigma de programación procedimental o estructurado, aunque en sus últimas versiones ha ido adquiriendo cada vez más un paradigma orientado a objetos, y desde su comienzo ha permitido su implementación en gran variedad de lenguajes diferentes a *C*. Por otro lado, cuenta con la ventaja de dejar la administración de los recursos de hardware a la implementación, permitiendo a un usuario básico lograr aplicaciones sin necesidad de conocer las operaciones de menor nivel.

En cuanto a *Direct3D*, esta es el resultado de una estratégia comercial que Microsoft requería para estimular el desarrollo de juegos sobre su sistema operativo *Windows*. La realidad antes de su lanzamiento era que la mayoría de los juegos eran lanzados para correr sobre *DOS* y el acceso al **hardware** de

video y sonido a través de la plataforma *Windows* era limitado y dificil. En consecuencia, Microsoft desarrolló para 1995 una *suite* de APIs que permitieran acceder directamente al **hardware** y facilitar de esta manera el trabajo de los desarrolladores para incrementar el lanzamiento de nuevos títulos y de igual manera la popularidad de *Windows.*

Para el desarrollo de *DirectX 1.0*, Microsoft integró el API 3D producido por una empresa llamada RenderMorphics, y su propia versión de una librería gráfica conocida como *Game SDK*. Lo cierto es que esta solución no resultó tan efectiva como esperaban, al terminar siendo compleja, mal estructurada y lenta.

Sin embargo, el esfuerzo de Microsoft no terminó ahí, quienes junto con miembros de la industria fueron construyendo versiones mejoradas y más adecuadas a las necesidades de la comunidad de desarrollo de juegos. Se puede decir que esto fue causa del factor diferenciador de este API con relación a *OpenGL*, que como ya se dijo, estaba orientado a estaciones de trabajo de alto nivel. Este proceso de mejora llevó al API de Microsoft hasta la versión 7.0, que fue la primera versión que obtuvo amplia aceptación.

Actualmente, *DirectX* se encuentra en su versión 9 y su funcionalidad ha mejorado notablemente, teniendo en cuenta su origen rocoso. Direct3D es el API controlador del hardware acelerador de gráficos 3D, que forma parte de *DirectX*. Adicionalmente cuenta con interfaces que permiten la interacción con tarjetas de sonido y periféricos de entrada como ratones, teclados y **joysticks**.

DirectX se encuentra basado en el modelo *COM* de Microsoft. Esto implica que sus APIs son facilmente implementados en lenguajes como Visual Basic y Visual Basic Script. Adicionalmente le da la ventaja añadida de facilitar la compatibilidad retrospectiva al mantener los componentes más viejos en la base de su estructura y construir las características mas recientes a partir de ahí. Esto trae como efecto que correr un programa que se basaba en una versión anterior de *DirectX* encontrará los mismos componentes necesarios en una versión posterior.

2.2.2 Ventaja fundamental de *OpenGL* sobre *Direct3D*.

Tal y como se pudo evidenciar de la evolución de cada uno de los API evaluados, existen claras ventajas de cada uno sobre el otro. No obstante, en el momento de tomar una desición sobre el estandar a adoptar para la ejecución del proyecto, el factor que se presenta a continuación fue vital para inclinar la balanza hacía *OpenGL*.

Dados sus origenes, *Direct3D* ha permanecido como una solución orientada solo a los sistemas operativos de *Windows* aún cuando ha aceptado la participación de la comunidad más para orientar sus esfuerzos hacía los

requerimientos de la industria y hacia las últimas innovaciones incorporadas por los desarrolladores de hardware para video y gráficos en sus soluciones comercializadas.

Por el contrario, *OpenGL* al permanecer bajo la supervisión de la *ARB* ha permitido la participación de un grupo más heterogeneo de empresas y organizaciones, con lo cual la portabilidad del estandar ha perdurado, permitiendo implementaciones exitosas en sistemas operativos diferentes.

Teniendo en cuenta la realidad nacional y los altos costos de las licencias de *Windows*, resulta claro que una implementación que se base en este sistema operativo no va a tener mejores posibilidades para generar un impacto profundo en estimular el desarrollo de alternativas educativas viables para la mayoría de nuestra población. La posibilidad que da *OpenGL* de hacer uso del software educativo en sistemas operativos más económicos y accesibles como *Linux*, facilitaría la penetración suficiente para generar el máximo impacto posible, en la medida que el valor de los equipos computacionales siga en disminución y las iniciativas gubernamentales, y particularmente ministeriales, en el tema de incentivar el desarrollo de las TICs sean cada vez mas cercanas a construir la realidad nacional.

Con esto en mente la decisión de tomar el *API* de *OpenGL* como base para la implementación del proyecto en discusión resulta coherente con la busqueda de soluciones de ingeniería que sean adecuadas a la realidad contextual donde se desarrolla el mismo, que esten ligadas con las necesidades que se quieren satisfacer y no solo con las virtudes intrínsecas de la tecnología.

Lo cierto es que el desarrollo de ambos *APIs* ha minimizado las diferencias distintas de la portabilidad, como resultado de la condición de jugador dominante que favorece a Microsoft y sus iniciativas, sin demostrar interés alguno por parte de esta empresa por cerrar esta brecha. En estas condiciones es importante distinguir el efecto que esta diferencia remanente tiene bajo las condiciones económicas locales.

2.3 API PARA AUDIO

Teniendo en cuenta los argumentos que se plantearon para la elección del *API* a usar para el desarrollo del entorno gráfico, resulta claro que no tendría sentido alguno el usar los demás *API* de la *suite DirectX* para las demás necesidades del proyecto. *OpenGL* como su nombre lo indica es una libreria gráfica y en consecuencia no cuentan con funciones que sirvan de interface con el **hardware** de audio.

Guardando entonces coherencia con dichos argumentos, se optó por implementar las necesidades de sonido por medio del estandar *OpenAL*, liderado por Creative Labs como un complemento a *OpenGL* a partir del 2000.

Esta elección igualmente permite la mplementación de código listo para uso multiplataforma.

El uso de este *API* ha venido adquiriendo mayor aceptación desde que Loki Entertainment lanzara los primeros juegos que se basaran en el estandar, gracias al impulso que Creative Labs le ha dado principalmente al incluir soporte a las librerias del estandar en sus diferentes desarrollos de **hardware** para audio, como sus populares tarjetas *SoundBlaster*.

2.4 PERIFÉRICOS DE ENTRADA

Con el fin de mantener la ventaja obtenida de usar *OpenGL* y *OpenAL* como *APIs* para la implementación de las necesidades del proyecto en cuestión de gráficos y audio, respectivamente, se llevó a cabo una implementación particular para la presentación y recepción de texto y comandos. El teclado se eligió como único elemento de interface para estas tareas.

Como paso inicial se procedió al diseño en bits blanco y negro de una fuente partícular que se puede ver a lo largo de todas las situaciones que involucran texto dentro del juego. Esta fue diseñada para cada uno de los caracteres que tienen su representación en el código ASCII.

A continuación, se crearon funciones globales y publicas que permiten enlazar cada uno de los caracteres en la formación de una oración, y que relacionan los caracteres ASCII leidos del teclado, con los caracteres que se presentan en la interface. Dichas funciones estan basadas sobre el estandar *OpenGL* con el fin de dar implementación a los cuadros de texto que permiten al jugador hacer seguimiento al dialogo entre los distintos personajes que encuentra en el curso del juego.

La entradas al teclado se leen a partir de una función implementada según la especificación del *OpenGL Utility Toolkit*, o *GLUT* por su sigla en inglés, que monitorea dichas entradas. La función implementada hace un llamado a la fuente diseñada para el juego y es modificada en su apariencia a través de *OpenGL* para dar efectos como el color de los caracteres en los cuadros de texto generados.

3. STORY IS KING

A continuación se presentará la introducción a la historia que ha de ser el eje central del juego de rol representado por el prototipo. El título del presente capitulo hace referencia a una frase que constituye un slogan interno en *Pixar Animation Studios*, en Hollywood, California, con el cual hacen alusión a la importancia que una adecuada historia tiene en el exito de cada uno de sus proyectos de animación.

3.1 UN DÍA COMO CUALQUIER OTRO

El personaje del juego es un estudiante cualquiera, de grado cualquiera, en un colegio cualquiera, quien asiste a sus clases en un día cualquiera. Ese estudiante es Yo.

En este día cualquiera, Yo tiene la entrega de los resultados de un extenso examén de matemáticas en el que cree haber obtenido un buen resultado. Sin embargo, en el momento en que su profesor le hace entrega de los mismos, Yo se da cuenta que los resultados no han sido alentadores.

En medio de su bajo estado de animo originado por estos resultados inesperados, el profesor hace una interrupción para anunciar que Mathew ha obtenido un resultado perfecto.

¿Cómo es esto posible?

¿Quién es Mathew?

Ha de ser el ocupante de esa silla que ha permanecido desocupada durante todo el año.

Pero, si nadie lo ha visto, cabe preguntarse como es que ha logrado obtener puntaje perfecto, dejando de lado incluso el hecho que no habría podido presentar el extenso examén.

Las inquietudes son bastantes. Y el lamentable resultado de Yo deja también varios interrogantes al crear un vacio en cuanto a su desempeño en matemáticas se refiere.

3.2 UNA PUESTA EN COMÚN

Al salir al recreo en este día cualquiera, Yo se encuentra con tres de sus compañeros y comparte los interrogantes que le atribulan. Los resultados de

ellos no son mas alentadores que los obtenidos por Yo y sienten la misma incertidumbre al desconocer los motivos que los llevaron a esa puntuación catastrófica.

No obstante contar con estas inquietudes académicas, existe un elemento adicional que alimenta su inquietud:

¿Quién es Mathew?

Ha de ser el ocupante de esa silla que ha permanecido desocupada durante todo el año.

El asunto requiere de una investigación exhaustiva. Y solo Yo y sus amigos están preparados para comenzar la busqueda de este estudiante cuya identidad les elude.

No cuentan con más que un nombre y un puntaje perfecto. Ah, y un gran número de inquietudes académicas!

Quiza la busqueda de Mathew les revele respuestas que los acerque al motivo por el cual sus resultados no fueron mejores.

3.3 EL MUNDO DE MATHEW

De repente, el cielo que hasta el momento se encontraba liderado por un sol inclemente se cierra de nubes oscuras y espesas. Impone el orden en este recreo desordenado con el azotar de las gruesas gotas que lanza sobre el arenero, el parque, los juegos de madera, y los niños que corren a buscar refugio bajo ese techo que alberga la fuente del saber.

Yo y sus amigos recorren el penoso camino de regreso al aula de clase, con sus lamentables resultados agarrados en una mano, su ropa mojada pegada en sus cuerpos, y la idea de Mathew en la cabeza.

Yo se sienta en su escritorio a revisar su examén.

Lo toma por sorpresa la idea de que tal vez el primer ejercicio había quedado bien.

¿El segundo también?

Muchos otros parecían seguir esta posibilidad.

– Acá huele a complot! – Murmuró para sí.

– El profesor no me quiere – Le chilló su a veces psicótica cabeza.

En ese instante una espesa niebla blanca comenzó a aparecer en el interior del aula, esparciendose bajo las patas de los escritorios y las sillas, mientras la voz del profesor se hacía cada vez más lejana...

Yo levantó la cabeza y descubrio un mundo que no había visto antes. Un mundo que parecía el campo de batalla de una de esas películas épicas, antes del inminente choque de los metales.

Suenan pasos.

Alguien viene.

Yo cuenta con su regla de 40 centímetros y un corazón que quiere escaparsele por la garganta.

Es mejor moverse y comenzar a correr...

4. ESTRUCTURA Y ARQUITECTURA

La arquitectura del proyecto es, como ya se había dicho, orientada a objetos. Aunque se aprovecharon también las características funcionales de *C++* y que a la vez son naturaleza del mismo paradigma utilizado en la implementación de *OpenGL*, el uso de clases ha sido de gran ayuda en el momento de actualizar el juego para generar nuevas posibilidades en su complejidad, diseño, e interfaz con el usuario, que se pueden dividir conceptualmente en tres grandes momentos o actualizaciones del juego.

4.1 ELEMENTOS BÁSICOS

Para el manejo de las formas básicas y su uso inmediato en el espacio tridimensional se ha implementado un modelo de clases que permite fácilmente crear nuevas instancias de dichos objetos, otorgando simplicidad en el momento de programar la funcionalidad indispensable para el juego. Esta creación de formas básicas permite explotar la característica de encapsulamiento de la programación orientada a objetos y empezó a utilizarse dentro del proyecto después de la segunda gran actualización del mismo. Anterior a la definición de estas en clases, se declaraban dentro de lo atributos de una clase, donde se hicieran necesarias.

Estas formas básicas se encuentran definidas en los archivos qqBasicGraphics.h y qqBasicGraphics.cpp[5]. En estos archivos se encuentran los siguientes elementos: valor de π; variables representativas para los ejes (x, y, y z) y las direcciones (norte, sur, oeste y este); funciones para dibujar cilindros sin el uso de *quadrics* de OpenGL, para cambiar de modo 3D a 2D, para volver a 3D, para creación de los caracteres e impresión de los mismos; y clases de punto, rotación, perímetro, color y plano.

Tabla 1. Clases básicas de manejo de gráficos.

qqPoint	La clase punto (*qqPoint*) contiene tres atributos que lo posicionan sobre el espacio tridimensional. Para esto se definen en tres flotantes que representan el valor en *x*, *y* y *z*. Se debe tener en cuenta que el punto es utilizado para representar elementos más complejos dentro del mundo virtual y por lo tanto no posee un método para dibujar el punto en el espacio. El punto contiene a la vez dos constructores, uno que recibe

[5] Todos los encabezados desarrollados han sido nombrados con el prefijo *qq* como distintivo para identificar funciones y clases propias.

	ningún parámetro y define el punto en el origen, y otro que recibe tres parámetros flotantes y que sitúa el punto en las coordenadas *x*, *y*, y *z* respectivas. Además de los constructores, el punto tiene definido tres métodos para obtener una coordenada determinada (*getX(), getY() y getZ()*) y otros tres métodos que trasladan el punto a través de un eje específico (*moveXaxis(difX), moveYaxis(difY) y moveZaxis(difZ)*).
qqRotation	La clase rotación (*qqRotation*) contiene un atributo para definir el ángulo de rotación y otros tres para determinar el nivel de influencia que ejerce el ángulo de rotación, es decir, si los tres niveles de rotación fueran 0.0, entonces el ángulo de rotación no tendría influencia y por lo tanto el objeto no rotaría. Si el nivel de rotación en el eje x fuera 1.0 y los otros dos 0.0, entonces el objeto rotaría sobre eje x. Los constructores de la rotación son dos al igual que en el punto, donde uno no recibe parámetros, inicializando todos los atributos con 0.0, y otro que recibe cuatro parámetros flotantes, definiendo el ángulo de rotación, y los niveles sobre el eje x, y, y z respectivamente. Posee cuatro métodos que obtienen los cuatro atributos de la clase (*getAngle(), getxAxis(), getyAxis(), y getzAxis()*), tres métodos para cambiar los atributos directamente: *getAngleXYZ(ang,x,y,z)* que cambia los cuatro atributos, *setAngle(ang)* que cambia solo el ángulo de rotación y *setRotationReference(x,y,z)* que cambia los niveles de influencia. Y por último, un método para rotar el objeto un ángulo determinado, *rotate(ang)* donde se suma *ang* al atributo del objeto.
qqPerimeter	La clase perímetro (*qqPerimeter*) que hereda de *qqPoint* posee solo un atributo que es el radio. Los constructores del perímetro son tres: uno sin argumentos (inicializa en 0.0), otro con un argumento para iniciar el radio con un valor determinado, y otro con cuatro argumentos para inicializar el radio, y la posición del punto en 3D. También funciona con un método para obtener el radio y otro para cambiarlo directamente. Esta clase facilita detectar colisiones cuando se programa la lógica del juego.
qqColor	La clase color (*qqColor*) es utilizada para definir los colores que se han de visualizar dentro del juego y es utilizada en texturas, luces, sólidos e interfaz. Al instanciarse un objeto de esta clase, se piden alores en escala para RGB o RGBA (Red-Green-Blue-Alpha) de 0.0 a 1.0. Los métodos que posee son solo para consultar estos valores (*getRed(), getGreen(), getBlue() y getAlpha()*).
qqPlane	Finalmente, como última forma básica y única que tiene representación en el espacio tridimensional, la clase Plano (*qqPlane*) que hereda de *qqPoint* y *qqColor*, es utilizada para

	dibujar planos dentro del juego. Aunque las superficies rectangulares son normalmente utilizadas en OpenGL junto con las superficies triangulares para dibujar modelos de alta complejidad, los planos definidos en el proyecto son solo para representar grandes planos que luego serán vistos en el juego como el suelo, el fondo y árboles (que utilizan una técnica de transparencia junto con dos planos para este propósito).

A partir de estas clases se obtienen una jerarquía aceptable para heredar propiedades gráficas a los distintos elementos que componen el juego. Así se aprovecha el potencial que otorga la implementación de herencia en el diseño de la estructura lógica.

4.1.1 Formato de imagen.

Otra clase básica que se utiliza pero que no afecta en el diseño del juego es el *CtargaImage* localizado en *CtargaImage.h* y *CtargaImage.cpp*. Esta clase maneja unos aspectos cruciales en el juego ya que permite importar información de archivos gráficos externos dentro del programa. El uso de esta clase se inicio en la última etapa de desarrollo del proyecto dado que requería de implementar su enlace con las rutinas de textura y *bitmaps* de OpenGL.

Anterior a la programación de *CtargaImage*, se contemplaron las varias alternativas para escoger el tipo de archivo gráfico estándar en el juego. Entre las extensiones disponibles para usar se encuentran: *bmp (bitmap), jpeg, tiff (Tagged Image File Format), png (Portable Network Graphics), y tga (Targa)*. Se termino escogiendo el formato Targa dada la extensa documentación que se encontró del formato, al igual de poseer capacidades para implementar en formatos de 8, 16 y 32 bits por pixel, dejando así posibilidad para el uso del canal alpha, o transparencia, en caso de ser necesitado.

El análisis que se desarrollo para la implementación del formato en conjunto con las necesidades de OpenGL fue de conseguir el algoritmo más rápido y sencillo para transformar el formato en un mapa de bits de tres bytes para cada canal de color (Rojo, Verde y Azul). En Targa se encontró además un amplio soporte, dado que es un estándar abierto al igual que *OpenGL* y tiene más tiempo de desarrollo que los nuevos formatos basados en vectores e índices de color.

La clase *CtargaImage* programada para MathEW se basa en las especificaciones estándar de Targa, y otras referencias soportadas por OpenGL que aunque no funcionales para el proyecto, sufrieron de modificaciones propias para su implementación correcta. A partir de la definición de esta clase y aquellas en *qqBasicGraphics.h* se prosigue al diseño

de estructuras más complejas que constituirán los aspectos visuales, lógicos y de flujo del proyecto.

4.2 EL MUNDO DE MATHEW

La interfaz gráfica del proyecto se encuentra estructurada para ser manejada en una clase mundo (*qqWorld*) la cual tiene como idea integrar todas las clases que se encuentran en un nivel del juego. Aún así, la clase mundo no hereda de ninguna otra clase y en vez contiene objetos de cada una de las otras clases dentro de ella. De igual manera, los métodos definidos dentro no manejan ninguno de los elementos del mundo sino que los define, controla su flujo y el momento de dibujarlos en la pantalla.

Tabla 2. Descripción de la clase qqWorld.

Método paint()	El método que dibuja todos los elementos del mundo en la pantalla (*paint()*) llama a otras métodos *paint()* que se encuentran en los objetos dentro de *qqWorld.* Al realizar esto asegura encapsulamiento del manejo en las capas externas del modelo orientado a objetos y además facilita modificaciones y cambios en el flujo normal cuando se cambia de estado en el juego.
Variables qqFocus	El juego posee dos estados fundamentales definidos en *qqFocus* como *QQ_BATTLE* y *QQ_MATHGAME*. El primero se presenta activo cuando el jugador se encuentra en batalla o caminando por el mundo de MathEW. El segundo se activa en el momento de interactuar con un objeto del juego que presente un desafío matemático. Cuando el usuario esta jugando en modo *QQ_BATTLE*, el flujo dentro del código es continuo llamando a la función de animación declarada para GLUT. En caso de encontrarse en medio de un desafío matemático, la función de animación es saltada y se entra en un flujo alterno en las funciones definidas para controlar la entrada y salida del programa. Desafortunadamente, este sistema no es el más conveniente pues se sale del enfoque orientado a objetos que se intenta implementar para el tamaño del proyecto y una modificación ha sido planificada para una siguiente actualización donde las interacciones con los objetos en MathEW son activados por los mismos objetos más que por intervención externa.
Atributos shapeW, shapeH	Entre los atributos de la clase mundo, se encuentran dos variables que son inicializadas en el momento en que el programa entra en la función que reconfigura la pantalla, para tener almacenado el tamaño de la pantalla durante todo el juego. Esto es necesario con el

	propósito de reinstaurar las propiedades de la cámara, la matriz de proyección y los modelos en el momento de saltar de modo 2D a modo 3D y viceversa.
Atributos back[], westSide[], eastSide[\	Para los fondos en el mundo de MathEW se han definido doce planos que muestran fondos para los lados y la parte de atrás. Cada lado esta compuesto de 4 planos que posteriormente se le es asignado una textura de 256x256 pixeles ya que el sistema solo permite texturas de 4096 pixeles por lado para mapas de un bit por píxel. Dado que las texturas importadas del formato Targa son de 16 bits por píxel, el máximo número de pixeles por lado pasa a ser de 256. De esta manera, cada lado posee una imagen de 1024x256 asegurando mejor resolución para la presentación del juego.
Atríbuto characterSet	Entre los objetos dentro del mundo, se encuentra un conjunto de personajes que incluye personaje principal, enemigos, maestros, un computador y árboles. En realidad, el motor que maneja los elementos que interactúan es la clase *qqCharacterSet* que abstrae un conjunto de “personajes”.
Atributos qqGame21, qqGameEC, conversation_focus, focus	Los atributos que facilitan el manejo del flujo del mundo y que se encuentran dentro de la clase son los cuadros de dialogo, las texturas, una variable para guardar el enfoque de conversación en la que se encuentra el juego, la entrada del usuario, la salida del programa y cuatro variables que incrementan o se reducen dependiendo del movimiento del personaje principal. Con estas últimas cuatro variables de busca establecer los límites se controlan que el personaje principal no salga del escenario. Otra variable importante es el enfoque de conversación ya que permite la continuidad de la historia para el juego.
Métodos readKeyboardInput(), readMouseInput(), paint(), moveWorld()	Los métodos de la clase mundo son parecidos a lo que son las funciones básicas de flujo para GLUT. Hay dos funciones que manejan la entrada/salida del programa que son *readKeyboardInput(key,x,y)*, *readMouseInput(button,state,x,y)* para entrada y *paint()* para salida. Otro método, *moveWorld(dir)*, controla el movimiento del personaje en el mundo hacia una dirección determinada. A la vez observa si la movida es permitida por el conjunto de personajes y en caso de recibir respuesta negativa de este, decide si detener el personaje principal, entrarlo en estado de desafío matemático o cambiar el personaje a modo de ataque en caso de ser un enemigo. Esta función, además de mover el conjunto de personajes, mueve cualquier otro objeto en el escenario.

Método moveCharacterSet()	A diferencia de la función *moveWorld(dir)* se ha definido una función *moveCharacterSet(dir)* que se encarga de mover todos los objetos dentro del conjunto de personajes en sentido contrario para asegurar una distancia relativa al personaje principal en todos los puntos del juego. Esta función es llamada cada vez que el personaje principal se mueve, sin embargo los movimientos seleccionados por inteligencia artificial son realizados independientemente dentro de un método *aiMove()* que se encuentra definido en la clase *qqCharacterSet.*
Método worldTurn()	Por último, el método *worldTurn(time)* hace que el mundo de MathEW funcione cuando el usuario no esta ejecutando ninguna instrucción. Esta función es muy sencilla y tiene un flujo muy convencional a excepción de una evaluación del lapso de tiempo en que el personaje principal ha dejado de atacar ya que los impulsos del teclado son más lentos que el procesamiento de un ciclo del juego, y así asegura que la animación del ataque sea realizada mientras la orden sea ejecutada por el usuario. El tiempo estimado para este lapso es de 0.1 segundos.

Otra característica en la definición de la clase del mundo son la declaración de estructuras para cada desafío matemático por aparte. Aunque para escalar las especificaciones técnicas del proyecto estos desafíos serian programados por aparte, declarándolos como instancias de una clase desafío que funcionen con *scripts* externos. La implementación de *scripts* externos no esta presente en este prototipo del juego debido a que la dimensión del mismo aún no justifica un sistema tan avanzado como lo es este. Para este caso, el uso de estructuras es suficiente y asegura velocidad y rápido acceso a las características del desafío.

4.3 LOS PERSONAJES

Los personajes son probablemente la estructura más compleja del juego y cuenta con funcionalidad y relaciones establecidas con casi cualquier otro elemento en el proyecto. Estos son divididos en dos clases para facilitar su implementación aunque realmente cuenta con otras dos clases que son definidas después como personajes por conveniencia de uso. Para otras actualizaciones contará con más clases según la diversidad de elementos que pueda contener un escenario.

Las dos clases principales que definen los personajes se encuentran en los archivos *qqCharacter.h* y *qqCharacter.cpp*. En estos archivos se definen todas las características y estructuras que puede poseer un personaje ya sea

principal o secundario. En una primera versión del proyecto, se separó la definición del personaje principal con la de los personajes secundarios, pero resultó siendo menos práctico a la hora de manejar interacción entre objetos de la misma clase y de otra. Por esta razón se procedió a crear instancias de cada uno en un arreglo de objetos para cada tipo de personaje.

Se verá al comienzo del archivo *qqCharacter.h* que se ha definido el máximo número de personajes como diez (*MAXGUYS*). La razón de escoger este número no es que se hayan definido diez personajes en total sumando enemigos, maestros y personaje principal, sino que considerando la dimensión del programa este es un número suficiente. Sin embargo se implementó previendo que en futuras actualizaciones, se pueda establecer un número ideal como máximo por escenario, buscando minimizar el consumo de recursos, ya que los personajes son los objetos que consumirían más de estos dada una mejora considerable (nuevas texturas, modelo con mallas, inteligencia artificial más sofisticada).

Otra constante declarada es el número de instancias que definen la animación de los personajes como diez (*MAXANIME*). La ejecución de estas instancias en tiempo real son verdaderamente rápidas y por lo tanto son controladas externamente con una función *delay(t)* definida en *qqBasics.h*. Dependiendo de la velocidad de ataque o de movimiento, la demora puede aumentar o disminuir teniendo en cuenta que la apariencia de la animación debe ser fluida para diez fotogramas específicos.

Seguido se define una enumeración para establecer los posibles estados en los cuales se puede encontrar un personaje en un determinado momento. Estos momentos son quieto, moviendo, y atacando (*QQ_STEADY, QQ_MOVE,* y *QQ_ATTACK* respectivamente). Cada vez que un personaje entra en un estado nuevo, el arreglo que maneja su animación es cambiado para representar su estado correspondiente. A la vez, cada estado repercute diferente sobre su entorno.

Para representar características de los personajes, se han definido cuatro estructuras que establecen las proporciones de tamaño (*size_type*), el tipo de arma (*weapon_type*), el tipo de ataque (*attack_type*) y la animación de las partes de un personaje (*anime_type*):

- *size_type:* almacena el tamaño general del personaje, tamaño de cabeza, torso, brazo, antebrazo, pierna, y pantorrillas.
- *weapon_type:* contiene el daño causado por el impacto del arma con un enemigo, y la longitud física del arma.
- *attack_type:* contiene información de la demora de cada ataque y el momento en que hay impacto con el enemigo en un punto entero entre 1 y 10.
- *anime_type:* contiene instancias de *qqRotation* en vectores de tamaño *MAXANIME* para brazo derecho e izquierdo, antebrazo derecho e

izquierdo, pierna derecha e izquierda, pantorrilla derecha e izquierda y una variable que determina en que momento se encuentra la animación (entre 1 y 10).

Entrando ya en la clase del personaje (*qqCharacter*), se puede ver que hereda de *qqPerimeter, qqRotation* y *qqColor*. Los personajes poseen un perímetro debido a que es necesario para determinar colisiones con otros objetos dentro del escenario. Al igual, como *qqPerimeter* hereda de *qqPoint*, esta herencia ayuda a que el personaje sepa ubicarse dentro del mapa y así facilitar la inteligencia artificial enfocada a búsqueda de camino directo mínimo. La rotación ayuda a mostrar hacia donde esta orientado el personaje, por lo tanto, la rotación de los personajes se realizan únicamente sobre el eje y suponiendo que el personaje no puede moverse hacia arriba o hacia abajo, solo hacia el frente, atrás y los lados.

El color define específicamente la tonalidad de la vestimenta del personaje. El color de piel es declarado después en un objeto de tipo *qqColor* llamado *skinColor*. De manera parecida, el color del arma es declarado.

Se ha logrado una significativa simplificación de los atributos y métodos en la clase *qqCharacter* con respecto a versiones anteriores de MathEW. Esto se ha logrado separando funcionalidad de las formas básicas definidas en *qqBasicGraphics.h* y procedimientos generales creados para clases más generales como *qqCharacterSet* y *qqWorld*. Por lo tanto, las propiedades diseñadas para *qqCharacter* son solamente las básicas (tamaño, tipo de arma, color de piel, dirección o orientación, animación, tipo de ataque, estado) y otras características agregadas y que terminan siendo parte del juego más no del personaje como velocidad y vida. Para un proyecto terminado de MathEW, se implementaría un diseño más complejo donde el personaje contemplaría elementos intrínsecos y externos como energía (Maná), experiencia, pociones, hechizos, etc.

Tabla 3. Descripción de la clase qqCharacter.

Atríbuto now	Hay una variable *now* declarada como flotante que funciona para determinar lapsos entre intervenciones del usuario. Esta variable funciona junto con la función *currentNow* definida en *qqBasics.cpp* y que se encarga de devolver el momento actual en segundos.
Constructor qqCharacter	El constructor para *qqCharacter* recibe como parámetros el tamaño total del personaje, la posición en coordenadas x, y, z, la orientación inicial, los valores RGB, la velocidad, vida inicial y momento de impacto durante un ataque. El radio del personaje es posteriormente asignado como el tamaño total dividido en 3,47. El valor 3,47 fue escogido por prueba y error para dejar un rango suficiente más allá de π y encontrar así siempre un radio que evite la superposición de los

	objetos. Dada una altura determinada. Las proporciones para las partes del personaje son las siguientes: ¼ para la cabeza, ¼ para el torso, 1/6 para los brazos, 1/6 para el antebrazo, ¼ para las piernas y ¼ para las pantorrillas. El tamaño del arma es la mitad de la altura total y su daño es siempre 1.
Métodos incLife(), KillMe(), incMoment(), getSpeed(), getLife(), getDamage(), getAanimeMoment(), getAttackMoment() getNow()	Entre los métodos que constituyen *qqCharacter*, los más básicos y sencillos son *incLife(dif)* que incrementa la vida un valor determinado. En este caso, para el prototipo del juego se usa únicamente para disminuir la vida y por lo tanto revisa si la vida es igual o menor a 0, en este caso llama a *KillMe()* con el fin de desaparecer el personaje del escenario. En caso de ser el personaje principal, el juego termina y se pasa a *game over*. Otro método similar es *incMoment()* que incrementa el momento en que se encuentra el personaje. El momento es siempre de 1 a 10 y funciona en conjunto con *anime* para ejecutar la animación de los personajes. Dependiendo del momento en que se encuentre el personaje, la rotacion de sus partes cambia. Por otro lado, están las funciones básicas para devolver velocidad (*getSpeed()*), vida (*getLife()*), daño infligido por el arma (*getDamage()*), momento de animación (*getAnimeMoment()*), momento de impacto de ataque (*getAttackMoment()*) y variable *now* (*getNow()*).
Método intersect()	Por otro lado, encontramos métodos más complejos, entre ellos un método para detectar colisión con otros objetos llamado *intersect(x,y,z,r,dir*) que determina si un objeto personaje intercepta con otro objeto en la posición *x,y,* y *z* y radio *r* en caso de moverse en dirección *dir*. El algoritmo para determinar esto se basa en condiciones que empiezan con casos especiales para movimientos norte/sur o este/oeste y luego una condición para casos generales para cada dirección. Está función devuelve 0 (Falso) en caso de no haber intersección y 1 (Verdadero) en caso de haberla.
Método linkRotationAngle()	El método *linkRotationAngle(dir)* hace que el personaje rote un ángulo determinado dada una dirección *dir*. Similarmente, *loadAnime(tState)* enlaza un estado *tState* con unos valores de rotación para cada parte del personaje. Este método funciona en conjunto con el método *swapState(tState)* que se encarga de cambiar el estado de un personaje en cualquier momento de juego. En esta función se asigna un tiempo a *now* en caso de que el estado al que se cambie sea de movimiento o de ataque. Si el nuevo estado es de pararse, *now* no es actualizado para luego detectar que el personaje a

	dejado de moverse en la función *worldTurn(time)* de la clase *qqWorld*.

Por último, para finalizar el funcionamiento orientado a objetos de la aplicación, se explicará la clase que abstrae el conjunto de personajes y que ha sido el núcleo que ha permitido la evolución en el diseño de la programación. Inicialmente, en la primera de las versiones del prototipo MathEW, todo el juego y toda la programación se encontraba dentro de *qqCharacterSet* (solo que para ese entonces se llamaba solo *characterSet*). Puede deducirse por lo tanto, que en consideración de funcionalidad e infraestructura todo se encuentra soportado en esta clase.

La clase *qqCharacterSet* no hereda de ninguna otra clase, pero contiene dentro todo tipo de objetos que representan algo físico en el espacio virtual. Estos objetos son los personajes (*guy[]*), el computador (*com*), y los árboles (*tree[]*). Claro está que posteriormente podrán ser añadidos todo tipo de objetos que luego darán mayor valor al juego. Mientras tanto, estos son los tipos de objetos, y en compañía con estos se encuentran otras variables que controlan su cantidad en el escenario. Las variables *enemies* y *masters* dicen cuantos enemigos y cuantos maestros hay respectivamente. El arreglo *kills[]* muestra que personajes han sido eliminados del escenario para no dibujarlos en rutinas *paint()*. Junto está la variable *numKills* para saber cuantos personajes están en *kills[]*, y finalmente otra variable *numTrees* para saber cuantos árboles hay en el escenario.

Ya que *qqCharacterSet* es una clase que maneja la clase *qqCharacter*, varios de sus métodos de obtención de datos llaman a otras rutinas que se encuentran dentro de *qqCharacter* para obtener información de uno de sus miembros. Estás funciones son *getCharLife(who)* que obtiene la vida de *guy[who]*, *getCharSpeed(who)* que obtiene la velocidad de *guy[who]*, *getCharNow(who)* que obtiene la variable *now* de *guy[who]*. Por otro lado hay dos métodos para obtener el número de enemigos y maestros (*getEnemies()* y *getMasters()* respectivamente).

Al igual que se presentan métodos para obtener de un personaje específico, también hay métodos para cambiar un atributo en alguno de los personajes. Estas funciones son *setCharAngle(who, ang)* que cambia el ángulo de *guy[who]* y *setCharState(who, s)* que cambia el estado de *guy[who]*.

Entre los métodos funcionales de *qqCharacterSet* está *actToward(who, dir, speed)* que hace que un personaje se mueva en torno a una dirección con una velocidad determinada. *AllowMove(who, dir)* que funciona junto con la función de inteligencia artificial para determinar si un movimiento es permitido para un personaje dado. Si hay colisión con algo, la función devuelve con un entero especificando la posición del personaje colisionado en *guy[]*, de lo contrario, si no hay colisión, devuelve -1. *attackAtoB(src, dest)* realiza las operaciones necesarias cuando un ataque es hecho de una fuente *A* a un destino *B*.

También revisa si con un ataque el personaje es eliminado, para posteriormente ponerlo en el arreglo *kills[]* o llegar a *game over* en caso de que el destino sea el personaje principal. *IncMomentSet()* incrementa el momento utilizando el método *incMoment()* en cada objeto *guy[]* con el fin de actualizar cada personaje en cada turno del mundo (*worldTurn(time)*). *IsKilled(who)* revisa si un personaje ha sido eliminado.

La inteligencia artificial del juego está programada en el método *aiMove()* de la clase *qqCharacterSet*. Esta inteligencia es bastante directa y busca que los enemigos busquen al personaje principal en todo momento. Siempre se encuentran ciertas condiciones en sus movimientos como revisar que no estén chocando con nada y que no hagan movimientos en "diagonal" (evitar cambios súbitos y momentáneos en la dirección del personaje con el fin de mejorar aspectos visuales de la animación).

Por último, el método *paint()* dibuja todos los personajes incluyendo enemigos, maestros y personaje principal dando excepción con aquellos que ya han sido eliminados utilizando la función *isKilled(who)*. También dibuja los otros objetos secundarios como el computador y los árboles.

4.4 MAIN

La parte principal de programa o *main(argc, argv)* de la aplicación en realidad no tiene mucha funcionalidad excepto de iniciar todos los procesos de *OpenGL* y *GLUT*. Anterior a esto, se define un objeto *world* de tipo *qqWorld* para uso en todo el programa. Es por esto que este se ha dejado de último en la presente explicación detallada del código.

Entre las características generales de la configuración de *OpenGL*, se tienen especificaciones de utilización de canales RGB, medidores de profundidad, definición de pantalla completa y declaración de las funciones de dibujo, animación, teclado, y Mouse. También hay una pequeña implementación de la librería *GLEW* (OpenGL Extensión Wrangler) que no tiene funcionalidad en el momento pero está programada para funcionar con reprogramación de *shaders* en alguna versión futura. Los *shaders* son funciones que manejan el sombreado e iluminación en *OpenGL* y tiene su propio lenguaje estándar. Este lenguaje de *shaders* ayuda a que la luz no afecte solo los vértices como está especificado en *OpenGL* sino que también impacte en cada píxel o fragmento de una superficie.

En la función *main* también se hace llamado a la función *init* donde se inician aspectos más específicos del escenario que va a ser manejado. Entre estos aspectos se encuentra la luz ambiente, el color de la neblina, el color de borrado de pantalla, se activan los materiales por color, la luz, la neblina y sus características, y a la vez se encuentra programada una segunda luz aunque no ha sido activada debido a la ineficiencia de los *shaders* predeterminados.

5. CONCLUSIONES

En los aspectos pedagógicos, se han identificado los elementos que hacen a un juego interesante y divertido. Se ha establecido claramente la relación de estos elementos con el juego propuesto y las virtudes que tienen los juegos de rol en incorporarlos.

La investigación de las teorias pertinentes en esta materia, así como el estado del arte en los intentos por involucrar las TIC en la educación, han permitido adoptar una filosofía constructivista para el diseño del proyecto. Esto ha fortalecido la visión con que el proyecto se concibió.

En materia tecnológica, el proyecto se vio beneficiado por la elección de una solución que contempla no solo las ventajas técnicas para la implementación, sino que considera la importancia que el proyecto tiene como solución a una necesidad humana y enmarcada en el contexto de un país en desarrollo. De esta manera el proyecto sale de su dimensión técnica y deja de limitarse a ser una implementación o adaptación tecnológica más, para constituir una propuesta viable para desarrollar soluciones a un problema clave para generar las competencias requeridas por la nación para garantizar un progreso sostenible.

No obstante estas consideraciones, es claro que las alternativas contempladas en ningún momento hicieron caso omiso a tecnologías que estuvieran al tanto de los últimos adelantos en desarrollo de soluciones de hardware para el procesamiento de gráficos y sonido. Consecuencia de esto, las decisiones tomadas han incorporado soluciones que se hacen escalables a la magnitud que se quiera dar al proyecto, sin prestar limitaciones adicionales a las que presupuestalmente se puedan presuponer.

6. RECOMENDACIONES

El desarrollo de una versión comercial del proyecto debería, además de mantener el enfoque pedagógico y los elementos motivacionales del juego, contemplar las siguientes posibilidades:

- Uso de una clase diseñada para dialogos y desafios, que se encuentre enlazada a un *script* para facilitar la escalabilidad en las interacciones entre los personajes y los retos presentados en el transcurso del juego

- Programación de shaders para mejorar la iluminación y sombreado, y por consiguiente, el atráctivo visual del juego

- Uso de Stencils para sombras, reflejos y otros efectos visuales.

- Añadir propiedades, elementos, capacidades y habilidades a los personajes, haciendolos personalizables a las preferencias del jugador. Esta posibilidad debería permitir la elección de sexo, dejando que Yo sea mujer.

- Habilitar el uso del mouse, como dispositivo de entrada para el desarrollo de algunos desafios.

BIBLIOGRAFIA

ASSOCIATION FOR COMPUTING MACHINERY. The ACM digital library [Biblioteca en línea]. ACM. Disponible en Internet <http:/portal.acm.org/di.cfm>

DEUBEL, Patricia. Game on! [Artículo en línea]. Chatsworth, California : 1105 Media, 2006. Disponible en Internet < http://www.thejournal.com/articles/17788_3>

FARLEX. The Free Dictionary [Diccionario en línea]. Huntingdon Valley, Philadelphia : Farlex. Disponible en Internet <http://www.thefreedictionary.com/>

KAFAI, Yasmin B. The educational potential of electronic games : From games-to-teach to games-to-learn [Artículo en línea]. Chicago : University of Chicago, 2001. Disponible en Internet < http://culturalpolicy.uchicago.edu/conf2001/papers/kafai.html>

LEPPER, Mark R. and GREENE, David. The hidden costs of reward : New perspectives on the psychology of human motivation. Hillsdale, New Jersey : Lawrence Erlbaum, 1979. 276 p.

LUMSDEN, Linda. Student motivation to learn [Documento en línea]. Eugene, Oregon : Educational resources information center, 1994. Disponible en Internet: < http://www.kidsource.com/kidsource/content2/Student_Motivatation.html#credits >

MALONE, Thomas W. What makes things fun to learn? : Heuristics for designing instructional computer games. En : ACM SIGSMALL SYMPOSIUM (3° : 1980 : Palo Alto) Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems. Palo Alto : ACM, 1980. p. 162-169.

PAPERT, Seymour. Constructivism. Westport, Connecticut : Ablex Publishing, 1991. 518 p.

ROY, Promit. Direct3D vs OpenGL : Which API to use when, where and why [Artículo en línea]. GameDev.Net. Disponible en Internet < http://www.gamedev.net/reference/articles/article1775.asp>

SLIWKA, Dirk. On the hidden costs of incentive schemes. En : WORKSHOP EMPIRICAL PERSONNEL ECONOMICS (1° : 2004 : Bonn) IZA conferences program. Bonn : IZA, 2004. 38 p.

WIKIMEDIA FOUNDATION. Wikipedia : The free encyclopedia [Enciclopedia en línea]. Wikimedia Foundation. Disponible en Internet : http://www.wikipedia.org/wiki/Main_Page

ANEXO 1. MANUAL DEL USUARIO

MathEW es un prototipo de un juego educativo enfocado en el área de las Matemáticas. Se llama MathEW como abreviación a su nombre completo Mathematics Entertainment World y como dice su nombre, su función es enseñar matemáticas de una manera divertida a través de un mundo fantástico creado para cumplir su propósito de educar jugando.

En su forma de prototipo, MathEW es un juego no comercial y se recomienda que durante su uso se haga apreciación de las tecnologías 3D utilizadas y del prospecto que puede ser para el desarrollo de un juego educativo que efectivamente enseñe todas las bases de las Matemáticas para los estudiantes de Bachillerato. Por lo tanto, este juego no esta diseñado aún para ser herramienta de enseñanza para los estudiantes de Bachillerato como tal, sino como herramienta para demostrar las capacidades del diseño de juegos a los estudiantes de las distintas ciencias computacionales.

Los requisitos para ejecutar MathEW es de tener instalado los controladores de OpenGL 1.1 para el manejo de gráficos 3D y OpenAL para el manejo de sonido igualmente en 3D. Además, esta versión corre únicamente en plataformas Macintosh y ha sido probado únicamente en un MacOS X Tiger. Aunque el código sea multiplataforma, el proyecto fue diseñado en un Mac y el ejecutable compilado es por lo tanto solo para Mac. Para su uso en Windows o Linux, se debe configurar el editor propio de la plataforma para reconocer las librerías de OpenGL, GLUT y OpenAL, de otra manera el código no compilará apropiadamente.

Ya con el ejecutable, la manera de correr el juego es directamente haciendo doble click en la aplicación. Este prototipo no posee menú aún así que el programa iniciará inmediatamente en modo de juego.

Los controles para jugar son los siguientes:
8 – Mover hacia delante, atacar enemigo que se encuentre delante, o dialogar con personaje que se encuentre delante.
4 – Mover hacia la derecha, atacar enemigo que se encuentre a la derecha, o dialogar con personaje que se encuentre a la derecha.
6 – Mover hacia la izquierda, atacar enemigo que se encuentre a la izquierda, o dialogar con personaje que se encuentre a la izquierda.
2 – Mover hacia atrás, atacar enemigo que se encuentre atrás, o dialogar con personaje que se encuentre atrás.
Escape – Termina la aplicación.