3 de oct. de 22 Estructura de datos

# Manual Técnico



Universidad de San Carlos de Guatemala Segundo semestre octubre de 2022

Carlos Eduardo Soto Marroquín 201902502

# INTRODUCCION

Dicho proyecto procesa información ingresada desde un archivo Json. La cual al ser cargada al sistema se almacena en diferentes tipos de estructuras de datos, se posee una interfaz grafica para la interacción con el usuario.

# Especificación técnica:

Los requerimientos mínimos para el posterior uso del programa son:

* Lenguajes implementados: C++.
* IDE usado: Ninguno
* Editor de código: visual studio code
* Sistema operativo: Windows 10 (64 bits)
* Interfaz gráfica: Python.
* Librerías utilizadas: Variadic Table, replace, sha256, graphviz, tkinter.

# Objetivos:

* Demostrar los conocimientos adquiridos sobre estructuras de datos no lineales como: matrices y árboles, poniéndolas en practica durante el desarrollo del juego batalla naval.
* Utilizar el lenguaje C++ para implementar estructuras de datos no lineales.
* Utilizar el lenguaje de programación Python para el desarrollo de interfaces graficas.
* Utilizar la herramienta Graphviz para graficar estructuras de datos lineales.
* Definir e implementar algoritmos de búsqueda, recorrido y eliminación.

# Alcances del proyecto:

La intención del proyecto es que el desarrollador de esta aplicación pueda definir e implementar las diferentes estructuras de datos no lineales como lo son matrices y árboles. Así mismo, la elaboración de algoritmos de búsqueda, eliminación y ordenamiento para el manejo de la información. Las salidas del programa es el reporte que se genera al graficar las diferentes estructuras de datos que se han desarrollado con la herramienta **Graphviz**.

# Lógica del programa:

Se tendrá un menú principal con el cual el usuario podrá interactuar con el. A continuación, se presentan las clases utilizadas para el proyecto:

* Funcionamiento del programa:
  + Main.
  + Menu.
  + User.
  + generacionImg
* Estructuras lineales:
  + LinkedListCategoria
  + DoublyLinkedListCircularUser
  + Cola tutorial
  + ArticleB
  + LinkedListBarco
  + PilaMov
  + ListaPilaMov
  + BlockChain
  + Hash Table
* Estructuras no lineales:
  + matriz
  + Merkle tree
  + AVL
  + Btree

# Funcionamiento del programa:

#### Main:

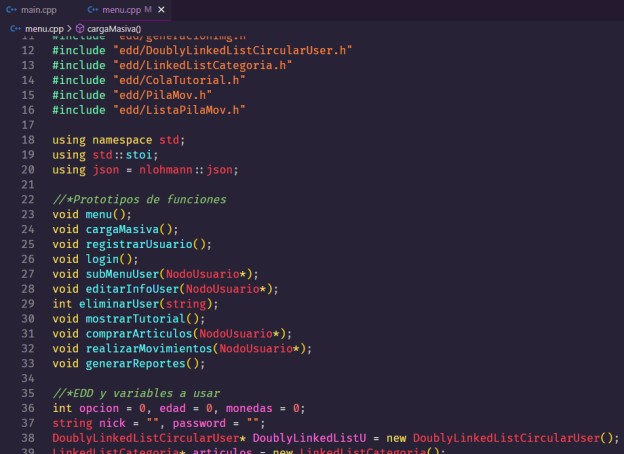
Es el encargado de iniciar el programa, llamando a la función menú, que es importada de otro archivo. A continuación, se muestra la clase:



#### Menú:

Elaboración propia, 2022

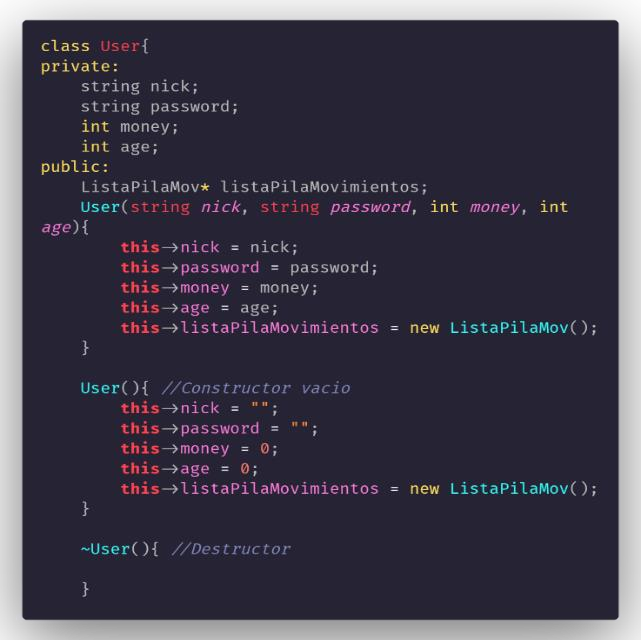
Es el archivo que contiene toda la lógica del programa, como los menús principales, los submenús, y los procedimientos que completan los menús principales y submenús.



#### User:

Elaboración propia, 2022

Dicha clase es la plantilla de todos los usuarios que se crearán contando con varios atributos como, el Nick, su contraseña, la cantidad de monedas y una edad, entre otros atributos.



#### generacionImg:

Elaboración propia, 2022

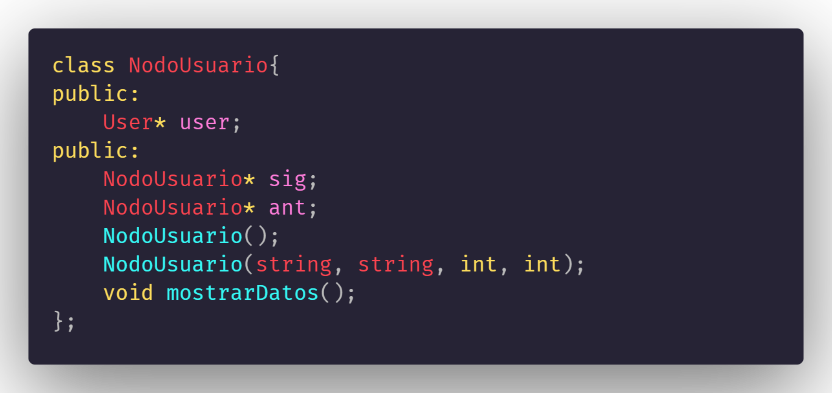
Posee varias funciones, genera un dot y lo pasa a una imagen, otra mueve los archivos de una carpeta a otra, y por último una función que abre la imagen.



Elaboración propia, 2022

Nodo Usuario:

Es el nodo que contendrá como atributo un objeto de tipo usuario, y que nos servirá como una pieza para la lista doblemente enlazada circular.



Elaboración propia, 2022

### DoublyLinkedListCircularUser:

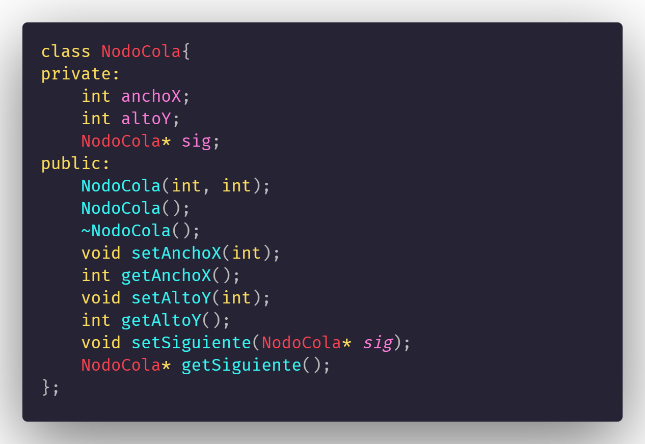
Es la lista doblemente enlazada circular, que almacenará cada nodo dentro de ella, y la cual posee varios métodos, como de búsqueda, ordenamiento, graficación, etc.



Elaboración propia, 2022

### Nodo cola:

Dicho nodo contendrá las coordenadas “X” e “Y”, o el ancho y alto, de los movimientos que se realicen en el tutorial del juego.



### Cola tutorial:

Elaboración propia, 2022

Es la estructura encargada de almacenar cada uno de los nodos cola, el cual a su vez contiene las coordenadas del juego, y el ancho y alto del tablero, este se usa para desplegar los movimientos del tutorial del juego.



Elaboración propia, 2022

### ArticleB:

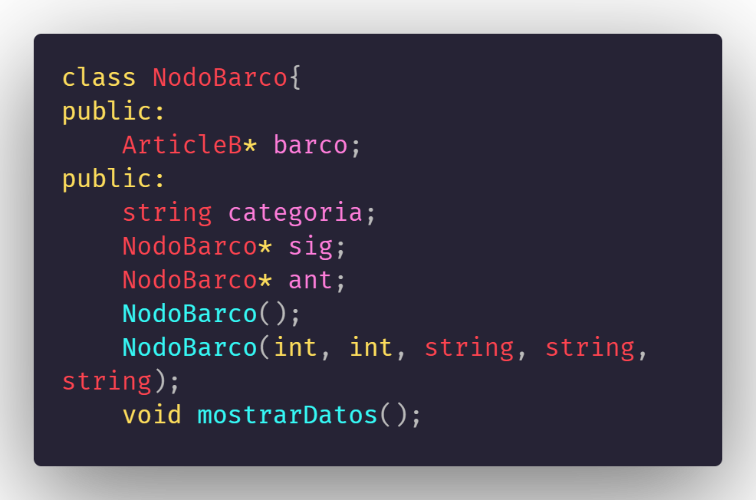
Dicha clase es la que contiene información de los barcos que se integrarán al juego.



### ~~Nodo Barco:~~

Elaboración propia, 2022

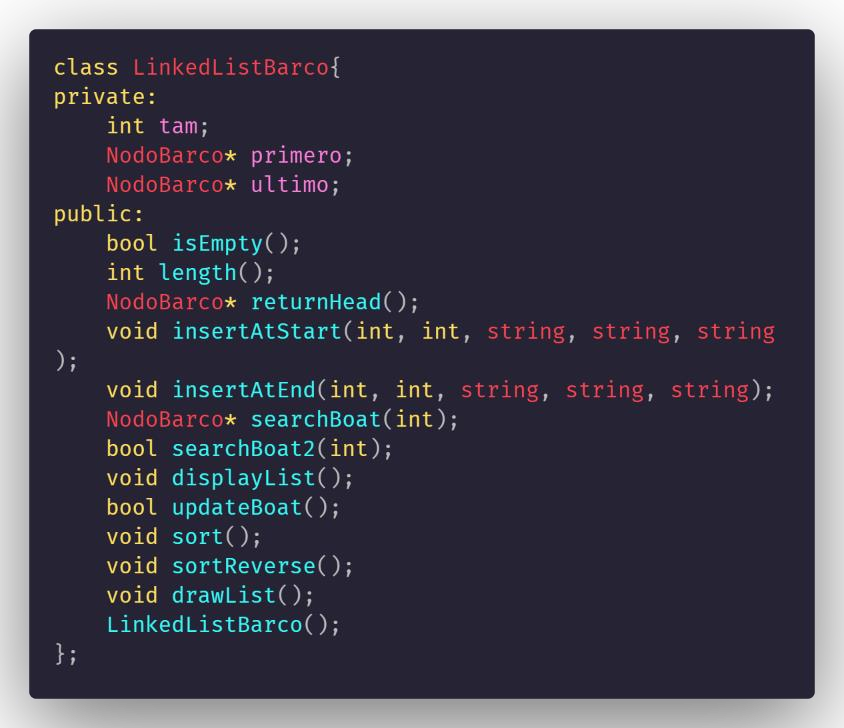
Almacena la información del objeto ArticleB (Barco), y la categoría del barco. Además de mostrar datos y sus apuntadores.



Elaboración propia, 2022

### ~~Linked List Barco:~~

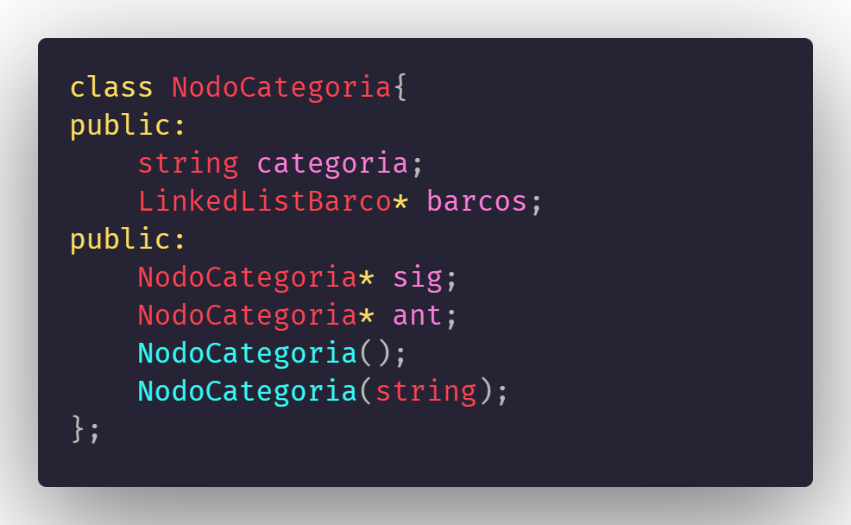
Es el encargado de llevar el control de los nodos barco, de ingresarlos, buscarlos, ordenarlos y mostrarlos.



### Nodo Categoria:

Elaboración propia, 2022

Es el encargado de almacenar la lista de barcos como atributo del nodo y el nombre de la categoría a la cual pertenecen.



Elaboración propia, 2022

### Linked List Categoria:

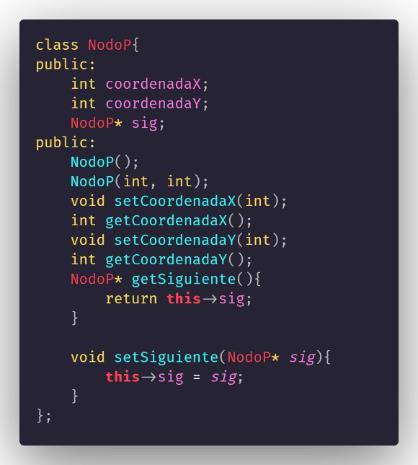
Lleva el control general de los nodos categoría, logrando así insertar los barcos en su categoría correspondiente, comprar un articulo de la tienda, y dibujar la lista de barcos.



### NodoP:

Elaboración propia, 2022

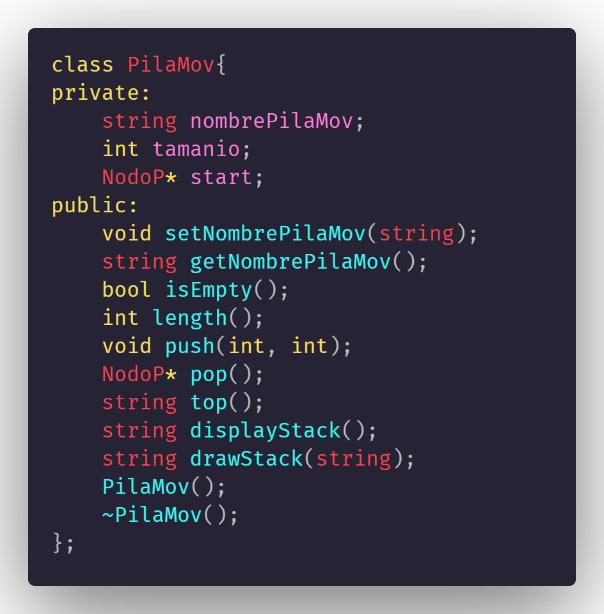
Es el nodo que lleva en sus atributos las coordenadas que el jugador implementa y que se van apilando en la pila.



Elaboración propia, 2022

### PilaMov:

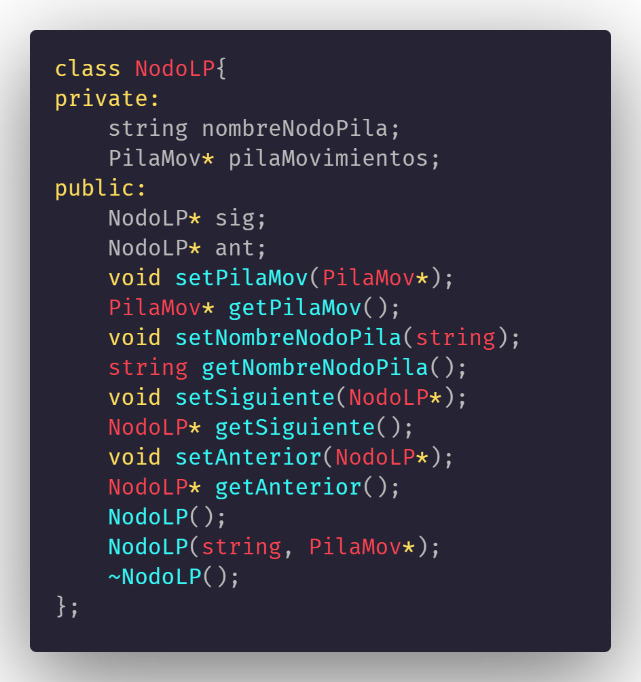
Lleva el control de los nodos pila, que llevan el registro de cada coordenada que ingrese el jugador o mejor mencionado como los movimientos que hace el jugador.



### NodoLP:

Elaboración propia, 2022

Este nodo llevará el control de una pila de movimientos, y el registro del nombre de la pila.



Elaboración propia, 2022

### ListaPilaMov:

Será la encargada de llevar el listado de pilas que tendrá la lista de usuarios.



Elaboración propia, 2022

**Estructuras lineales:**

#### Lista circular doblemente enlazada:

Dicha lista se utilizo para almacenar los usuarios de la aplicación. Posee varios métodos que se utilizaron para que fuera funcional, los cuales son los siguientes:

IsEmpty:

bool DoublyLinkedListCircularUser::isEmpty(){ return **this**->primero == NULL;

}

Se utiliza para verificar si la lista se encuentra vacía.

Length:

int DoublyLinkedListCircularUser::length(){ return **this**->tam;

}

Se utiliza para llevar el conteo de nodos en la lista.

JoinNodes:

void DoublyLinkedListCircularUser:: joinNodes(){ if(**this**->primero != NULL){

**this**->primero->ant = **this**->ultimo; **this**->ultimo->sig = **this**->primero;

}

}

Se utiliza para unir el primer nodo con el último. Completando

el enlace circular. InsertAtEnd:

void DoublyLinkedListCircularUser::insertAtEnd(string *nick*, string *password*, int *money*, int *age*){

NodoUsuario\* nuevo = new NodoUsuario(*nick*, *password*, *money*, *age*);

NodoUsuario\* aux = new NodoUsuario();

if(**this**->isEmpty()){

**this**->primero = **this**->ultimo = nuevo;

}else{

aux = **this**->ultimo;

**this**->ultimo = aux->sig = nuevo;

**this**->ultimo->ant = aux;

}

**this**-> joinNodes(); **this**->tam += 1;

aux = NULL;

}

Se usa dicho método para agregar al final el nodo. DisplayListSE:

void DoublyLinkedListCircularUser::displayListSE(){ if(**this**->isEmpty()){

cout<<"Lista vacia"<<endl; return;

}

VariadicTable<std::string, std::string , int, int> ut({"nick", "password", "monedas", "edad"});

NodoUsuario\* aux = new NodoUsuario(); aux = **this**->primero;

while (aux != NULL){

ut.addRow(aux->user->getNick(),aux->user-

>getPassword(), aux->user->getMoney(), aux->user->getAge()); aux = aux->sig;

if(aux == **this**->primero){ break;

}

}

ut.print(cout); aux = NULL; cout<<"\n";

}

Se utiliza para desplegar la lista de usuarios de inicio a fin.

DeleteNode:

bool DoublyLinkedListCircularUser::deleteNode(string *nick*){ NodoUsuario\* actual = new NodoUsuario();

actual = **this**->primero; bool check = false;

if(**this**->primero != NULL && check != true){ do{

if(actual->user->getNick() == *nick*){ check = true;

(actual->ant)->sig = actual->sig;

}

actual = actual->sig;

} while (actual != **this**->primero && check == false); if(!check) return false;

return true;

}else{

return false;

}

}

Se utiliza para eliminar un nodo en cualquier posición. SearchUser:

NodoUsuario\* DoublyLinkedListCircularUser::searchUser(string

*nick*, string *password*){

if(**this**->isEmpty()) return NULL; *//\*Retorna nulo en caso de que la lista este no contenga usuarios*

NodoUsuario\* actual = **this**->primero; while (actual != NULL){

if( (actual->user->getNick().compare(*nick*) == 0) && (actual->user->getPassword().compare(*password*) == 0)) return actual; *//\*retorna el nodo del usuario en caso de encontrarlo*

actual = actual->sig;

if(actual == **this**->primero) return NULL; *//\*Retorna nulo en caso de no encontrar al usuario*

}

}

Se utiliza para buscar un usuario, en caso de que exista retorna el nodo usuario, en caso contrario retorna NULL.

Sort:

void DoublyLinkedListCircularUser::sort(){ NodoUsuario\* aux = new NodoUsuario(); NodoUsuario\* actual = new NodoUsuario(); NodoUsuario\* temp = new NodoUsuario();

if(!**this**->isEmpty()){ actual = **this**->primero;

while(actual->sig != **this**->primero){ aux = actual->sig;

while(aux != **this**->primero){

if(aux->user->getAge() < actual->user-

>getAge()){

temp->user = actual->user; actual->user = aux->user; aux->user = temp->user;

}

aux = aux->sig;

}

actual = actual->sig;

}

}

return; *//\*Llegados aqui se ordeno todo de menor a mayor*

}

Se ordena la lista con el método burbuja, de menor a mayor.

SortReverse:

void DoublyLinkedListCircularUser::sortReverse(){ NodoUsuario\* aux = new NodoUsuario(); NodoUsuario\* actual = new NodoUsuario(); NodoUsuario\* temp = new NodoUsuario();

if(!**this**->isEmpty()){ actual = **this**->primero;

while(actual->sig != **this**->primero){ aux = actual->sig;

while(aux != **this**->primero){

if(aux->user->getAge() > actual->user-

>getAge()){

temp->user = actual->user; actual->user = aux->user; aux->user = temp->user;

}

aux = aux->sig;

}

actual = actual->sig;

}

}

return; *//\*Llegados aqui se ordeno todo de menor a mayor*

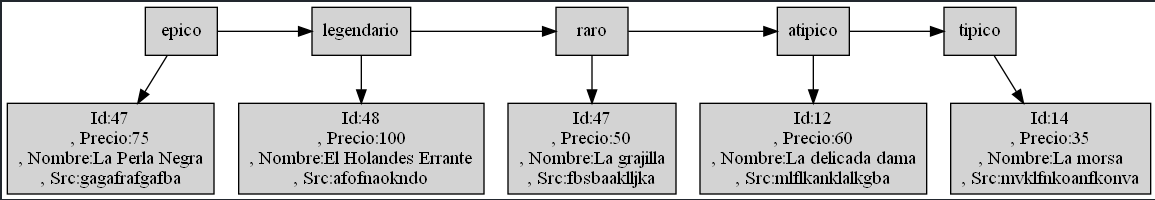
}

Se ordena la lista con el método burbuja, de mayor a menor.

#### Lista de Listas:

Esta estructura de datos fue utilizada para los artículos, ya que se van dividiendo por categorías, y según su categoría es como se irán almacenando los datos de los barcos. Dicha estructura

esta compuesta de dos listas y dos nodos, se tendrá una lista principal y otra secundaria, al igual que los nodos uno principal y otro secundario.



#### Pila:

Se utilizo la pila para almacenar los movimientos que realiza el jugador, al mismo tiempo que se pueden retirar de la pila en caso de ser necesario.

Push:

Es usado para ingresar los datos por la cabeza de la pila.

void PilaMov::push(int *coordenadaX*, int *coordenadaY*){ NodoP\* nuevo = new NodoP(*coordenadaX*, *coordenadaY*); if(**this**->isEmpty()){

**this**->start = nuevo;

}else{

nuevo->setSiguiente(**this**->start);

**this**->start = nuevo;

}

**this**->tamanio += 1;

}

Pop:

Es usado para retornar los datos por la cabeza de la pila.

NodoP\* PilaMov::pop(){ NodoP\* tope= NULL; if(!**this**->isEmpty()){

tope = **this**->start;

**this**->start = **this**->start->sig;

**this**->tamanio -=1;

}

return tope;

}

Top:

Sirve para verificar que valores se tienen en la cima de la pila.

string PilaMov::top(){ if(!**this**->isEmpty()){

return "X: " + to\_string(**this**->start->coordenadaX) + ", Y: " + to\_string(**this**->start->coordenadaY);

}else{

return "pila vacia";

}

}

#### Cola:

Se utilizo la cola para almacenar los movimientos que se van cargando al momento de cargar el tutorial al juego.

Enqueue:

Sirve para poner los datos al final de la cola.

void ColaTutorial::enqueue(int *anchoX*, int *altoY*){ NodoCola\* nuevo = new NodoCola(*anchoX*, *altoY*); if(**this**->isEmpty()){

**this**->frente = nuevo;

}else{

**this**->final->setSiguiente(nuevo);

}

**this**->final = nuevo;

**this**->tamanio += 1;

}

Dequeue:

Sirve para retirar los datos al inicio de la cola.

NodoCola\* ColaTutorial::dequeue(){ NodoCola\* nuevo = **this**->frente;

**this**->frente = **this**->frente->getSiguiente();

if(**this**->isEmpty()){

**this**->final = NULL;

}

**this**->tamanio -= 1; return nuevo;

}

In front:

Sirve para saber que datos poseemos al frente de la cola.

string ColaTutorial::in\_front(){

return "anchoX:" + to\_string(**this**->frente->getAnchoX()) + ", altoY:" + to\_string(**this**->frente->getAltoY());

}

#### Lista de Pilas:

Nos sirve para llevar el conteo de jugadas que ha realizado el jugador.

Insertar al final:

Nos sirve para insertar el nombre de la pila y la pila al final de la lista.

void ListaPilaMov::insertarAlFinal(string *nombreNodoPila*, PilaMov\* *pilaMovimientos*){

NodoLP\* nuevo = new NodoLP(*nombreNodoPila*,

*pilaMovimientos*);

NodoLP\* aux = new NodoLP();

if(**this**->estaVacio()){

**this**->primero = **this**->ultimo = nuevo;

}else{

aux = **this**->ultimo;

**this**->ultimo = aux->sig = nuevo;

**this**->ultimo->setAnterior(aux);

}

**this**->tamanio +=1;

}

Desplegar Lista:

Nos sirve para recorre la lista de pilas e ir viendo cada una de las pilas de movimiento que ha realizado el jugador.

void ListaPilaMov::desplegarLista(){ string cadena = "", pilas = ""; if(**this**->estaVacio()){

cout<<"La lista esta vacia"<<endl; return;

}

NodoLP\* aux = new NodoLP(); aux = **this**->primero;

while (aux != NULL){

cout<<"Pila: "<<aux->getNombreNodoPila()<<endl; cout<<aux->getPilaMov()->displayStack()<<endl;

*// cout<<aux->getPilaMov()->drawStack(aux-*

*>getNombreNodoPila());*

aux = aux->getSiguiente();

}}

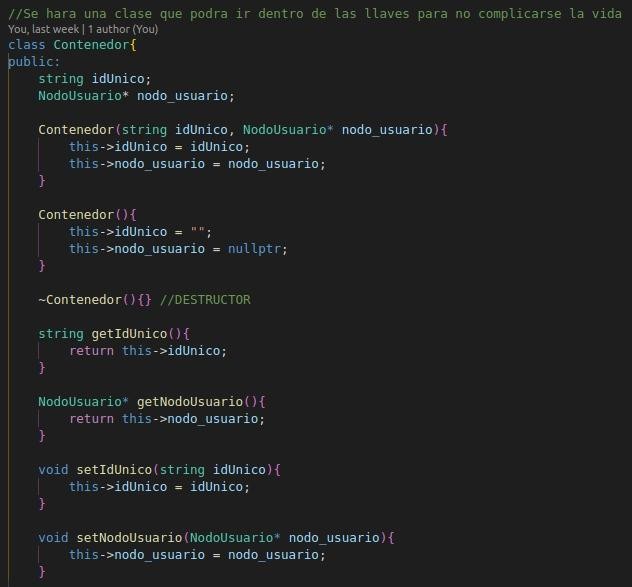
## Estructuras no lineales:

#### Árbol B:

El árbol B fue utilizado para almacenar la referencia de cada nodo de la lista de usuarios.

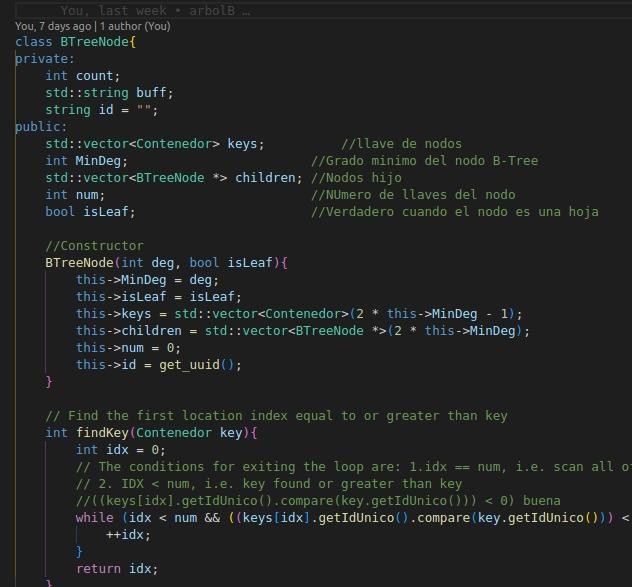
Contenedor:

Es la clase que engloba todo de nuestro Árbol B.



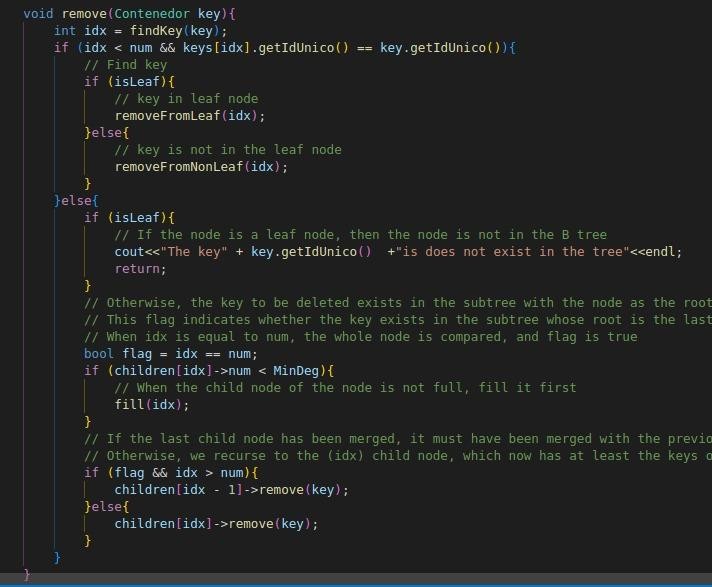
BTreeNode:

El nodo de nuestro árbol b es el encargado de poder almacenar cada uno de los usuarios encontrados en la lista:



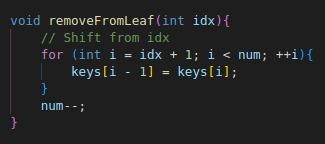
remove:

es el encargado de eliminar un nodo especificado por el usuario de nuestro árbol b:



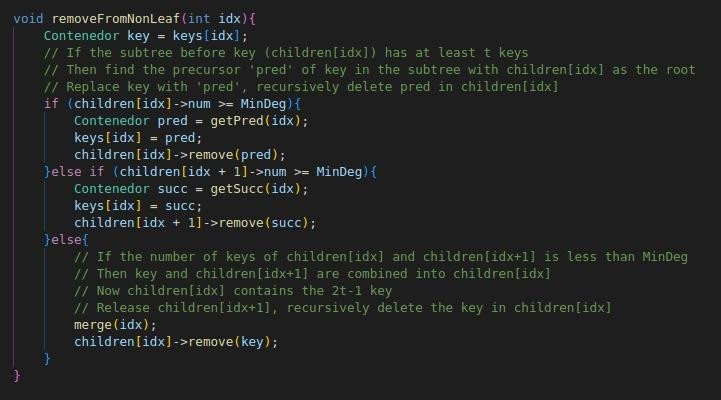
removeFromLeaf:

A la hora de eliminar un usuario del árbol b tambien lo eliminamos de la hoja de dicho árbol:



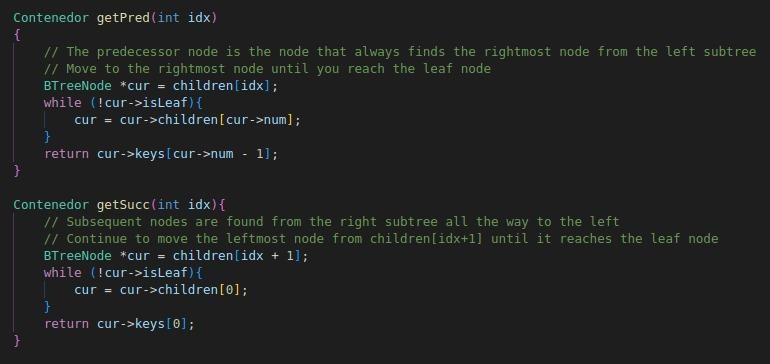
removeFromNonLeaf:

Eliminamos el nodo, pero no de la hoja del árbol b:



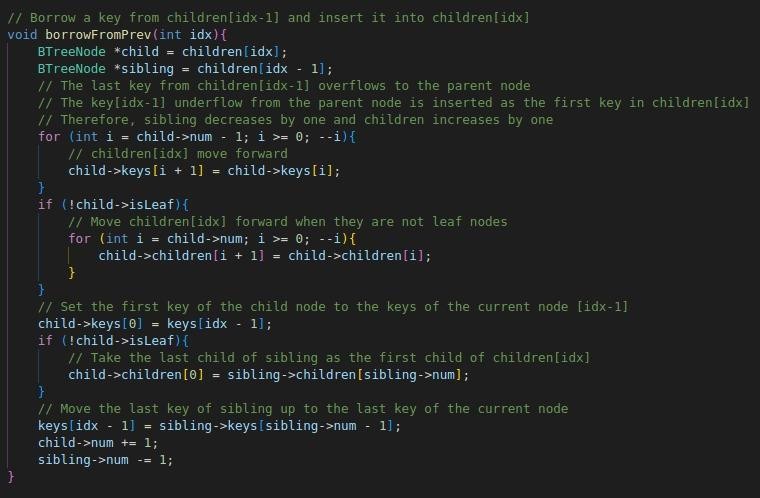
getPred y getSucc:

getPred encargado de encontrar el nodo mas a la derecha del subárbol de la izquierda y se mueve hacia el nodo más a la derecha hasta llegar al nodo hoja. getSucc es el encargado de que encuentre desde el subárbol de la derecha hasta el de la izquierda y continuar moviendo el nodo más a la izquierda de los hijos hasta llegar al nodo hoja:



borrowFromPrev:

Llena una llave del nodo anterior:



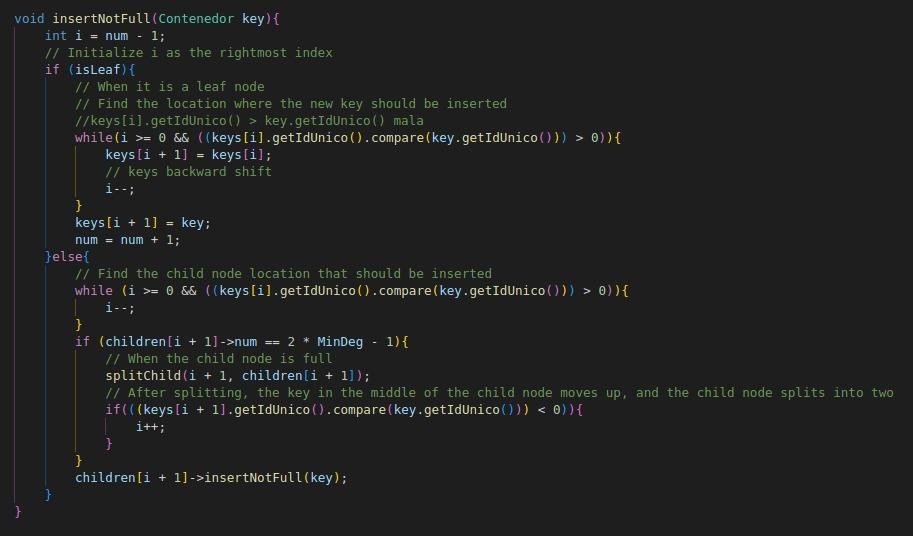
borrowFromNext:

Rellena una llave del siguiente nodo:



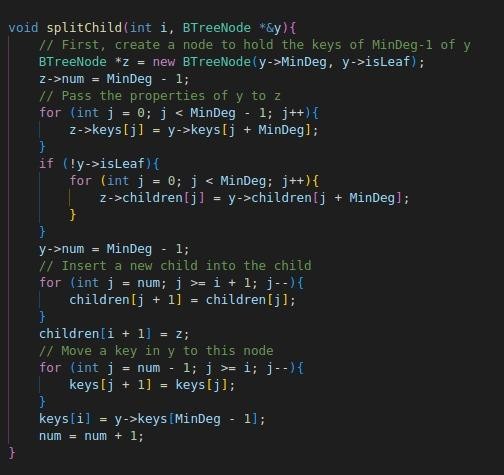
insertNotFull:

Cuando se trata de un nodo hoja encuentra el lugar donde debe insertarse la nueva llave y luego encuentra la ubicación del nodo hijo que debe ser insertado:



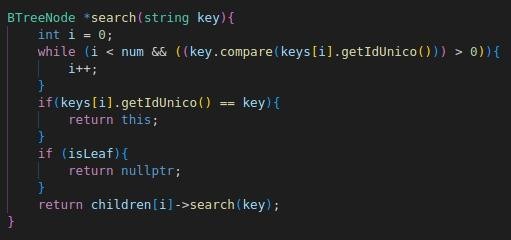
splitChild:

Crea un nodo con la llave obtenida. Luego pasa las propiedades de la variable y a la z e inserta un nuevo nodo hijo y por ultimo mueve la llave de y al nuevo nodo hijo creado:



Search:

Clase encargada de buscar un dato dentro del árbol B:



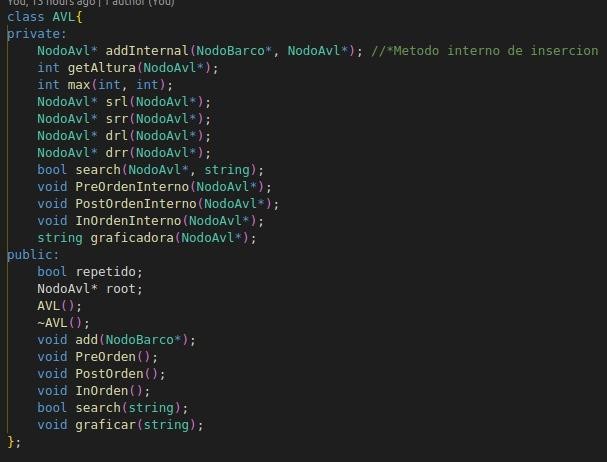
toDot:

Clase encargada para generar el dot de nuestro árbol b y poder graficarlo con graphviz:



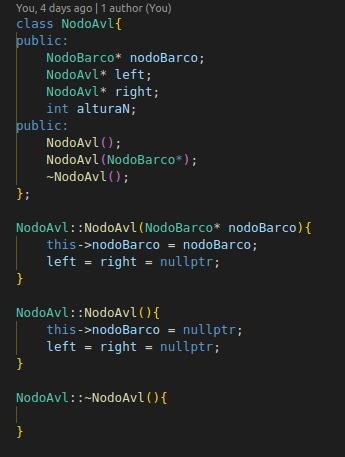
#### Árbol AVL:

El árbol AVL es el encargado de almacenar los barcos que cada usuario se compra en su cuenta:



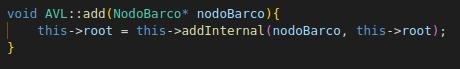
nodoAvl:

Encargado de almacenar los barcos usados por cada compra realizada:



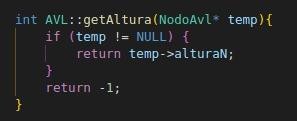
Add:

Es el encargado de recibir el dato y enviarlo a addInternal que es el encargado de poder almacenar y ordenar los nodos dentro del árbol AVL:



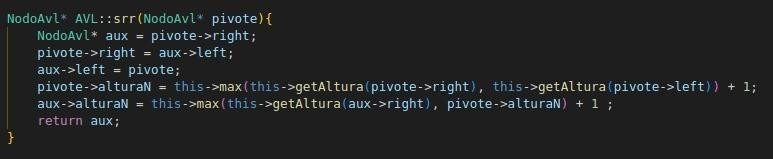
getAltura:

Función encargada de obtener la altura de dicho árbol:



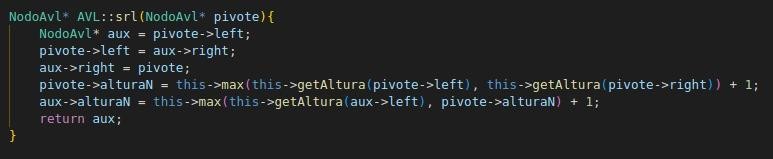
Srr:

Encargada de realizar la rotación simple por la derecha dentro del árbol AVL:

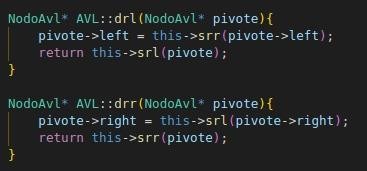


Srl:

Realiza la rotación simple por la izquierda dentro del árbol AVL:

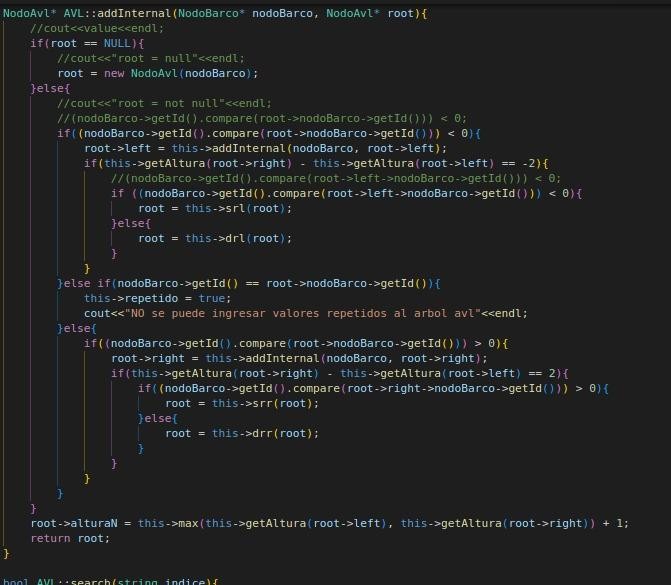


Rotaciones dobles por la izquierda y derecha:



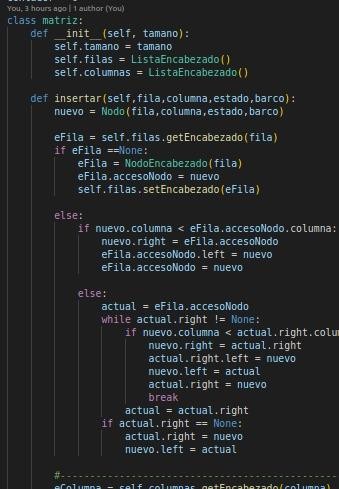
addInternal:

Realiza la inserción de cada uno de los datos ingresados/obtenidos al árbol AVL:



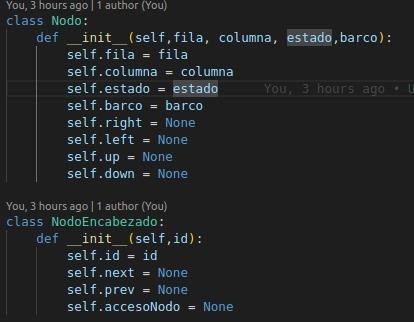
#### Matriz Dispersa:

Utilizada para la implementación e impresión del tablero de juego:



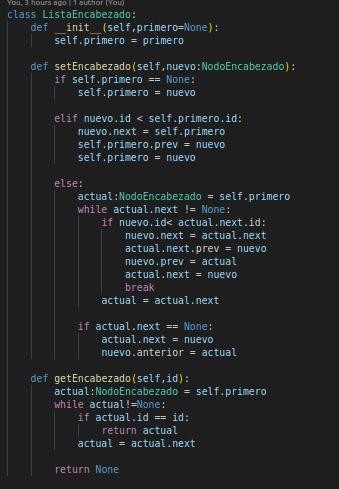
Nodo y NodoEncabezado:

Encargados de almacenar cada uno de los barcos y ser utilizados para su inserción dentro del tablero:



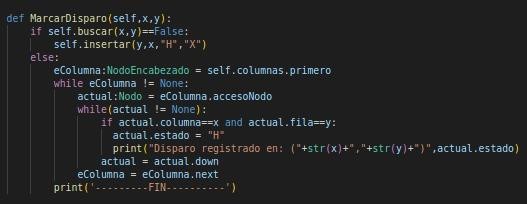
ListaEncabezado:

Clase encargada de poder obtener los números de fila y columna para la inserción de datos en su coordenada correcta:



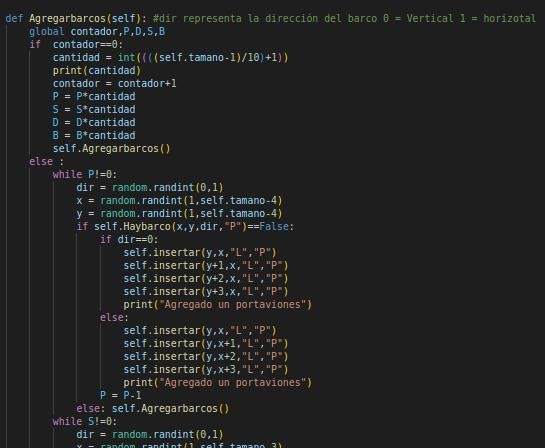
MarcarDisparo:

Muestra las coordenadas hacia donde fue enviado el disparo:



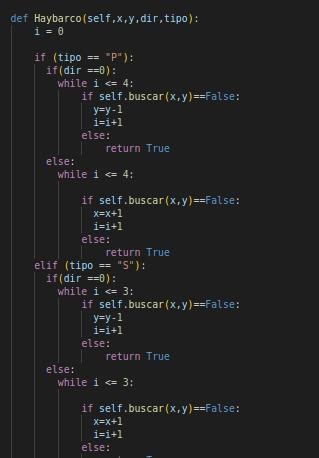
Agregarbarcos:

Encargada de insertar cada uno de los barcos en su coordenada correcta dentro del tablero:



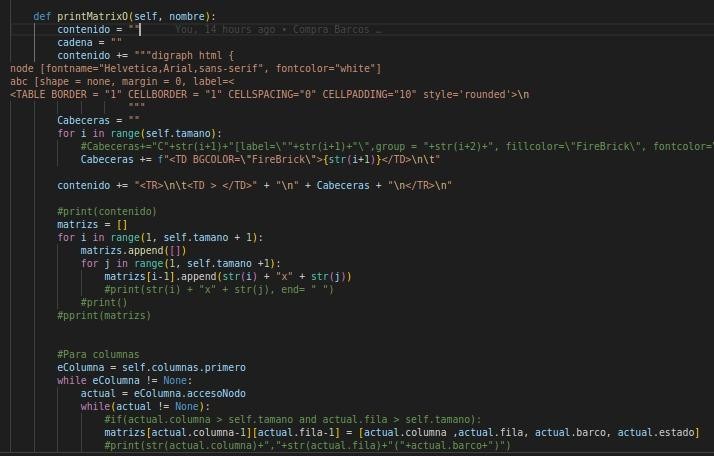
Haybarco:

Identifica el tipo de barco que se encuentra en el tablero:



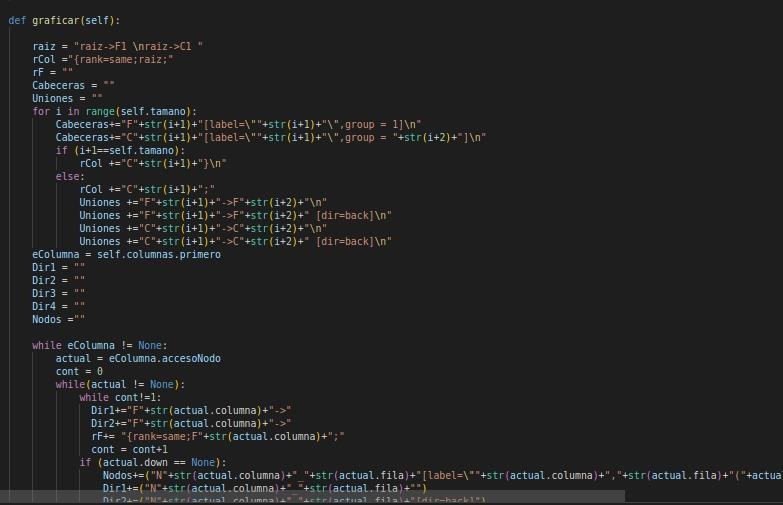
printMatrix0:

Es la encargada de imprimir la matriz como tablero con sus numero de fila y columna al igual que cada uno de los barcos en sus coordenadas correctas:



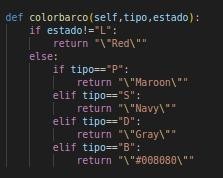
Graficar:

Encargada de generar el la cadena del dot de la grafica del tablero:



Colorbarco:

Establece el color de cada uno de los barcos según su rareza:

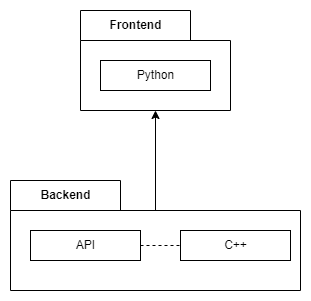


Hash Table:

Utilizada para almacenar las skins de barcos en un carrito de compras:

ghd

# ANEXOS:



**Diagrama de componentes.** Elaboración propia 2022.

# CONCLUSIONES

Mediante el desarrollo de este proyecto, se ha logrado desarrollar, entender e implementar cada una de las funciones, estructura de datos, y métodos previamente solicitados, para poder concluir que la estructuración de datos es bastante importante para el manejo de información de objetos como tal.

Conocimos, identificamos y aplicamos las estructuras no lineales en la solución de problemas del mundo real. Al igual conocimos un poco acerca de las teorías de esta unidad así como el aprendizaje utilizando el lenguaje de programación C++ y Python para la implementación de las estructuras no lineales.

Se pudo implementar las operaciones básicas (insertar, eliminar, buscar) en un árbol b, así como también el uso de Graphviz para el grafico de cada estructura.