UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

Rodriguez Rentería Jesus Alejandro

Código:215510307

Materia: Seminario de solución de problemas de estructuras de datos II

Profesor: Gómez Anaya David Alejandro

Sección: D16

Sede: Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI)

Fecha:25 de octubre del 2020

1. Planteamiento del problema

Se busca modelar un personaje para un juego con programación orientada a objetos. La creación de personaje se lleva en memoria principal, pero a falta de la utilización de listas, todas las cuentas se almacenan en memoria secundaria. El personaje que este en memoria principal puede jugar, comprar cosas y guardarla en su inventario, ganar experiencia, modificar sus atributos, etc.

2. Objetivos

Implementar una lista de personajes en un fichero binario, utilizando la estrategia de almacenamiento secundario: Campos de dimensión.

1. Realizar misiones: implementar un sistema de juego para completar una misión con éxito, o fracasar.

2. Añadir, utilizar y desechar artículos de una lista de artículos.

3. Marco teórico.

Para el entendimiento de la práctica se necesita conocer el uso de los ficheros: Tenemos las clases fstream (fichero, en general), ifstream (fichero de entrada) y ofstream (fichero de salida), todas ellas definidas en "fstream.h". Leeremos y escribiremos con << y >>, al igual que para la pantalla. Cerraremos un fichero con "close"(tanto si lo hemos abierto como para leer o para escribir) y comprobaremos si se ha terminado un fichero de entrada con "eof".

4. Desarrollo.

El código cuenta con 2 clases que son FileList y personaje.

La clase FileList cuenta con un solo atributo de tipo string la cual guarda el nombre del archivo binario que se va a crear en un futuro o ya existe.

Los métodos con los que se cuenta son:

1. FileList(string): Es un constructor con parámetros
2. void add(Personaje obj): Se ingresa un personaje como parámetros y la función se encarga de leer los atributos y su largo. Esto con el fin de hacer el almacenamiento en el archivo binario con los campos de dimensión necesarios. Apéndice 1
3. void showall(): Función extra para pruebas. Recorre el archivo binario en busca de los personajes para poderlos instanciar en memoria y hace la impresión de estos.
4. Personaje* find(string): Función para el inicio de sesión de cuentas busca la cuenta en el archivo binario y regresa el objeto instanciado en memoria principal

5. bool find_nombre(string): Realiza un recorrido por el archivo binario haciendo comparaciones de los nombres si es que se llega a tener éxito con la búsqueda regresa true.
6. void remove(string): Se ingresa un nombre por parámetros (string) y se realiza la búsqueda si hay coincidencias se elimina el objeto relacionado de archivo binario.

La clase personaje cuenta con 7 atributos de los cuales nombre, clase, bando historial de misiones, e inventario son de tipo string y los atributos de nivel y experiencia de tipo int. Los métodos con los que se cuentan son los siguientes:

1. Personaje(): Constructor sin parametros
2. Personaje(string,string,string,string,string,int,int): Constructor con parámetros
3. string toString(): imprime todos los elementos de un objeto y los regresa en string
4. string get_nombre(): obtiene el nombre del objeto.
5. string get_clase(): obtiene la clase del objeto.
6. string get_bando(): obtiene el bando del objeto.
7. string get_historial_misiones(): obtiene todo el historial de misiones.
8. string get_inventario(): obtiene el inventario del objeto.
9. int& get_nivel(): obtiene el nivel del objeto.
10. int& get_experiencia():obtiene la experiencia del objeto.
11. void set_nombre(string): ingresa un nombre al objeto (usado para la función modificar)
12. void set_experiencia(int): ingresa experiencia al objeto (Usado para la función de modificar).
13. void set_nivel(int): ingresa un nivel al objeto.
14. void set_historial(string): Agregar el historial de batalla del objeto.
15. string imprime_historial(): imprime de manera ordenada las batallas del objeto
16. void mod_inventario(): Modificar inventario lee la clase a la que pertenece el personaje ya sea unidad de infantería o unidad blindada en cuyo caso se desplegara un menú de armas o modelos de tanque de los cuales puede añadir a su inventario.

17. void mision(): Se revisa la clase del personaje. Se tiene que indicar la arma con la cual se tiene que presentar al campo de batalla. Para posteriormente empezar el juego y ganar experiencia.

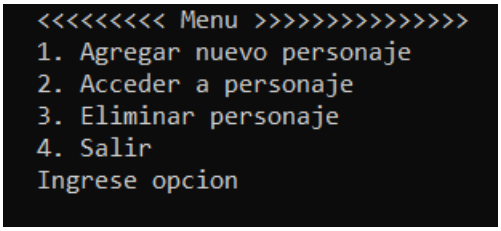
18. void aumento_exp(int): Función que es llamada por misión para hacer la suma de los puntos de la misión en cuestión.

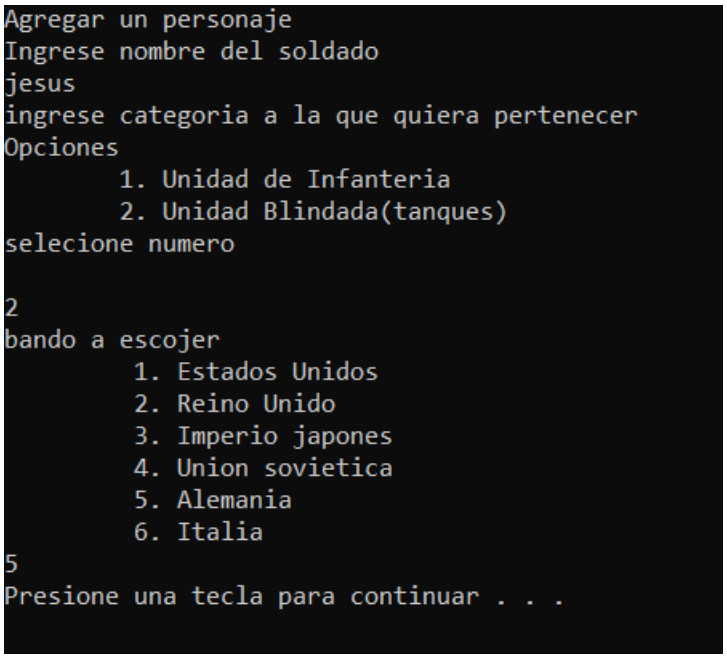
El menú principal cuenta con 3 funciones:

1. Agregar un nuevo personaje: Solicita todos los atributos para instanciar un objeto de tipo personaje. La función manda llamara a las validaciones de nombre repetido y guarda el personaje en el archivo binario.
2. Acceder a personaje: Pide el nombre del personaje que desea iniciar sesión y otorga un. Despliega un sub menú de todas las acciones que se pueden hacer con la cuenta.
3. Eliminar Personaje: Solicita el nombre del personaje que desea eliminar. Busca en el archivo binario si es que se encuentra coincidencias borra el registro.
4. Salir.

5. Pruebas y resultados.

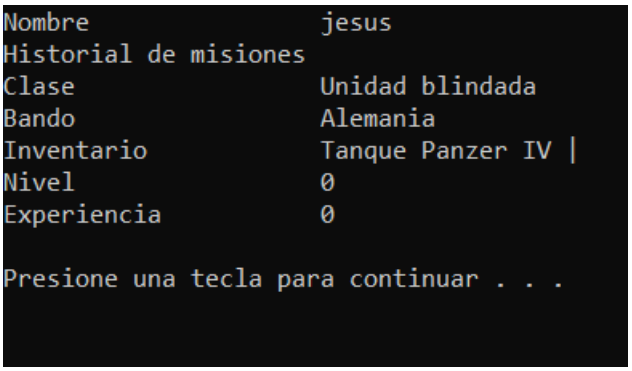
Se muestra el menú principal al iniciar el programa

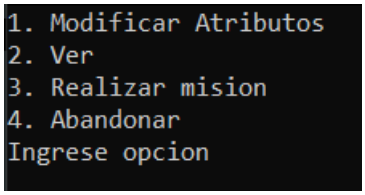




Se muestra la opción uno de agregar nuevo personaje. Podemos observar todo los bandos que se pueden elegir.

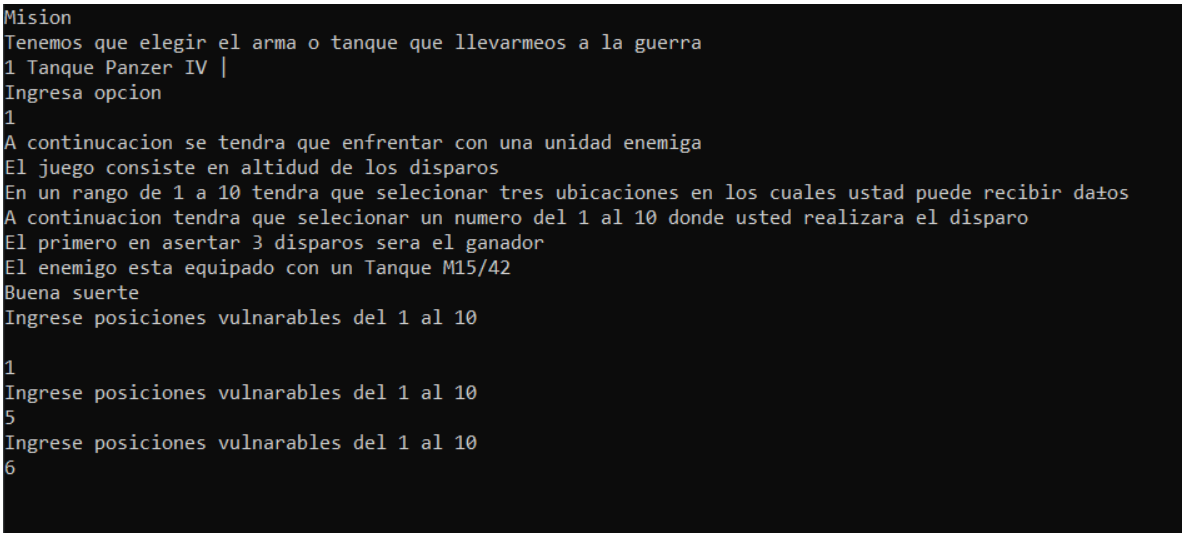
En la imagen se contempla el objeto de tipo personaje ya instanciados y corriendo en memoria principal la cuenta





Menú cuando hay una cuenta activa en memoria principal (donde se elige si se puede jugar hacer modificaciones, etc)

Misión se contempla que en la pantalla se distingue las opciones de armas a utilizar las reglas del juego y las posiciones en las cuales es posible llegar a recibir algún daño por parte del enemigo



6. Conclusiones.

La creación de este programa implico el uso de apuntadores char que fueron la principal ayuda para poder hacer el almacenamiento de los datos, primero ingresando el largo de nuestro dato y después la cadena de datos como tal. De igual manera la lectura de los datos se ejecuta con el mismo orden. El archivo de texto binario creado es de alguna manera la columna vertebral para el funcionamiento de del código.

7. Apéndices.

Apéndice 1.

La función de agregar personaje(código). Se puede visualizar como se cuenta el largo de la cadena y se guarda para después almacenar el contenido, esto se repite varias veces con todos los atributos a guardar.

Una vez que se termina el objeto en memoria principal queda respaldado en memoria secundaria. Por lo tanto podemos seguir usando nuestras variables en memoria principal para otros objetos de tipo personaje

```
void FileList::add(Personaje obj){
    int contcad;

    ofstream fout;
    fout.open(filename,ios::binary|ios::app);
    contcad = obj.get_nombre().length();
    //Ingresar el espacio en bites de la cadena
    fout.write((char*) &contcad, sizeof(int));
    //Almacena de manera concecutiva la cadena
    fout.write(obj.get_nombre().c_str(), contcad*sizeof(char));
    contcad= obj.get_clase().length();
    fout.write((char*)&contcad, sizeof(int));
    fout.write(obj.get_clase().c_str(), contcad*sizeof(char));
    contcad =obj.get_bando().length();
    fout.write((char*)& contcad, sizeof(int));
    fout.write(obj.get_bando().c_str(), contcad*sizeof(char));
    contcad =obj.get_historial_misiones().length();
    fout.write((char*)& contcad, sizeof(int));
    fout.write(obj.get_historial_misiones().c_str(), contcad*sizeof(char));
    contcad= obj.get_inventario().length();
    fout.write((char*)& contcad, sizeof(int));
    fout.write(obj.get_inventario().c_str(), contcad * sizeof(char));

    fout.write((char*) &obj.get_nivel(), sizeof(int));
    fout.write((char*) &obj.get_experiencia(), sizeof(int));
}
```