Introducción a la Computación Gráfica

Obligatorio 1
Curso: 2015

Leandro Burgos 4.582.911-8
Liber Azambuya 4.602.288-6
Gonzalo Torterolo 4.868.387-6

Índice

Introducción

Definición del problema

El problema del laboratorio consiste en desarrollar una variante en 3D del juego Missile Command, desarrollado por Atari y publicado el año 1980. Es un juego de estilo shooter cuyo objetivo principal es defender las ciudades de los misiles que caen desde el cielo.

Análisis del problema

Puntos importantes a analizar:

- Escenario del juego (balas, misiles, edificios y universo).
- Colisión entre balas y misiles, colisión entre misiles y edificios
- Velocidades y direcciones de las balas y misiles.
- Aspectos de jugabilidad (vida del jugador, puntaje, niveles, etc).
- Cargado de modelos 3ds.
- Cargado y renderizado de texturas.
- Movimiento de la cámara con el teclado.
- Movimiento del puntero de la cámara con el mouse.
- Manejo de eventos (disparo de balas, salir del juego, pausar el juego, etc.).
- HUD del juego.

Diseño de la solución

La solución se basa en un conjunto de clases nombradas luego en la Arquitectura.

Las clases màs importantes serán la clase Figura, de la cual heredan el resto de las clases relacionadas con los objetos que interaccionan en el juego (Misil, Bala, etc.), para modelar los movimientos tanto de las figuras como de la cámara, utilizaremos la clase Vector, que se usarà tanto para las posiciones de dichos objetos así como para las velocidades de los mismos. Tendremos una clase Game para modelar todos los aspectos relacionados al juego y sus variables (niveles, puntajes, vida del jugador, cantidades de los objetos, estados del juego en pausa, etc.) y otra clase para modelar el Menú.

Mediante estas clases se representará los objetos involucrados en el juego y lo que se verá en cada instante del mismo será una representación visual de los estados actuales de cada uno de estos objetos. Es decir, las balas y los misiles tendrán una posición, una dirección y una velocidad asociadas, la cantidad y velocidad de los misiles así como la cantidad de edificios estarán relacionados con el estado actual de la clase Level, que llevará un estado del avance del juego.

Así mismo, la cámara también tendrá un estado asociado y esto se verá reflejado en la posición y perspectiva del jugador.

El universo del juego estará compuesto por un suelo en donde estarán los edificios, y una semi esfera en donde se ubicarán el resto de los objetos, cada uno de ellos con una textura asociada.

A su vez, en todo momento se visualizará el estado actual del juego (nivel, puntaje, vida restante, cantidad de balas restantes) en el HUD, implementado mediante el uso de fuentes de texto y renderizados mediante una proyección ortogonal.

El juego comenzará con el jugador ubicado en el nivel 1 con una cantidad de vida inicial, un score inicial en 0 (cero), una cantidad de edificios iniciales y una cantidad de misiles por caer. Cada una de estas cantidades estarán configuradas en el archivo xml y serán individuales para cada nivel.

Los misiles caerán del cielo cada determinado tiempo configurable, y habrán en cada instante una máxima cantidad de misiles simultáneos en el cielo, cantidad configurable en el XML para cada nivel.
El jugador a su vez comenzará cada nivel con una cantidad de balas limitadas y configurables en el XML, y con dicha

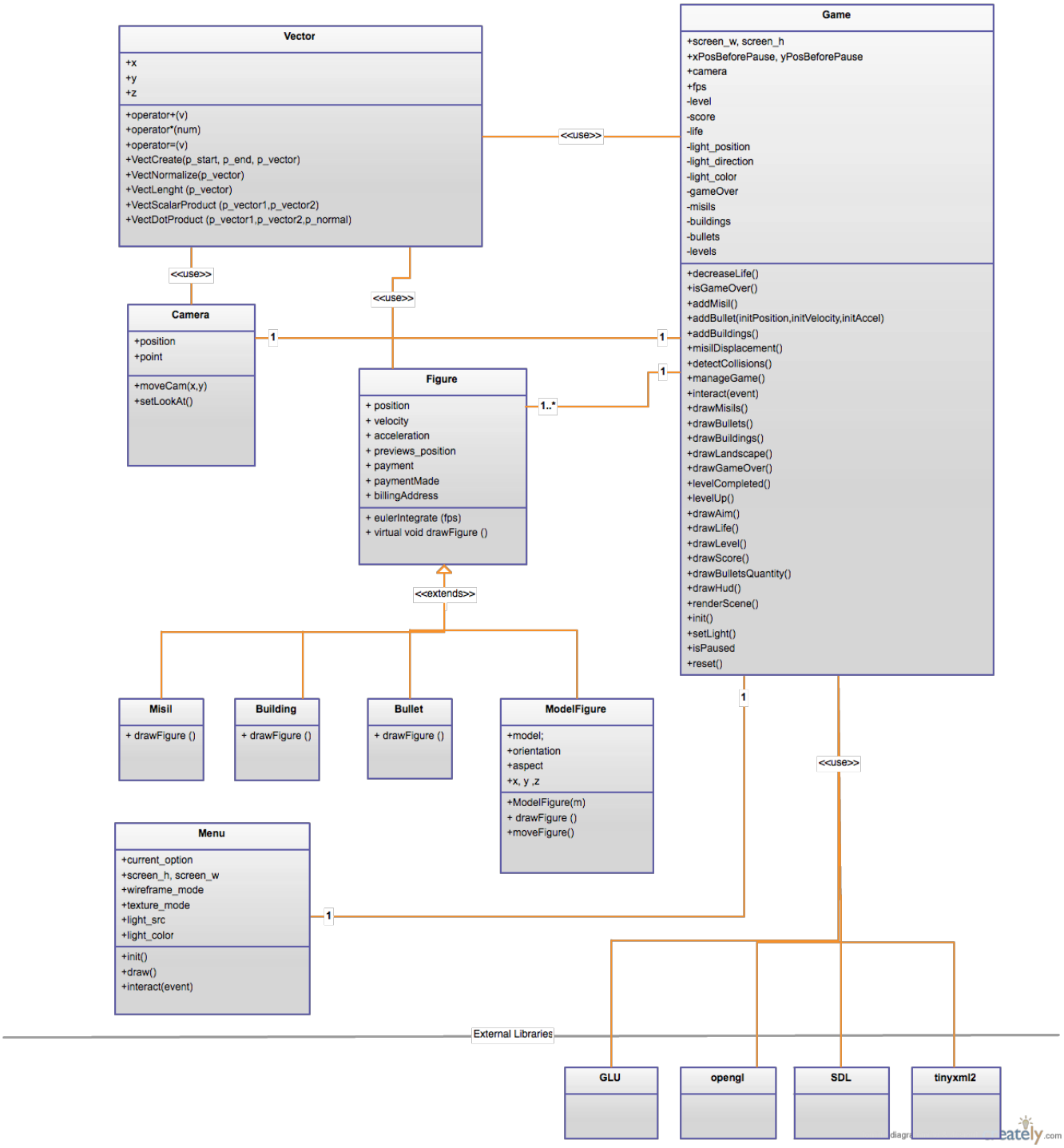
cantidad de balas deberá destruir todos los misiles correspondientes a cada nivel.
Una vez destruida un misil, se incrementarà el score del jugador en 120 puntos.

Cada misil irá dirigido hacia uno de los edificios restantes al momento de lanzarse, y el jugador deberá destruirlo con una bala. En caso de no ser destruido, el misil colisionará con el edificio y ambos desaparecerán de la escena, disminuyendo en 1 (uno) la vida restante del jugador.
Si la cantidad de edificios destruidos es tal que el jugador se queda sin vida restante, perderá el juego y verá en pantalla el cartel de “GAME OVER”.

En caso de que el jugador destruya todos los misiles, o destruya la cantidad suficiente como para que la cantidad de edificios destruidos no alcancen a quitarle toda la vida, avanzará de nivel.
Cuando el jugador avanza de nivel se reiniciarán la cantidad de balas restantes, la vida del jugador y la cantidad de misiles a caer, así como su velocidad.
También se le sumarán puntos adicionales al score del jugador dependiendo del estado final del nivel. Por cada bala sobrante se aumentarán 150 puntos, y por cada punto de vida sobrante se aumentarán 300 puntos.

El juego consta de 5 niveles configurados en el XML, pero su cantidad puede ser cambiada en cualquier momento, así como las variables correspondientes a cada nivel.
Una vez que el jugador destruye todos los misiles correspondientes al último nivel, el juego terminará y se mostrará en pantalla el score final del jugador con un cartel de “VICTORY”.

Arquitectura



Implementación

Descripción del entorno utilizado

El entorno de desarrollo consistió en:.

- IDE CodeBlocks 13.12
- GNU GCC MinWG Compiler 4.8.1
- Sistema operativo Windows 7 y 8
- Versionado usando git.
- Gimp y Blender: Varias veces fué necesario importar, exportar o realizar alguna modificación de los recursos como imagenes o archivos 3ds, para esto se usaron estas herramientas.

Algoritmos y técnicas gráficas implementadas

No se implementó ninguna técnica gráfica específica para realizar la tarea.

Librerías utilizadas

SDL (Simple DirectMedia Library):

Es una librería multimedia multiplataforma que ofrece funcionalidad para manejo de ventanas, lectura de teclado, reproducción de sonido, etc.

Versión utilizada: 1.2

SDL_ttf:

Es una extensión de SDL que permite renderizar texto, solo se uso una mínima parte de esta librería, para poder generar textos dinámicos fácilmente. Se encarga de 2 cosas principalmente, de leer archivos (TrueTypeFont) ttf y de generar texto con una fuente dada.

OpenGL

Es una interfaz para la generación de gráficos (Graphics rendering API), permite generar imágenes de alta calidad a partir de primitivas geométricas. Es independiente del sistema de ventanas y del sistema operativo.
Algunas de las cosas que provee OpenGL:

- Un conjunto de funciones que controlan la configuración del sistema de dibujado.
- Un sistema de proyección que permite,especificar objetos en 3 dimensiones y llevarlos a coordenadas de pantalla.
- Un conjunto de funciones para realizar transformaciones geométricas que permiten posicionar los objetos en el espacio.

GLU (OpenGL Utility Library)

Es un conjunto de funciones que simplifican el uso de OpenGL para especificar la visual, construcciones simplificadas de superficies cuadráticas (entre otras cosas).

Código desarrollado por terceros

Enumeración y descripción de las porciones de código que se utilizaron y no fueron desarrolladas por los estudiantes.

- Tinyxml2: librería para lectura y escritura de archivos xml
- ModelType: librería para cargado de objetos 3ds. El renderizado fué implementado aparte.
- LoadBMP: Es un snippet de código que permite leer la metadata de un bmp, conocer sus dimensiones y cargar la imagen en una textura de openGL.

Estados de la aplicación

Enumeración de los estados de la aplicación y cómo el usuario puede interaccionar con la misma.

- Estado inicial: presentación
    - El estado inicial consiste en el jugador ubicado en una posición inicial predeterminada y con la cámara hacia los edificios que debe defender. Un conjunto de edificios cuya cantidad dependerá de las configuraciones del XML asociado. Un puntaje inicializado en 0 y un nivel inicial, que tendrá asociados la cantidad de edificios, velocidad de los misiles, etc. Una cantidad de vida restante para el jugador, que dependerá de la cantidad de edificios inicial.
- Estado de juego
    - Estado jugando: consiste en un estado de evolución constante en donde contínuamente caerán misiles del cielo, y una vez destruidos todos los misiles de cada nivel se accederá al nivel siguiente de manera instantánea, reapareciendo los edificios e iniciando nuevamente la caída de los misiles.
    - Estado pausado: el juego almacenará su estado actual y todo se mantendrá así hasta salir de la pausa, ningún objeto se moverá y la cámara permanecerá quieta.
- Estado de opciones
    - El estado de opciones será un estado de pausado del juego, donde pueden modificarse distintos aspectos relacionados con el mismo, estos aspectos son:
        - La velocidad del juego
        - Los colores del juego y sus elementos
        - La dirección de la iluminación del juego
        - Habilitar o deshabilitar texturas
        - Habilitar o deshabilitar wireframes

Estructuras de datos

Las estructuras principales son:

Misil: Estructura de datos que representa un misil.
Bullet: Estructura de datos que representa una bala.
Building: Estructura de datos que representa un edificio.

Para representar un escenario del juego se tienen tres listas en la clase Game donde una lista representa los misiles que hay actualmente en el juego, otra lista representa las balas que le quedan al jugador y otra que representa los edificios que todavía no fueron derribados.

Velocidad de simulación

En base a que se tiene definido un motor físico, y en base a esto se especifican las velocidades de cada figura del juego, utilizamos una simple ecuación para conocer la cantidad a mover de cada objeto en cada frame a modo de dar una sensación de movimiento realista, independientemente de los FPS que se quieran usar. Nos referimos como FPS a los frames (renderizaciones) por segundo. El parámetro FPS es fácilmente configurable. Un FPS menor implica también una aplicación menos exigente para el procesador.

Uso de OpenGL

Especificación del formato que describe el archivo de escena utilizado.

Manejo del estado de OpenGL

Explicación de cómo se maneja el cambio de estado de OpenGL (settings), cuales son los estados que se trabajan y cómo la forma de trabajo elegida incide en la estructura de la aplicación.

Iluminación, color y materiales

En esta sección se describen cómo se utilizaron las características que brinda OpenGL relacionadas con la iluminación, color y materiales.

La iluminación implementada consiste una luz posicional que utilizará 4 puntos laterales, un punto superior y uno inferior. A partir de esos puntos -que son seleccionables dinámicamente- se emitirá una luz de tipo GL_LIGHT0, desde una de las posiciones seleccionadas. Estas posiciones serán una combinación de dichos 4 puntos laterales (izquierda, derecha, adelante y atrás) y el punto superior o inferior.

Los comandos a utilizar serán:

glEnable(GL_LIGHTING)
glEnable(GL_LIGHT0)
glShadeModel(GL_SMOOTH)
glColorMaterial (GL_FRONT_AND_BACK, GL_EMISSION)
glLightfv(GL_LIGHT0, GL_POSITION, light_position)

Para los colores del juego se implementó una solución similar, con 4 combinaciones de colores que se utilizarán como entrada para los comandos de OpenGL que se encargan de asignar valores a los parámetros materiales del modelo de iluminación.

Estos parámetros serán una combinación de la reflectancia ambiental de los materiales (GL_AMBIENT) y la reflectancia difusa (GL_DIFFUSE), donde en cada una de las 4 combinaciones se le asignará a cada una de las reflectancias valores distintos RGBA.

Display List

No se utilizaron display lists en la solución planteada.

Vertex arrays

No se utilizaron vertex arrays en la solución planteada.

Texturas

La utilización de texturas sobre sólidos no presentó gran dificultad, el mayor trabajo consistió en la carga de la misma desde un archivo. En nuestro caso solo aceptamos archivos bmp, un formato sin compresión, lo que generó la necesidad de comprimir las imágenes para mantenerse dentro del límite de tamaño o disminuir la calidad de las imágenes. Optamos por disminuir la calidad de las imágenes, asumiendo que no presenta mayores retos técnicos. Para futuras versiones se sugiere comprimir (usando zlib o similares). Del mismo modo, no hizo falta usar grandes niveles de MipMapping, pues tampoco la calidad de las imágenes dejaban que esto fuera perceptible, sin embargo, dado el diseño ordenado de la aplicación no sería difícil aumentar el nivel de MipMapping. El mapeo de texturas sobre sólidos también incluyó mapear texturas sobre los archivos 3ds.

También se utilizaron texturas para imprimir textos, esto de manera indirecta mediante SDL_ttf, que escribe sobre una textura de openGL.

Modelos 3D

Los modelos cargados desde 3ds tienen la restricción de estar compuestos solo de triángulos. Los modelos consisten por lo tanto de listas de triángulos dispuestos en el espacio. A cada uno de estos triángulos también se le pueden agregar una normal y un mapeo de un textura (que debe ser elegida adecuadamente y que venian con los archivos 3ds). Los pormenores

de la carga son manejados por un librería externa. La renderización se facilitó en base a que fueran todos objetos uniformes, pero se debió normalizar el posicionamiento del objeto para facilitar su manejo. Cada modelo 3D tiene definidos límites que pueden ser utilizados también para mover ó intersectar con otros modelos, una orientación y una serie de factores de escalado.

Cámara

La aplicación desarrollada pertenece al género de videojuegos conocido como FPS(First Person Shooter), por lo tanto el movimiento de la cámara juega un rol importante ya que representa el movimiento del jugador dentro de la misma.

Para la representación de la misma utilizamos dos puntos en el espacio. Uno que indica la posición actual de la cámara y el otro que indica el punto al cual está mirando la misma. Esto lo realizamos así debido a que la librería "glu" provee una operación para su manejo (gluLookAt(...)) la cual recibe como parámetro ambos puntos mencionados anteriormente.

La cámara está compuesta por dos tipos de movimientos, uno que solamente realiza una rotación del punto al cual está mirando y el otro que realiza una traslación de la cámara a una nueva posición.

1. **Rotación mira:** Para realizarlo utilizamos los eventos generados por el movimiento del mouse de forma que cada vez que se generen dichos eventos, calculamos el nuevo punto al cual está mirando la cámara en base a la nueva posición del puntero del mouse.
2. **Movimiento cámara:** El movimiento de la misma lo realizamos de forma similar a como se explican los movimientos en la sección correspondiente. Para este caso, el vector velocidad se calcula dependiendo del evento que generó el movimiento(flechas de navegación) .

   En caso que se quiera mover la cámara hacia adelante, calculamos el vector que se encuentra entre el punto al cual está mirando la cámara actualmente y la posición de la misma. En caso que el movimiento deseado sea hacia atrás, se multiplica dicho vector de velocidad por el valor -1.

   En caso que se quiera mover la cámara hacia la derecha, se obtiene el vector perpendicular al mencionado anteriormente con sentido hacia la derecha. En caso que se quiera mover la cámara hacia la izquierda, se obtiene el vector perpendicular pero con sentido hacia la izquierda.

   Ejemplo vectores: en el eje y siempre se tiene el valor cero dado que no se desea que la cámara se mueva en dicha dirección.

   v1 = (x,0,z) vector que va desde el punto de posición de la cámara al punto de vista de la cámara. Utilizado para el movimiento hacia adelante

   v2 = (-x,0,-z) Vector con sentido opuesto a v1, utilizado para el movimiento hacia atrás.

   v3 = (z,0,-x) Vector perpendicular utilizado para movimiento hacia la derecha.

   v4 = (-z,0,x) Vector perpendicular utilizado para movimiento hacia la izquierda.

Restricciones de movimiento:

No se desea que la cámara se mueva entre los edificios ya que esto afecta la jugabilidad. Para restringir esto se generó un polígono que encierra a todos los edificios y en cada nuevo movimiento de la cámara se controla que la posición de la misma no caiga dentro del polígono, de ser así no se realiza el movimiento manteniéndola quieta.
Tampoco se permitirá al jugador moverse más allá de los límites del juego, que incluyen el suelo y la media esfera que conforma el cielo.

HUD

Para dibujar el HUD se usa proyección ortogonal. El mismo consiste en varios textos. Estos dibujos serán una combinación de letras y números para representar el estado actual del juego (vida del jugador, cantidad de balas restantes, puntuación actual y nivel actual).

El HUD también contará con una mira de disparo, que consiste en una circunferencia y 2 semirectas perpendiculares de color azul, ubicados en el centro de la pantalla en todo momento.

Dicho HUD será incluido dentro de los elementos a renderizar en cada nuevo frame, por lo que su actualización será dinámica y estará relacionada directamente con el valor de las variables que lo implican.

Desarrollo del obligatorio

Cargado de información

settings.xml: Archivo de configuración que contiene la definición de los niveles utilizados en la aplicación.

ejemplo de archivo con dos niveles:

```
<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">

<GAME>
    <LEVEL>
        <LEVEL_NUMBER>1</LEVEL_NUMBER>
        <MISSILE_SPEED>25</MISSILE_SPEED>
        <CANT_MISSILES>10</CANT_MISSILES>
        <CANT_SIMULTANEOUS_MISSILES>3</CANT_SIMULTANEOUS_MISSILES>
        <CANT_BUILDINGS>10</CANT_BUILDINGS>
        <CANT_BULLETS>30</CANT_BULLETS>
        <LIFE>5</LIFE>
    </LEVEL>

    <LEVEL>
        <LEVEL_NUMBER>2</LEVEL_NUMBER>
        <MISSILE_SPEED>20</MISSILE_SPEED>
        <CANT_MISSILES>20</CANT_MISSILES>
        <CANT_SIMULTANEOUS_MISSILES>5</CANT_SIMULTANEOUS_MISSILES>
        <CANT_BUILDINGS>10</CANT_BUILDINGS>
        <CANT_BULLETS>60</CANT_BULLETS>
        <LIFE>5</LIFE>
    </LEVEL>
</GAME>
```

Donde:

1. LEVEL_NUMBER: Número de nivel que se está definiendo.
2. MISSILE_SPEED: Velocidad de los misiles.
3. CANT_MISSILES: Cantidad de misiles que contiene el nivel.
4. CANT_SIMULTANEOUS_MISSILES: Cantidad de misiles que puede haber simultáneamente en el cielo.
5. CANT_BUILDINGS: Cantidad de edificios.
6. CANT_BULLETS: Cantidad de balas.
7. LIFE: Cantidad de vida del jugador.

Movimiento

Todas las figuras que realizan algún movimiento contienen tres vectores indicando su posición, velocidad y aceleración. Al crear la figura, estos vectores son cargados con el valor inicial correspondiente y en cada frame se calcula la nueva posición de la figura aplicando Euler de la siguiente forma:

nueva_velocidad = velocidad_actual + aceleracion * dt

nueva_posicion = posicion_actual + nueva_velocidad * dt

De esta forma, las figuras se van a mover en la misma dirección que indica su vector de velocidad.

Misiles: Para el movimiento de los misiles, les asignamos una posición inicial random en el cielo y elegimos un edificio al azar como destino de dicho misil. Para calcular el vector de velocidad (el cual indica la dirección del movimiento) obtenemos el vector entre ambas posiciones realizando la resta entre el punto que indica la posición del edificio y el punto que indica la posición del misil. En el caso de los misiles utilizamos el vector vacío como aceleración de forma que se van a desplazar a velocidad constante.

Balas: Para el movimiento de las balas, utilizamos como posición inicial la misma posición donde se encuentra la cámara y para calcular el vector de velocidad realizamos la misma técnica que con los misiles pero calculando el vector entre la posición de la cámara y la del punto al cual está mirando la cámara, con esto nos aseguramos que la bala siempre va a dirigirse con dirección al punto que estamos visualizando desde la cámara. Al igual que los misiles no utilizamos aceleración para las balas.

Colisiones

En el caso de que sea necesaria la detección de colisiones en la aplicación, especificar las técnicas utilizadas para lograrlo. Detallar cuales son las decisiones que se toman en el caso de que se detecte una colisión.

Para detectar las colisiones necesarias en el juego (balas y misiles, misiles y edificios), se utilizará solamente el estado actual de las posiciones de cada uno de los objetos implicados. A partir de dichas posiciones, se iterará sobre cada uno de los objetos y para cada objeto se verificará si cumple con alguna de las condiciones de colisión.

Las condiciones de colisión son las siguientes:

- Dado un misil, este podrá colisionar con un edificio si cumple con una condición de cercanía al centro del edificio, que se calculará a partir de las posiciones en ese instante de cada misil con cada uno de los edificios.
- Dado un misil, este podrá colisionar con cada uno de las balas lanzadas por el jugador si cumple con la condición de cercanía con el extremo “posterior” de la bala, que se calculará a partir de la posición actual del misil con cada una de las posiciones actuales de las balas disparadas.

En caso de ocurrir una colisión, simplemente se eliminarán ambos objetos colisionados de cada una de las colecciones que los administran y que son usadas para la administración de las balas, misiles y edificios presentes en cada instante del juego.

Una vez eliminado un objeto de la colección que lo administra, este desaparece de la imagen ya que no será parte del renderizado de objetos del juego.

Settings

Explicar cómo se implementa la interfaz para el cambio de estados de dibujo en la aplicación.

Para los cambios de estado de dibujo de la aplicación (wireframes on/off, textures on/off, etc.) simplemente se utilizaron variables booleanas en la clase Game, modificables desde el menú del juego.
Dependiendo de los valores configurados en el menú, luego se consultan estas variables y en cada frame nuevo a dibujar se aplicarán los comandos necesarios para satisfaces dichas configuraciones (glPolygonMode(GL_FRONT_AND_BACK, GL_LINE), glDisable(GL_TEXTURE_2D), etc.).

Características particulares de la solución

Exponer las características específicas de la aplicación que no se detallaron en las otras secciones.

Una de las soluciones utilizadas en la implementación, que a simple vista parece casi trivial pero que fue una de las más difíciles de encontrar a la hora del diseño del juego, es el problema de direccionar las balas hacia donde apunta la mira en el

instante de disparada, así como también direccionar los misiles hacia alguno de los edificios restantes en el momento de lanzado un misil.

Esto fue solucionado mediante vectores de velocidad, que una vez aplicados a una figura esta se mantendrá y hará avanzar a dicha figura en dirección a ese vector y con la rapidez indicada por el mismo. Para hallar dicho vector, la solución utilizada fue realizar una resta de posiciones (x,y,z). Esto es, dada la posición del jugador y la posición hacia donde apunta la mira, si restamos esta última con la primera, obtenemos un vector que tiene como dirección la recta que va desde un punto hacia el otro, y luego multiplicando dicho vector por un coeficiente podemos aumentar o disminuir su velocidad, para luego asignárselo a la bala y que esta se dirija en dicha dirección con dicha velocidad.

Lo mismo en el caso de lanzar un misil en dirección a un edificio, con una velocidad determinada.

Conclusiones

El universo de OpenGL y SDL es muy amplio y nos permiten realizar variados tipos de aplicaciones. A nuestro entender, consiste en 5 grandes áreas a investigar para el aprovechamiento de dichas librerías. Las mismas son:

Manejo de la cámara: Existen distintos puntos de vista a la hora de elegir el manejo de la cámara de una aplicación. En nuestro trabajo se eligió FPS (First Person Shooter) donde la cámara acompaña el movimiento del jugador.

Cargado de modelos: El cargado de modelos es fundamental para mejorar la estética de un escenario. Los modelos son generados utilizando una herramienta externa y cargados al escenario de nuestra aplicación.

Manejo de luces: Para el renderizado de los objetos, es necesario la utilización de técnicas de iluminación para mejorar la calidad visual de dichos objetos. Esto incluye luces posicionales y direccionales, que generarán sombra en los objetos y permite ver a los objetos en 3 dimensiones y no simplemente en un plano de un solo color.

Movimiento de objetos: Detrás del movimiento de los objetos existe una lógica basada en los conocimientos proporcionados por la física. Fue necesario utilizar los conceptos de vector, velocidad, acelerción, vector normal y también funciones de integración de la velocidad y movimiento.

Colisión entre objetos: Determinar la colisión entre dos objetos está fuertemente relacionado con la forma elegida para el movimiento de los mismos.

En el trabajo presentado, la investigación de los puntos mencionados y el buen manejo de los mismos presentaron la mayor complejidad a la hora de realizar la aplicación.

Teniendo ya definido como manejar estos puntos en nuestra aplicación, resulta simple escalarla agregando objetos en el escenario que la mejoren estéticamente y agreguen valor a la misma. Siguiendo la misma lógica y procedimientos, sería fácil incluir más objetos 3ds, mejores texturas, niveles, iluminación, etc., que mejoren la calidad y jugabilidad del juego.
Para esto creemos que fue importante la modularización utilizada al momento de implementar las clases y objetos que conforman la aplicación, ya que están débilmente acoplados y los cambios en una clase casi no impactan en el resto, así como también se hace simple la inclusión de nuevas clases y objetos a la física del juego. Creemos que la arquitectura utilizada le brinda solidez y robustez a la aplicación y es por eso que se facilita el escalado de la misma.

Trabajo futuro

Mejorar las texturas y los modelos utilizados para la aplicación: Las texturas y modelos seleccionados para la construcción de la aplicación no son los óptimos, esto debido a que se tiene un requerimiento no funcional para la entrega que está relacionado al peso de la aplicación(no debe exceder los 10 MB).

Sonido: Agregar sonido a los disparos, colisiones, pasos del jugador y sonido ambiental sería de gran aporte al juego.

Escenario: Mejorar la estructura del escenario, de manera que también permita mejorar las texturas que lo conforman, ya que hoy en día consta de una media esfera para modelar el cielo y una textura única en él. Podría mejorarse dicha estructura para utilizar más de una textura, como podrían ser un cielo con montañas debajo.

Jugabilidad: dado el estilo de juego que elegimos (FPS), pensamos que podría agregarle valor a la jugabilidad el hecho de sacarle provecho al movimiento por el suelo del jugador. Esto podría lograrse si los misiles también se dispararan desde una altura similar a los edificios y el jugador tuviera que desplazarse para eliminarlos.

Menú: debido a la falta de tiempo en algunos aspectos del juego que llevaron más tiempo del supuesto, no se le dedicó demasiado esfuerzo a la estética y configuración del menú. Creemos que se podría mejorar todo lo relacionado al mismo.

Misiles: se probó el uso de un modelo 3ds junto con sus texturas para implementar los misiles, pero no hubo tiempo suficiente para realizar las debidas pruebas sobre el mismo por lo que no fue incluido a la solución final.

Compresión de recursos: A medida que los recursos y la estética se eleve, surgirá la necesidad de bajar el peso de os recursos, para eso se sugiere comprimir los datos, una muy buena libreríá puede ser miniz (https://code.google.com/p/miniz/).

Referencias

Páginas web de referencias de algoritmos, técnicas, librerías y código fuente que fue utilizado en el trabajo obligatorio.

Documentación de métodos, constantes, etc.
https://opengl.org/

Técnicas para el manejo de la cámara.
http://www.lighthouse3d.com/tutorials/glut-tutorial/keyboard-example-moving-around-the-world/

Código utilizado para cargar modelos 3ds y cargado de BMPs.
http://www.spacesimulator.net/tutorials/

Modelos 3ds gratuitos descargables
http://www.gameartmarket.com/details?id=ag9zfmRhZGEtM2RtYXJrZXRyMgsSBFVzZXIiE3BpZXJ6YWtAcGllcnphay5jb20MCxIKVXNlclVwbG9hZBjfjJmo6ycM