USAC GAMES – BATALLA NABAL

Documento de apoyo técnico.

Por Gladys Leticia Ajuchán Vicente - RA201807389

MANUAL TÉCNICO

Objetivo:

Aplicar los conocimientos del curso Estructuras de Datos en el desarrollo de una aplicación que permita manipular la información de forma óptima.

Objetivos específicos:

- Demostrar los conocimientos adquiridos sobre estructuras de datos lineales poniéndolos en práctica en el desarrollo del juego batalla naval.
- Utilizar el lenguaje C++ para implementar estructuras de datos lineales.
- Utilizar la herramienta Graphviz para graficar estructuras de datos lineales.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación.

ESPECIFICACIONES TÉCNICAS

Requisitos de hardware

- Procesador Intel preferiblemente superior a 4ta generación en caso de AMD superior a Opteron.
- 2GB de Memoria RAM

Requisitos de software

- Contar con un sistema operativo compatible con Python, Windows, MacOS, distribuciones de Linux.
- Tener Python instalado en el sistema operativo, muy importante para poder ejecutar los comandos y pueda visualizarse la interfaz gráfica.

BATALLA NABAL

La empresa Usac Games desea implementar un videojuego que permitan desarrollar la agilidad mental de los usuarios, por lo cual ha planeado desarrollar una aplicación con el juego Batalla naval y le solicita a usted como estudiante de estructura de datos poder implementar algoritmos, funciones y estructuras que permitan que el juego tenga un rendimiento óptimo y fluido. Debido al alcance y complejidad de las funcionalidades del videojuego se ha decidido dividirlo en 3 fases.

A continuación, se describen de forma general las funcionalidades a cubrir en cada en la Fase 1, utilizando el lenguaje de programación C++.

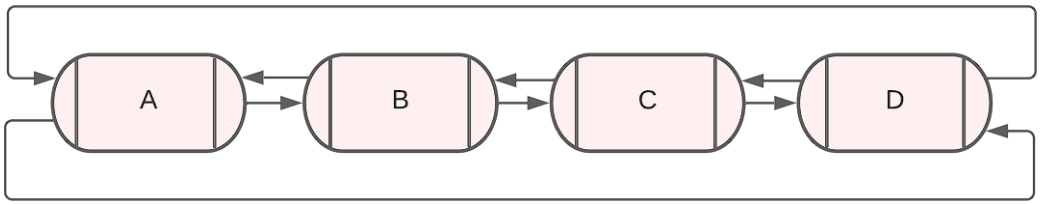
Las Listas estructuras utilizadas son las siguientes:

- ***Lista:*** Registro de usuarios.
  Encriptación: Aplicar seguridad a la información de los usuarios.
- ***Lista de listas***: Tienda de Skins del juego (los puntos por partida serán la moneda).
- ***Pila:*** Retroceder jugadas (Push y pop de movimientos) .
- ***Cola***: Tutorial del juego (push y pop de información con movimientos).

ESTRUCTURAS IMPLEMENTADAS

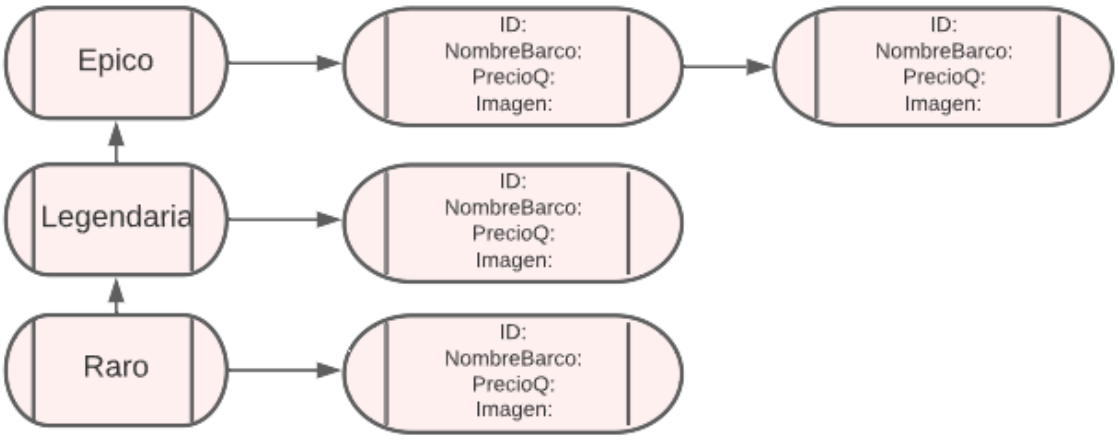
Lista

Esta lista contiene un nodo *principio* que apunta al primer elemento y un nodo *último,* cada elemento contiene una referencia al nodo siguiente y al nodo anterior excepto el primero y el último los cuales tienen un apuntados a NULL hacia el anterior y el siguiente respectivamente.



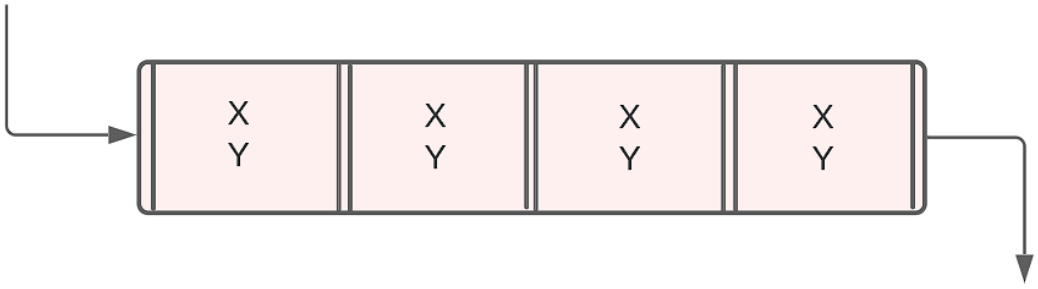
Lista de listas

Esta lista contiene un nodo *principio1* que se forma a través de un arreglo con n cantidad de artículos relacionados por la categoría, esto se almacena a través de una lista conteniendo más listas.



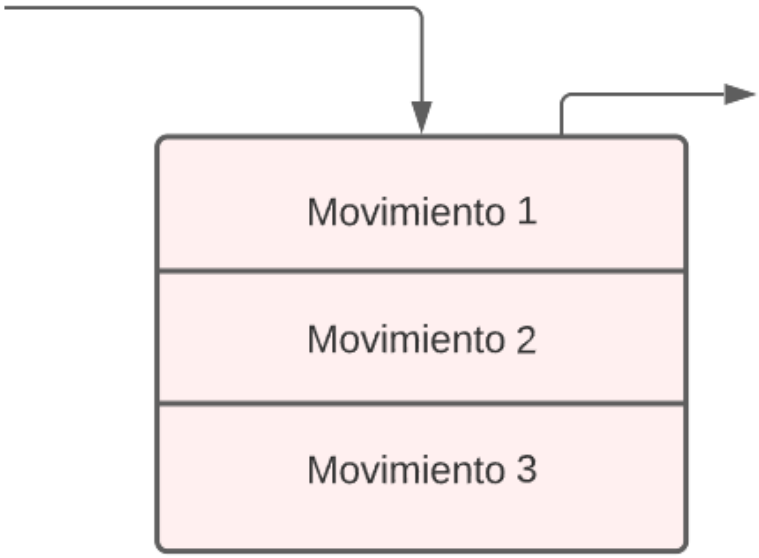
Pila

La pila se encuentra implementada heredando de la lista enlazada simple ya que es necesario manipular el nodo *principio2*.



Cola

La cola se encuentra implementada en la lista enlazada doble en la cual expone los métodos de agregar al inicio y eliminar al final de manera que la infraestructura es FIFO.



MODELOS

Ingresar:

```
void ListaUsuarios::Ingresar(string Nickname, string Password, int Monedas, int Edad) {
    Usuario* ingresando = new Usuario();
    ingresando->Nickname = Nickname;
    ingresando->Password = sha256(Password);
    ingresando->Monedas = Monedas;
    ingresando->Edad = Edad;
    NodoUsuario* nuevo = new NodoUsuario();
    nuevo->user = ingresando;
    ingresando = NULL;

    if (principio == NULL) {
        principio = ultimo = nuevo;
        principio->siguiente = ultimo;
        ultimo->atras = principio;
        contador++;

    } else {

        if (BuscarUsuario(Nickname) == false) {
            ultimo->siguiente = nuevo;
            nuevo->atras = ultimo;
            ultimo = nuevo;
            principio->atras = ultimo;
            contador++;
        }
    }
}
```

Ingresando Tutorial:

```
void Tutorial::IngresandoTutorial(int x, int y) {
    Nodo* nuevo = new Nodo();
    nuevo->x = x;
    nuevo->y = y;
    if (principio2 == NULL) {
        principio2 = nuevo;
    } else {
        Nodo*temp = principio2;
        while (temp->siguiente != NULL) {
            temp = temp->siguiente;
        }
        temp->siguiente = nuevo;
    }
}
```

Ingresando Producto:

```
void Tienda::IngresandoProducto(int id, string categoria, double precio, string nombre, string desconocido) {
    NodoCategoria* nuevo= new NodoCategoria();
    nuevo->NombreCategoria=categoria;
    if(principio3== NULL) {
        nuevo->productos->IngresandoProducto(id, categoria, precio, nombre, desconocido);
        principio3=nuevo;
    }else{
        NodoCategoria* temp= principio3;
        bool existe= false;
        while(temp!=NULL){
            if(temp->NombreCategoria==categoria){
                temp->productos->IngresandoProducto(id, categoria, precio, nombre, desconocido);
                existe = true;
                break;
            }
            temp=temp->siguiente;
        }
        if(!existe){
            nuevo->productos->IngresandoProducto(id, categoria, precio, nombre, desconocido);
            temp->siguiente= nuevo;
        }
    }
}
```

Ingresando Producto:

```
void ListaProductos::IngresandoProducto(string id, string categoria, double precio, string nombre, string desconocido) {
    NodoProducto* nuevo = new NodoProducto();
    nuevo->id = id;
    nuevo->categoria = categoria;
    nuevo->precio = precio;
    nuevo->nombre = nombre;
    nuevo->desconocido = desconocido;
    if (principio1 == NULL) {
        principio1 = nuevo;
    } else {
        NodoProducto*temp = principio1;
        while (temp->siguiente != NULL) {
            temp = temp->siguiente;
        }
        temp->siguiente = nuevo;
    }
}
```

Ingresar Movimientos:

```
void ListaMovimientos::IngresandoMovimientos(string NombreMovimiento, Pila* movimientos) {
    NodoMovimiento*nuevo= new NodoMovimiento();
    nuevo->NombreMovimiento=NombreMovimiento;
    nuevo->movimientos=movimientos;
    if(principio5!=NULL){
        principio5=nuevo;
    }else{
        nuevo->siguiente=principio5;
        principio5=nuevo;
    }
}
```

MÉTODOS

Nombre	***Implementación***	***Utilización***
Insertar inicial	Implementado en lista, lista de listas, pila y cola.	Se inserta un elemento antes del *principal* y mueve el apuntador *principal* al nuevo elemento.
Insertar final	Implementado en lista y lista de listas.	Se inserta un elemento al final y si existe el apuntador último lo desplaza al nuevo elemento.
Obtener inicial	Implementado en lista, lista de listas, pila y cola.	Obtiene el primer elemento sin modificarlo.
Obtener final	Implementado en lista, lista de listas, pila y cola.	Obtiene el último elemento sin modificarlo.
Eliminar inicial	Implementado en lista, lista de listas, pila y cola.	Remueve el primer elemento y cambia de lugar el *principal* al siguiente.
Eliminar final	Implementado en lista, pila y cola.	Remueve el último elemento y cambia de lugar el nodo último al anterior.
Verificación vacía	Se implementa en todas las estructuras.	Verifica la existencia de algún elemento.
Limpiar	Se implementa en todas las estructuras.	Remueve todos los elementos de la lista.
	Implementado en la lista.	Navega hasta encontrar un elemento

Buscar		con propiedades que coincidan con el parámetro buscado y retorna.
Modificar	Implementado en la lista.	Navega hasta encontrar un elemento con propiedades que coincidan con el parámetro buscado y almacena la información en la nueva posición.