**Blockbuster – Renta y Compra de Películas**



**Escuela**: Instituto Tecnológico de Tijuana (*Tomás Aquino*)

**Profesor**: Luis Mitre Padilla

**Materia**: Fundamentos de Programación

**Grupo**: 1D (*Turno Vespertino*)

**Integrantes del equipo:**

1. López Garay Luis Felipe
2. Tapia Ramos Itzel
3. Moreno Moreno Diego
4. Díaz Martínez José Luís

**Introducción**

A la hora de elegir el tipo de programa que íbamos a desarrollar, nos planteamos cosas como un sistema de correo electrónico simulado, un programa interbancario, un juego de ajedrez, un piano, un Blockbuster entre otras cosas. Al final decidimos que sería buena idea crear un sistema de rentas y compras online con la temática de algo que a los 4 nos gusta: el cine y las películas. Tardamos un poco en conceptualizar lo que queríamos empezar a diseñar y desarrollar así que durante las clases de programación, a las 8 de cada día, nos pasábamos bocetos e ideas de lo que deberíamos implementar, las reglas de las compras, rentas y afiliaciones, fechas de expiración y los menús que debería haber tenido el menú de clientes. Hubo varias cosas que quedaron al aire como el menú de empleados y los gráficos que íbamos a usar pero si en una cosa estuvimos de acuerdo es que yo, Felipe, durante mi aprendizaje del lenguaje C++ me topé con la dificultad de utilizar los tipos de datos primitivos char[n] y el manejo de arreglos en un lenguaje de medio nivel como C; es por eso que durante mi visita a video tutoriales y páginas de referencia sobre C++, encontré con que este lenguaje incluye 2 clases muy útiles: String y Vector. Acordamos utilizar strings y vectores (arreglos) por su facilidad de uso, comprensión, implementación y la cantidad de funciones de ayuda que se podían utilizar para operar entre ellas además que son conceptos con los que ya habíamos trabajado antes en otros lenguajes como Python o JavaScript.

Antes de empezar a desarrollar, nuestro compañero Diego propuso un diseño tentativo para lo que sería el menú de clientes, creando por separado una función que se encargaría de trazar los bordes de la interfaz de usuario y esto me dio una idea; utilizar esos mismos bordes para todos los menús, para así darle una sensación de encapsulamiento y consistencia en el lenguaje de diseño establecido. No nos conformamos con simples listados de opciones así que implementamos un sistema ambicioso de botones creados a partir de puro texto y símbolos igual y menos; fue muy difícil implementar al principio ya que no podía controlar las direcciones tan fácilmente hasta que se me ocurrió la idea de hacer un arreglo para todos los menús en los que hubiera que desplazarse entre botones y/o campos de texto (captura de datos). Como veía la limitación del usuario en una interfaz de usuario inspirada altamente en mi fascinación y experiencia con las aplicaciones y formularios web (botones, alertas, gráficos de apoyo, ventanas, campos de texto, sub-menús ocultos, opciones y checkboxes) tuve que pensar en cómo se tendría qué desplazar el usuario entre tanto elemento de la interfaz, así entonces se me ocurrió detectar la tecla que presionara el usuario y buscando un poco en Internet se logró encontrar un ciclo infinito que pedía una tecla y en base a la tecla presionaba, actuaba de forma que yo quisiera, es así como se hacen los juegos en consola.

Este ciclo decidí romperlo de 2 formas, una al detectar que el usuario presionara ESC y la otra, forzarlo a que se rompa cuando se haga algo dentro del menú en cuestión que requiera volver al menú anterior, para esto, debajo del ciclo había una función que dibujaba el menú anterior, y en el menú anterior justo después de la llamada a la función de mayor jerarquía, re-acomodar todo para que quede como recién abierto o como se había quedado antes de llamar a la función (menú). Esto se logró por medio de una cascada de ciclos while y llamados a funciones más arriba en la jerarquía para que el programa fluyera como una especie de muñeca rusa o las ramas de un árbol, subiendo y bajando entre ellas cuando fuera necesario, esto lo desarrollamos José Luís y yo, Felipe.

Otra parte de la metodología fue separar el comportamiento del programa en 4 simples archivos, el principal que lanzaría los menús de registro o login e incluyera todas las librerías y archivos de ayuda que hubiera qué poner en el orden adecuado para que todo fluyera correctamente; el de *render* que, curiosa palabra se me quedó pegada desde hace mucho, maneja todo lo que tenga qué ver con dibujar menús, avisos, mensajes, errores, vaciar datos en sus respectivas coordenadas y animaciones o gráficos ASCII Art; el que más orgullo y satisfacción nos causó, el que database que maneja todo lo que tenga qué ver con la lectura y escritura de archivos, para explicar el funcionamiento y filosofía de trabajo de este archivo se hará en otro párrafo; uno que contuviera utilerías y funciones de mucha ayuda para un código más eficiente y expresivo, cosas como manejar operaciones con fechas, separar o unir arreglos, convertir entre tipos de datos entero-texto y viceversa, etc; por último pero no menos importante, de hecho el núcleo de nuestro programa y donde el código de todos los menús reside, el archivo menus.cpp que contiene todos los menús y sub-menús, cada una en su respectiva función y en un orden adecuado descendente. Fue de gran importancia haber separado el diseño de la lógica principal del programa pues esto nos permitió tener un código muy limpio y mantenible, reutilizable y expandible.

Para mi diseño de las ideas que fluían de nuestra mente, decidí utilizar Excel pues pude especificar las coordenadas y diseñar cómodamente, lo que veía era lo que iba a obtener y cuando le pedía a Diego que me los transformara a coordenadas y couts en C++, le era muy fácil pues ya estaba todo puesto en el archivo de Excel.

El logro más grande fue la idea de visualizar el manejo de archivos como si estuviera trabajando con una base de datos del tipo SQL, ya que es un lenguaje y plataforma con la que ya he trabajado antes, los archivos se pueden conceptualizar como una tabla, cada línea un registro y cada cadena de texto separada por, valga la redundancia, un separador (‘;’ el más frecuente o ‘,’ para sub-arreglos) representa un campo o celda. De hecho, hasta abajo del archivo database.cpp (por cierto, utilizamos nombres en inglés como render y database porque estamos familiarizados con ellos y suena muy cool y universal, no porque lo hubiéramos copiado de algún lado).

El proyecto llevó demasiado tiempo y aunque **no lo hemos terminado**, fuimos realistas y nos tomamos muy muy enserio el diseño de interfaz y la experiencia de usuario cómoda, entendible y de fácil manejo y con muchas utilidades, teniendo en cuenta cada posible eventualidad (para eso fue el *debugueo* de la versión **v0.7-Alpha**) para que al usuario no le ocurra. Lamentamos que no se haya podido concluir pero prometemos que la versión final estará en Github a más tardar el Viernes.

Cosas de gran utilidad y conveniencia como el paginado en el menú ‘Catálogo’, la búsqueda de películas en tiempo real, el manejo de fechas (algoritmos simples pero difíciles de concebir a las 2 de la mañana en un pizarrón pequeño y con dolor de cabeza), checkboxes para activar o desactivar funcionalidades y crear melodías con C++ llevaron su tiempo y quizá pudimos haberlas omitido para terminar a tiempo, pero estoy, tanto yo como el equipo, muy orgullosos y sorprendidos de lo que logramos implementar sin ser expertos en el lenguaje C++ y la pesadez que causa limitarse a la consola de Windows.

**Marco Teórico**

***Vector***: Clase y plantilla de C++ que permite manejar arreglos de un tipo de datos con la conveniencia de poder cambiar su tamaño, agregándole y quitándole elementos, tiene funciones muy útiles y simples de entender como .push\_back() (insertar) o .size() (cantidad de elementos dentro del arreglo).

***String***: Arreglo de caracteres, en C++, se utiliza como una plantilla fácil de usar, con la posibilidad de aplicar el operando de suma, resetearla y pasarla como parámetro fácilmente entre funciones. Sin mencionar que tiene una sintaxis muy fácil de recordar.

***SQL***: Lenguaje de programación y plataforma de base de datos desarrollada por SUN Microsystems. Un ejemplo de consulta puede ser SELECT FROM películas WHERE genero = ‘Drama’ y fue esto lo que nos permitió conceptualizar database.cpp.

***Toggle***: Cambiar el estado de algo, de forma binaria, encendido o apagado.

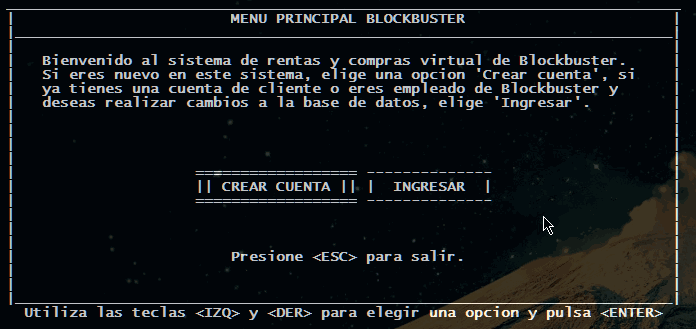
***User Experience***: Conocida como experiencia de usuario, es el estudio de las necesidades y acciones del usuario ante una interfaz humano-maquina o pieza de software. Permite desarrollar interfaces que sean fáciles de entender, usar y que hagan el uso de un programa algo placentero, que el usuario sepa lo que está usando y cómo usarlo.

***Render***: En jerga informática se refiere al proceso de generar una imagen o vídeo mediante el calculo de iluminación GI partiendo de un modelo en 3D. Un termino quizá mal utilizado por nosotros pero que en definitiva suena muy cool. Aparte que no quería un archivo llamado diseño.cpp para evitar problemas con la codificación.

**Parámetro**: Datos que recibe una función, se debería decir argumento pero muchas personas lo entienden también como parámetro. La función toma un argumento o varios como datos de entrada y los utiliza pera producir una salida.

**Conclusiones**

Fue un proyecto de mucha dificultad por la cantidad de características que implementamos y queríamos implementar, al final, estamos felices con lo que logramos y para dar unos ejemplos de la interfaz y de cómo funciona el código, se puede ir a código fuente y leerlo sin más pues traté de comentarlo y hacerlo súper expresivo y legible.





****

**Referencias**

<http://www.cplusplus.com/reference/vector/vector/>

<http://www.cplusplus.com/reference/string/string/?kw=string>

<https://technet.microsoft.com/en-us/library/bb264565%28v=sql.90%29.aspx?f=255&MSPPError=-2147217396>

<http://www.phy.mtu.edu/~suits/notefreqs.html>

<http://www.cplusplus.com/forum/beginner/12272/>

<http://stackoverflow.com/questions/4060601/make-sounds-beep-with-c>

<http://code.runnable.com/VHb0hWMZp-ws1gAr/splitting-a-string-into-a-vector-for-c%2B%2B>

<http://en.cppreference.com/w/cpp/algorithm/transform>

<http://www.elcodigoascii.com.ar/>

<http://stackoverflow.com/questions/10463201/getch-and-arrow-codes>

<http://www.cplusplus.com/doc/tutorial/preprocessor/>

<http://www.cplusplus.com/reference/ctime/>

<http://stackoverflow.com/questions/9987562/determining-the-difference-between-dates>

<http://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm>

<http://www.cplusplus.com/reference/string/string/c_str/>

<http://stackoverflow.com/questions/5590381/easiest-way-to-convert-int-to-string-in-c>

<http://www.cplusplus.com/doc/tutorial/files/>

<http://mathbits.com/MathBits/CompSci/Files/Append.htm>

<http://www.cplusplus.com/forum/general/151146/>

<http://www.cplusplus.com/reference/algorithm/>

<http://www.cplusplus.com/reference/cstdlib/>

<http://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm>