Programación Orientada A Objetos

Simulación de una Carrera Alienígena

Integrantes:

Franco Palau

Martin Olmos

Profesor: Cesar Aranda

Universidad Nacional de Cuyo

Facultad de Ingeniería

Ingeniería Mecatrónica

Año 2017

Índice

Introducción:

Descripción del problema

En este trabajo se aborda la implementación para una simulación de carrera Alienígena.

Dicha carrera consiste en naves alienígenas que se mueven en pistas elípticas. Cada nave está identificada, visualmente y en cada carrera, por un número asignado en orden diferente del resto. La cantidad de vueltas a realizar para ganar la carrera, la cantidad de naves (y por ende las pistas), así como las velocidades mínima y máxima que pueden alcanzar son fijadas por el operador.

A su vez cada nave posee una identificación única, visible y es capaz de variar al azar, durante el recorrido, tanto su color como su velocidad (en más o en menos de la actual y dentro de los límites establecidos) cada 180º recorridos.

Los datos de la velocidad y el tiempo ocupado por cada nave al momento de pasar por la línea de largada/llegada son guardados en un archivo para luego mostrarlos al finalizar la carrera.

Vale destacar que las naves siguen trayectorias también elípticas pero no necesariamente "concéntricas".

Identificación de temas abordados

Dentro de los principales temas que se abordan en este trabajo se incluyen:

_ Utilización de una librería para implementación de Interfaz Gráfica

_ Diseño e Implementación de un diagrama de clases

_ Uso de técnicas referidas al paradigma de objetos, tales como creación de clases y objetos, agregación y composición, encapsulamiento,etc.

_ Utilización de aspectos diversos referidos al lenguaje C++, tales como manejo de archivos, utilización de diferentes estructuras de datos, y su correspondiente manejo, etc.

Librería Utilizada

SDL: Simple DirectMedia Layer (Versión 2.0)

SDL es un conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan funciones básicas para realizar operaciones de dibujo en dos dimensiones, gestión de efectos de sonido y música, además de carga y gestión de imágenes. Pese a estar programado en C, tiene wrappers(capas) a otros lenguajes de programación como C++, Ada, C#, BASIC, Erlang, Lua, Java, Python, etc. También proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia.

Existen una serie de bibliotecas adicionales que complementan las funcionalidades y capacidades de la biblioteca base (SDL).

SDL Mixer: Extiende las capacidades de SDL para la gestión y uso de sonido y música en aplicaciones y juegos. Es compatible con formatos de sonido como Wave, MP3 y OGG, y formatos de música como MOD, S3M, IT y XM.

SDL Image: Extiende notablemente las capacidades para trabajar con diferentes formatos de imagen. Los formatos compatibles son los siguientes: BMP, JPEG, TIFF, PNG, PNM, PCX, XPM, LBM, GIF, y TGA.

SDL TTF: Permite usar tipografías TrueType en aplicaciones SDL

Razones de la elección de esta librería

- Biblioteca Multiplatarforma, ya que el proyecto se desarrolló en sistemas Linux y paralelamente en sistemas Windows
- Fácil y rápida Instalación
- Integración completa con los IDEs utilizados (Eclipse y Visual Studio)

- Amplio repertorio de información en la Web para su utilización

Instalacion de la Biblioteca

Lo primero que debemos hacer es instalar la biblioteca SDL , para ello desde la consola de linux ejecutamos los siguientes comando uno por vez.

Comandos:

_ apt-get install libsdl2debian

_ apt-get install libsdl2-dev

_ apt-get install libsdl2-mixer-dev

_ apt-get install libsdl2-ttf-dev

Una vez instaladas las librerias, nos dirigimos a Eclipse -> Propiedades del proyecto, allí en la pestaña de C/C++ Build, escogemos la opción Linker->agregar libreria, y escribimos lo siguiente:

-lSDL

-lSDLmain

-lSDL_mixer

-lSDL_image

-lSDL_ttf

Finalmente en nuestros archivos hay que agregar los correspondientes includes:

_ #include <SDL2/SDL.h>

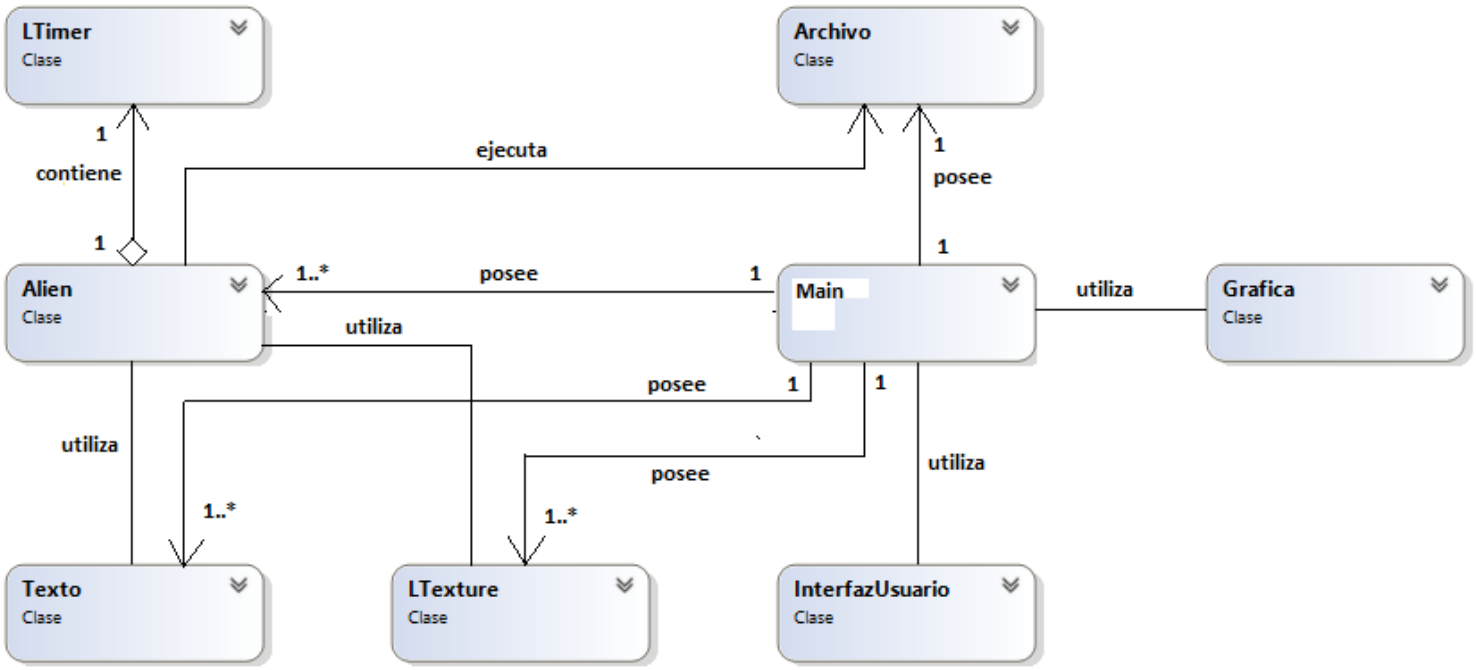
_ #include <SDL2/SDL_image.h>

_ #include <SDL2/SDL_mixer.h>

_ #include <SDL2/SDL_ttf.h>

Diseño de clases

A continuación se explicita el diagrama de clases en el cual se basó este proyecto:



Breve descripción de las clases utilizadas:

Alien: Es la clase más importante del programa. Se encarga de calcular las nuevas posiciones de los aliens, renderizar la imagen, variar colores y velocidades, así también como verificar diferentes estados del alien

Timer: Una clase simple que simula el funcionamiento de un timer. Le provee información a Alien.

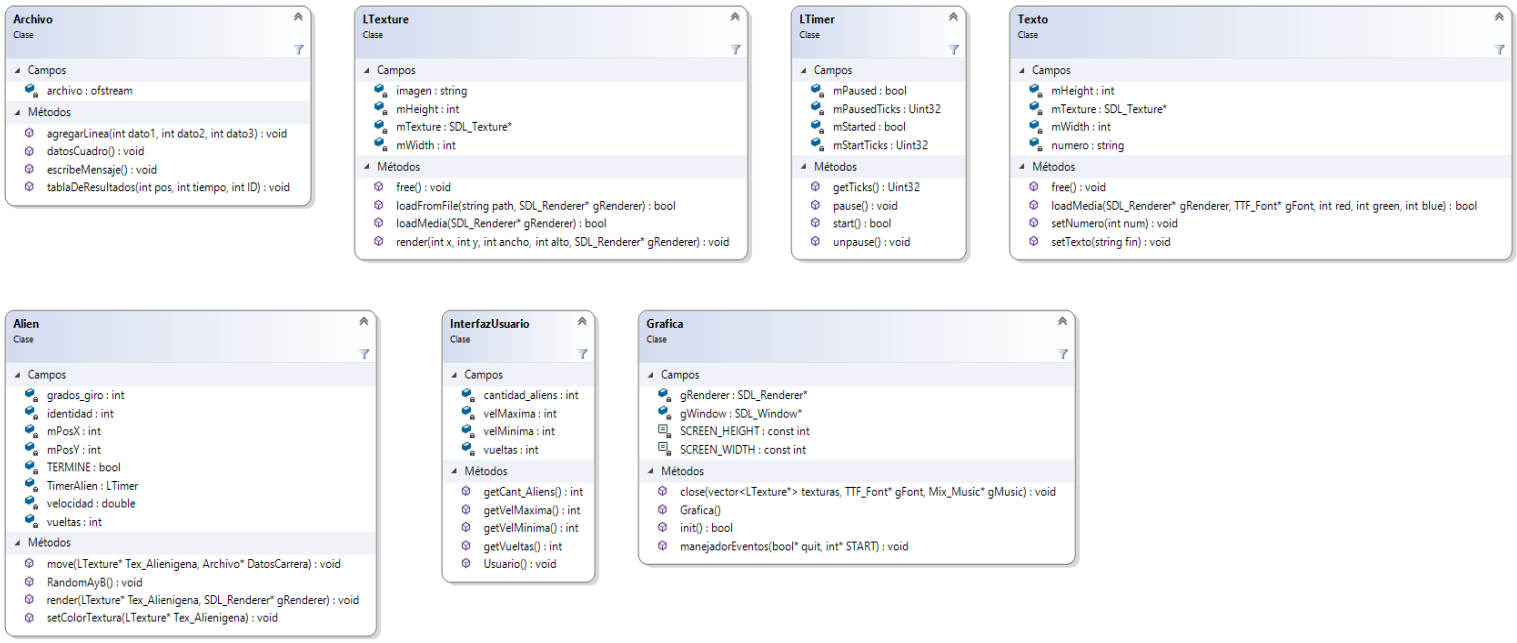
Archivo: Esta clase se encarga de recolectar los datos y guardarlos en un archivo.

Textura: Esta clase se encarga de cargar las imágenes y renderizarlas

Gráfica: Se encarga de desplegar la ventana principal, así como de cerrarla y liberar memoria

Texto: Se encarga de manejar todo texto que se muestre en pantalla, con sus respectivas características

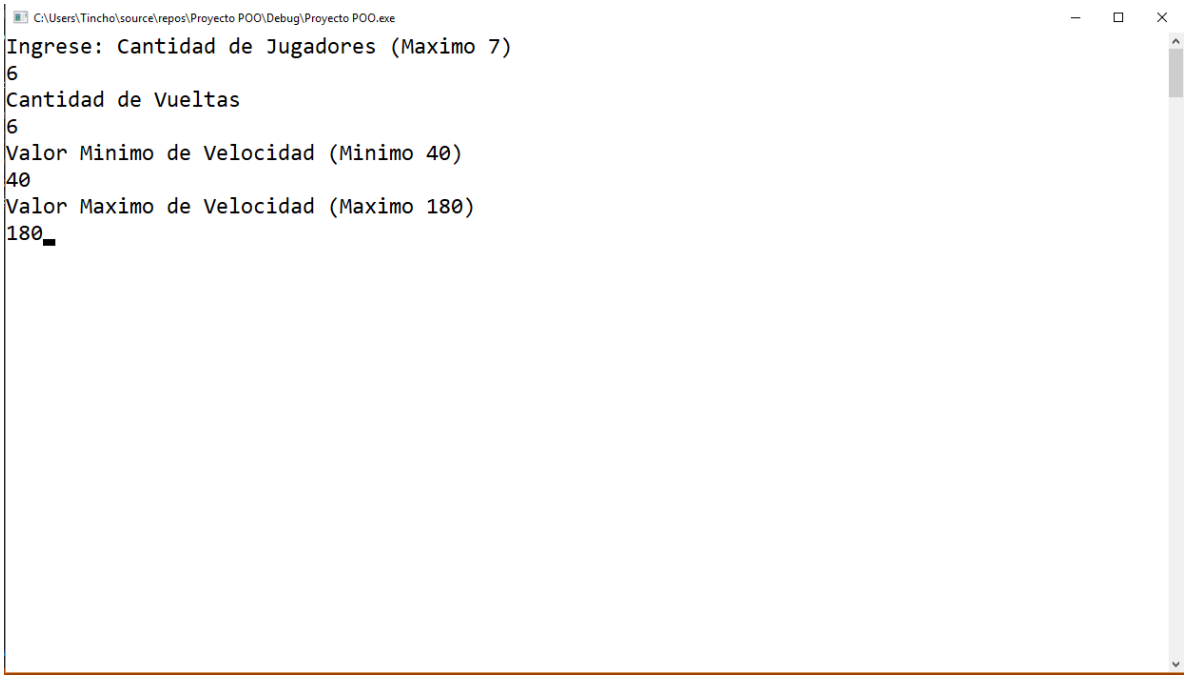
InterfazUsuario: se encarga de desplegar los mensajes de petición de datos, así como guardar los respectivos inputs.



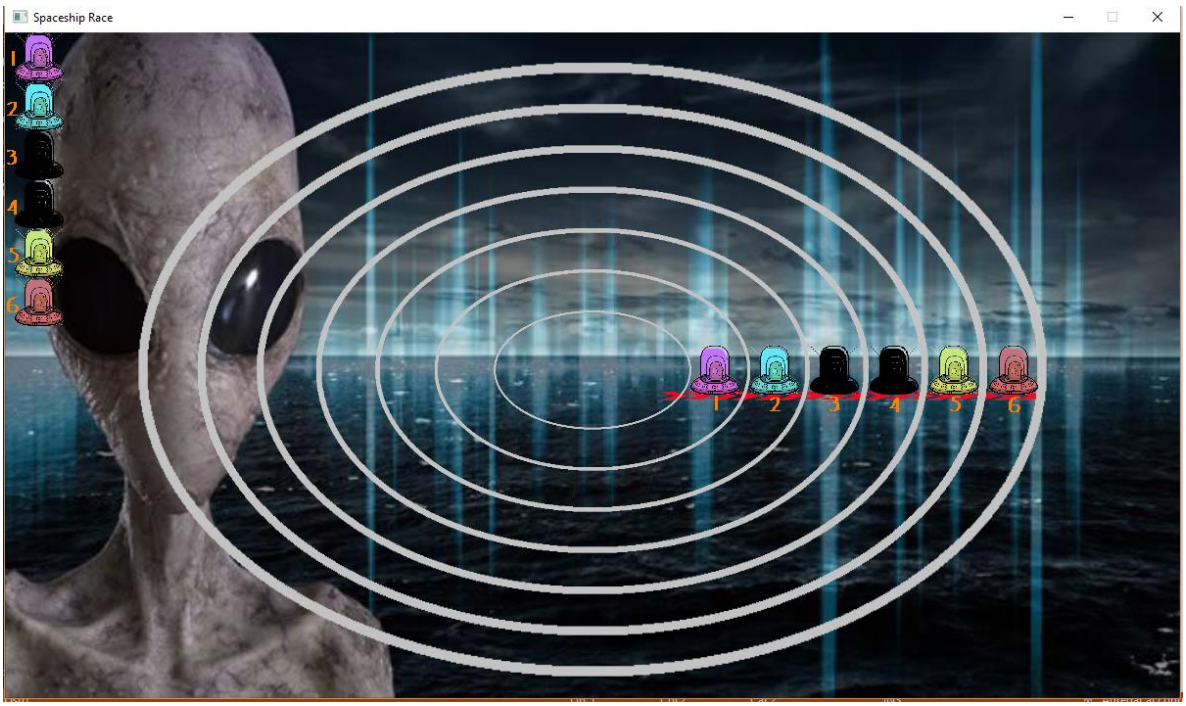
Aplicación

A continuación se muestran diferentes capturas de pantallas de los distintos estados por los que pasa el juego. Solo se ha hecho una ejecución ya que se extendería demasiado agregar más de una, se deja esto para la presentación oral.

Luego de cada imagen hay una breve explicación de lo que está ocurriendo durante ese momento en el juego.

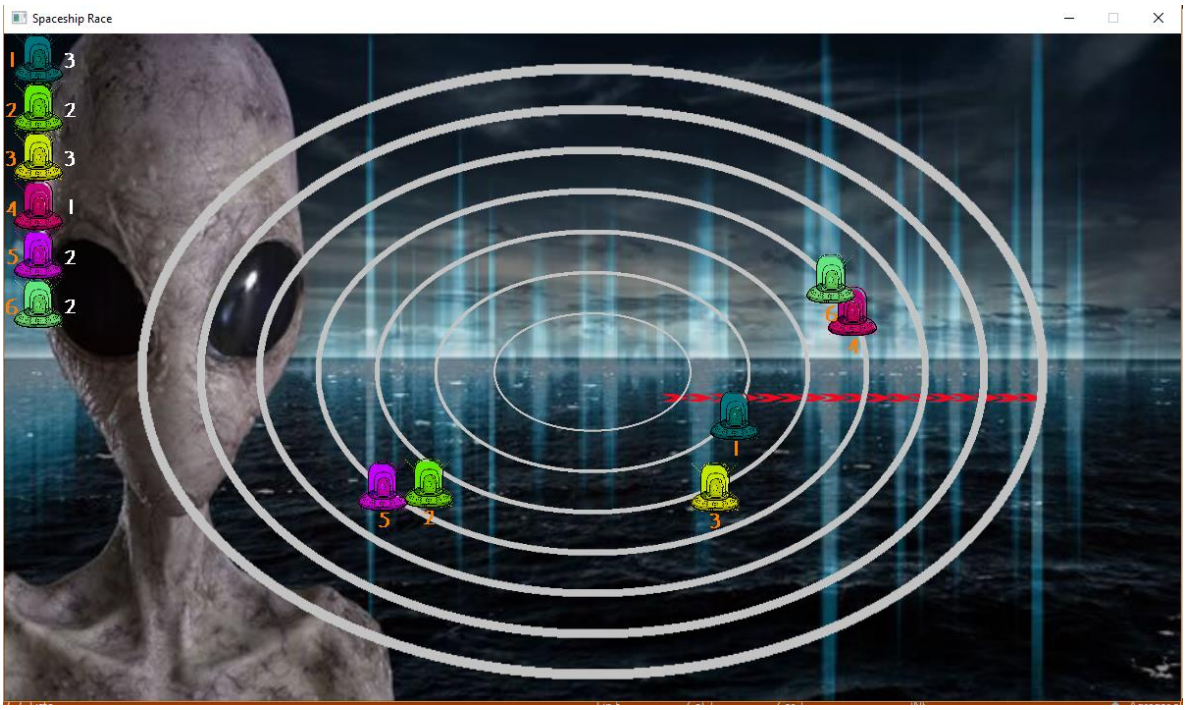


En primer lugar se abre una ventana donde se le pide al usuario que ingrese la cantidad de Aliens, la cantidad de vueltas a recorrer, y los valores mínimos y máximos de velocidad que puede tomar los aliens.



Luego, se muestra una pantalla como la anterior, donde vemos que se ha colocado los aliens en sus pistas, asi también como arriba a la izquierda se muestras los aliens y sus correspondientes identificadores (números).

En este caso el sistema no arranca hasta que presionamos la tecla “UP”.



Una vez presionada la fleca de arriba, comienza la carrera, es decir los aliens se empiezan a mover, y también comienza una música de fondo. Acá ocurren varios eventos: los aliens cada 180° van a cambiar tanto su color como su velocidad. Cada 360° (es decir una vuelta) veremos que el contador de arriba a la izquierda nos va mostrando las vueltas que cada alien va completando.

En esta imagen vemos cuando los aliens ya han finalizado la carrera, y arriba a la izquierda se muestra el mensaje "Ganador!" a aquel que termino la carrera primero.

El programa se queda esperando hasta que se apriete el icono "X" para salir del mismo.

Luego todos los datos de la carrera en tiempo real se van guardando en un archivo, el cual finalizada la carrera contiene la siguiente información:

Alien	Vuelta	Tiempo
1	1	8 ms
2	1	11 ms
3	1	13 ms
4	1	18 ms
5	1	20 ms
6	1	23 ms
2	2	17398 ms
4	2	18936 ms
5	2	25999 ms
3	2	28902 ms

```
   6                  2               30316 ms
   3                  3               43049 ms
   4                  3               53519 ms
   1                  2               54485 ms
   5                  3               57092 ms
   3                  4               57493 ms
   2                  3               57983 ms
   6                  3               59809 ms
   1                  3               61206 ms
   5                  4               62209 ms
   4                  4               62755 ms
   3                  5               63063 ms
   6                  4               66134 ms
   2                  4               66725 ms
   5                  5               69147 ms
   1                  4               69150 ms
   3                  6               73618 ms
   4                  5               75652 ms
   6                  5               76387 ms
   1                  5               78178 ms
   5                  6               78372 ms
   2                  5               79778 ms
   6                  6               85861 ms
   4                  6               86107 ms
   2                  6               87251 ms
   1                  6               91444 ms
LA CARRERA HA FINALIZADO
RESULTADOS FINALES:
Posicion        Alien        Tiempo
    1              3          73618 ms
    2              5          78373 ms
    3              6          85861 ms
    4              4          86107 ms
    5              2          87252 ms
    6              1          91445 ms
```

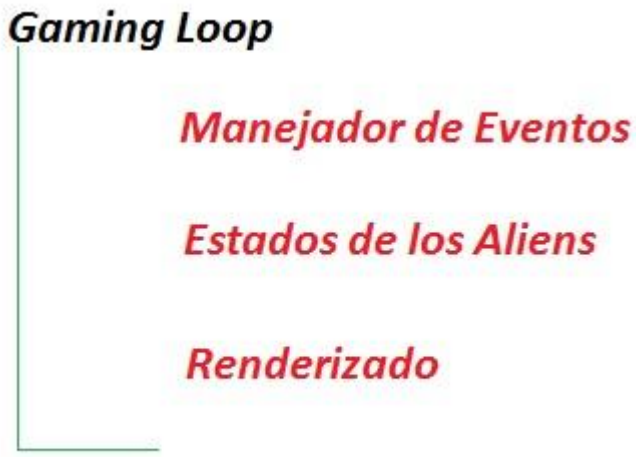
Partes Principales del Programa

Anteriormente se describió como se encaró este problema según la lógica de objetos, mostrando las correspondiente clases y el diagrama UML.

A continuación se muestran algunas líneas que se enfocan en la lógica que sigue el programa.

La lógica de este programa, así como la de muchos juegos, es el clásico estilo "Gaming Loop", donde el concepto básico es un bucle que corre hasta que el usuario por medio de alguna interrupción o comando le indica al programa que termine.

Dentro de este Loop se realizan todas las tareas necesarias para la ejecución del mismo. En nuestro caso, existen tres tareas principales a realizar: Hacer un polling de los eventos que van ocurriendo, Verificar el estado de los aliens y por ultimo Renderizar todas las imágenes con sus posiciones actualizadas



En primer lugar se muestra la inicialización de variables. Nótese que al tener varios Aliens (y por ende varias pistas, y varias texturas) se ha optado por utilizar la clase <vector> para manejar dichos Objetos.

```
vector<LTexture*>Pistas;
vector<LTexture*>texturas_alienigenas;
vector<Alien*>alienigenas;
vector<Texto*>texto;
vector<Texto*>texto_vueltas;
vector<int>tiemposFinales;
vector<int>idFinal;
Archivo DatosCarrera;
TTF_Font *TexturaTexto = NULL;
Mix_Music *Musica = NULL;
Grafica ventana;

LTexture *Fondo = new LTexture("Fondo.jpg");
LTexture *Meta = new LTexture("Meta.png");
```

Luego de esto, y posterior a la carga de las imágenes, comienza nuestro "Gaming Loop", el cual en primer lugar llama a la función manejador de eventos, la cual se muestra a continuación

```
void Grafica::manejadorEventos(bool* quit, int* START)
 {
    SDL_Event e;
    //Handle events on queue
    while (SDL_PollEvent(&e) != 0)
    {
        //User requests quit
        if (e.type == SDL_QUIT)
        {
            *quit = true;
        }
        else if (e.type == SDL_KEYDOWN) {
            switch (e.key.keysym.sym) {
            case SDLK_UP:
                *START = 1;
                break;
            }
        }
    }
}
```

```
bool quit = false;
while (!quit)
{
    ventana.manejadorEventos(&quit, &START);
    if (START == 1){
        if (Mix_PlayingMusic() == 0){
            Mix_PlayMusic(Musica, -1);
        }
        for (int i = 0; i < alienigenas.size(); i++) {
            if (alienigenas[i]->getTimer()->isStarted()) {
                if (!(alienigenas[i]->getTermine())){
                    alienigenas[i]->move(texturas_alienigenas[i], &DatosCarrera);
                    texto_vueltas[i]->setNumero(alienigenas[i]->getVueltas());
                }
                else {
                    if (alienigenas[i]->getTimer()->isPaused()){
                        tiemposFinales.push_back(alienigenas[i]->getTimer()->getTicks());
                        idFinal.push_back(alienigenas[i]->getID());
                        alienigenas[i]->getTimer()->unpause();
                        if (tiemposFinales.size() == alienigenas.size()){
                            DatosCarrera.escribeMensaje();
                            START = 0;
                            for (int k = 0; k < tiemposFinales.size(); k++) {
                                DatosCarrera.tablaDeResultados(k,tiemposFinales[k],idFinal[k]);
                            }
                        }
                    }
```

En la última imagen de arriba se explicita la forma en que se verifican los estados de los Aliens. Para resumir el comportamiento de este paquete de código, se chequea el timer de cada Alien y si está andando se llama a la función "move()". En caso de que el Alien ya haya terminado, no entra en esa porción de código, sino que se pausa el timer, se almacena el tiempo y luego se escribe ese tiempo en el archivo.

A continuación se muestra la función "Move()", la responsable de calcular la posición del alien, y verificar que pasa cada 180 y 360°.

```
void Alien::move(LTexture* Tex_Alienigena, Archivo* DatosCarrera)
{
    mPosX = x0 + a*cos(velAngulo);
    mPosY = y0 + b*sin(velAngulo);
    grados_giro = (velAngulo*180/M_PI);
    if (fmod(grados_giro,180) == 0 && grados_giro!=mPosYact2)
    {
        mPosYact2 = grados_giro;
        velocidad = rand() % (velSup - (velInf + 1)) + velInf;
        velocidad = velocidad / 10000;
        RandomAyB();
        setColorTextura(Tex_Alienigena);
    }
    if (fmod(grados_giro,360) == 0 && grados_giro != mPosYact1)
    {
        mPosYact1 = grados_giro;
        vueltas++;
        DatosCarrera->agregarLinea(identidad, vueltas, TimerAlien.getTicks());
        if (vueltas == cantidad_vueltas) {
            TERMINE = true;
            TimerAlien.pause();
        }
    }
    velAngulo = velAngulo + velocidad;
}
```

Conclusiones

Con la realización de este trabajo se pudo aplicar varios conceptos vistos durante el cursado, referidos tanto al paradigma orientado a objetos como al lenguaje de programación utilizado, C++.

Así mismo, debido a que el trabajo se realizó en grupo, tuvimos que implementar metodologías varias de trabajo, como por ejemplo, el uso de

la herramienta GitHub; así también como trabajar "modularmente" para que cada integrante pudiera enfocarse en una (o más) clases, y luego fusionar el resultado con el programa principal.

También, pudimos analizar, aprender e implementar varios conocimientos adquiridos sobre programación aplicada a interfaces gráficas, tema completamente nuevo para nosotros.

Análisis de ventajas/desventajas de los componentes usados.

El componente principal utilizado fue la librería SDL2 , la cual, como se mencionó anteriormente, permitió el trabajo en diferentes plataformas, para las cuales otras librerías no serían compatibles.

A su vez vale recalcar que dicha librería contiene un set de variadas funciones, las cuales nos sirvieron para la implementación de todas las características y funciones de nuestro proyecto.

También se usaron diferentes funcionalidades del lenguaje C++, como por ejemplo la clase <vector>, la cual nos permitió manejar de forma sencilla y eficaz todos los sets de objetos que el programa utiliza.

Posibles extensiones a la solución dada

Dentro de las posibles extensiones, se podría implementar una pantalla de inicio, con alguna animación o comunicación con el usuario. Ademas, al finalizar la carrera se podría mostrar una ventana estilo "pop up" con un mensaje de finalización y la posibilidad de reiniciar la carrea.

Otra alternativa más desafiante, sería generar, antes de la carrera, un sistema de apuestas en base a cada alien, y luego de la carrera mostrar los porcentajes ganados por cada jugador.

Por último, agregar diferentes "poderes" a cada alien sería una posibilidad muy atractiva.

Referencias

_ Como Programar C, C++ y Java 4ta Edición - Deitel & Deitel

_ http://wiki.libsdl.org/FrontPage

_ http://lazyfoo.net/tutorials/SDL/