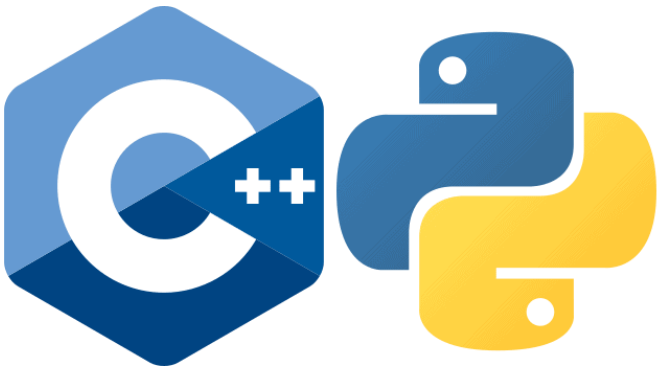
Laboratorio de Estructuras de Datos



# PROYECTO EDD – FASE 2

**MANUAL TECNICO**

FECHA: 03/10/2022

Juan Josue Zuleta Beb
Carné: 202006353

## Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación, deseñada a través de código de estructura de datos con paradigma orientado a objetos. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

## Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

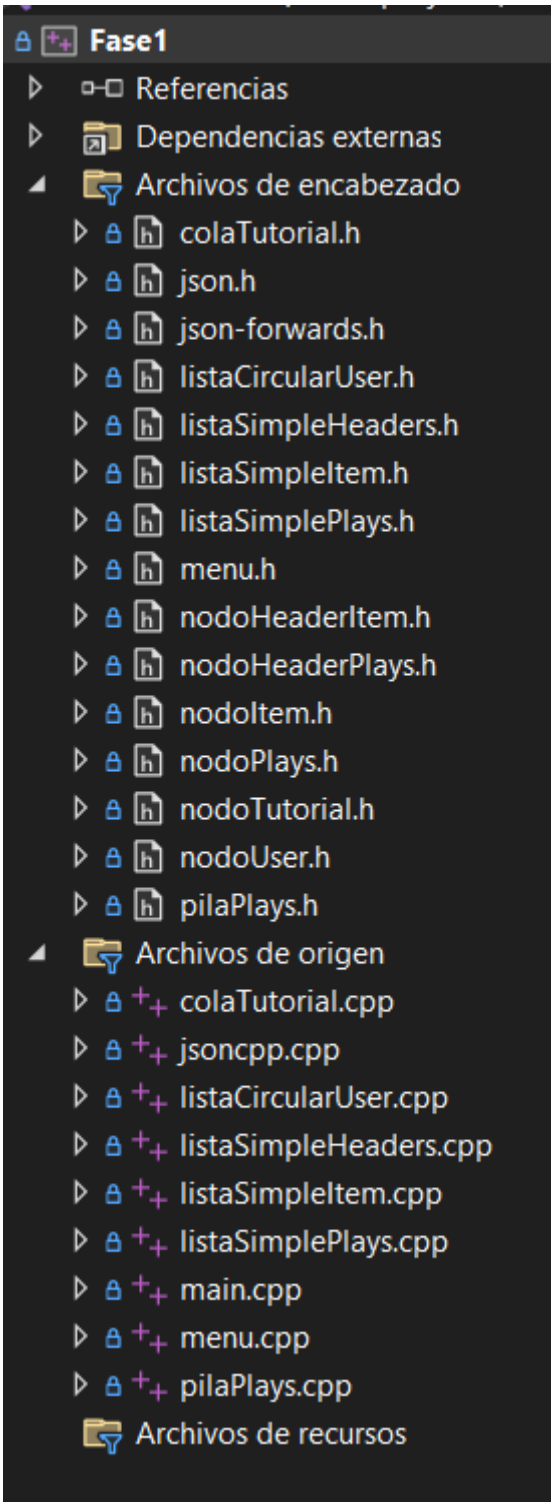
## Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

**Estructura raíz:**

El proyecto tiene la siguiente estructura de directorios:



## DESCRIPSION DE LOS METODOS Y CLASES:

Los métodos y clases utilizados para este programa fueron pensados y diseñados específicamente para el desarrollo e implementación de este, siguiendo rigurosamente los requerimientos de diseño inicialmente propuestos en el manual de requerimientos.

### La clase main.cpp:

Contiene el método principal de nuestra aplicación, esta clase se encarga de inicializar los objetos que usaremos dentro del programa a lo largo de toda la ejecución. Ayudando así a la persistencia de datos en cualquier área del programa.

Esta clase es la que se encarga de generar la conexión con el lenguaje de programación Python, en el cual se consumirá por medio de una interfaz grafica de usuario, las estructuras definidas en C++.

### Librería implementada para la conexión con Python

Para realizar la conexión con el lenguaje Python se ultilizo la librería Crow, la es un marco C++ para crear servicios web HTTP o Websocket. Utiliza un enrutamiento similar a Python's Flask, lo que lo hace fácil de usar. También es extremadamente rápido, superando múltiples marcos C++ existentes, así como marcos que no son C++.

Puerto de Conexión Local:

La API estará alojada en el puerto local '5000': http://localhost:5000

```
app.port(5000).multithreaded().run();
```

Método de Estado de la Aplicación:

Por medio de un método "request" desde Python podemos ejecutar este enlace a la API de C++ la cual tendrá un response textual cuando el servidor este activo y un error con código 504 cuando el servidor este inactivo.

```
crow::SimpleApp app;
CROW_ROUTE(app, "/")
    ([] {
    return crow::response("API -> UP!");
        });
```

Método de inicio de sesión:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y retornar un estado, el cual dependerá si el usuario existe y si no existe no podrá logearse, también verificara si la contraseña ingresada pertenece al usuario.

```
CROW_ROUTE(app, "/login")
    ([&listaUsers](const crow::request& req)
        {
            auto x = crow::json::load(req.body);
            if (!x)
                return crow::response(400);
            string nick = x["nick"].s();
            string pass = x["password"].s();

            nodoUser* usuario = listaUsers.searchUsers(nick);

            if (usuario == nullptr) {
                return crow::response("Usuario no encontrado");
            }

            if (usuario->getPassword() == pass) {
                return crow::response(usuario->getNick());
            }

            return crow::response("incorrecta");

        });
```

Método de Registro:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y crear u nuevo nodo para generar un nuevo usuario y asignarle los atributos de nodo usuario.

```
CROW_ROUTE(app, "/sign_up")
    .methods("POST"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);

        nodoUser* nuevo;

        string nick = x["nick"].s();
        string pass = x["password"].s();
        string age = x["edad"].s();

        if (listaUsers.searchExist(nick) != true) {
            nuevo = new nodoUser();

            nuevo->setNick(nick);
            nuevo->setPassword(pass);
            nuevo->setAge(stoi(age));

            listaUsers.addToEnd(nuevo);
            return crow::response("registrado");
        }

        return crow::response("repetido");

    });
```

Método de Editar información:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y edita por medio del método PUT la información actual de usuario.

```
CROW_ROUTE(app, "/edit")
    .methods("PUT"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);
        string name = x["name"].s();
        string nick = x["nick"].s();
        string pass = x["password"].s();
        string age = x["edad"].s();

        nodoUser* usuario = listaUsers.searchUsers(name);

        if (nick != "") {
            usuario->setNick(nick);
        }
        if(age != "") {
            usuario->setAge(stoi(age));
        }
        if (pass != "") {
            usuario->setPassword(pass);
        }

        return crow::response("datos actualizados");

    });
```

Método de Eliminar Usuario:

Este método se encarga de realizar un request hacia la lista doblemente enlazada de usuarios y edita por medio del método DELETE la información actual de usuario y elimina el nodo para eliminar la información del usuario.

```
CROW_ROUTE(app, "/delete")
    .methods("DELETE"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);
        string nick = x["nick"].s();
        string pass = x["password"].s();

        nodoUser* usuario = listaUsers.searchUsers(nick);

        if (listaUsers.deleteUser(usuario->getNick(), usuario->getPassword()) == true) {
            return crow::response("Usuario Eliminado!");
        }

        return crow::response("Eliminacion imcompleta!");

    });
```

Método de Apilar Jugadas:

Este método se encarga de realizar un request hacia la pila de jugadas y por medio de la conexión con la lista de usuarios y la pila se guarda el nombre del usuario y las jugadas realizadas en forma de pila.

```
CROW_ROUTE(app, "/up_plays")
    .methods("POST"_method)([&listaUsers,&newStack](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);

        string nick = x["nick"].s();
        string posx = x["x"].s();
        string posy = x["y"].s();

        nodoUser* usuario = listaUsers.searchUsers(nick);

        if (usuario != NULL) {

            newStack->doGamePlays(usuario, stoi(posx), stoi(posy));
            return crow::response("Juagada aniadida");

        }

        return crow::response("Usuario no encontrado");

    });
```

Método de Actualizar Monedas:

Este método se encarga de realizar un request hacia la lista circular de usuarios y actualiza el atributo monedas, del usurario por medio del método PUT y de esta forma interactúa la interfaz grafica con la base de datos en C++.

```
CROW_ROUTE(app, "/up_coins")
    .methods("PUT"_method)([&listaUsers](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);
        string nick = x["nick"].s();
        string coins = x["monedas"].s();

        nodoUser* usuario = listaUsers.searchUsers(nick);

        if (coins != "") {
            usuario->setCoins(stoi(coins));
        }

        return crow::response("Monedas actualizadas");

    });
```

Método de Carga Masiva:

Este método se encarga de realizar un request hacia la las lista circular de usuarios, hacia la cola de movimientos, hacia la lista de artículos y hacia la lista de listas de Artículos, y llama el método de carga masiva de cada uno para poder almacenar en memoria lo obtenido del archivo en formato Json.

```
CROW_ROUTE(app, "/masiva")
    .methods("POST"_method)([&listaUsers, &structList, &listItem, &queue](const crow::request& req)
    {
        auto x = crow::json::load(req.body);
        if (!x)
            return crow::response(400);

        string ruta = x["ruta"].s();

        listaUsers.loadFile(ruta);
        structList.loadFile(ruta);
        listItem.loadFile(ruta);
        queue.loadFile(ruta);

        return crow::response("Informacion Cargada!");
    });
```

Método de reporte de usuarios:

Este método se encarga de realizar un request hacia la lista circular de usuarios, y llama a los métodos de ordenamiento y retorna la lista ordenada en forma ascendente y descendente, para su posterior visualización en formato HTML.

```
CROW_ROUTE(app, "/usersUp")
    ([&listaUsers]()
        {
            std::vector<crow::json::wvalue> temp = listaUsers.bubbleSortUP();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });


CROW_ROUTE(app, "/usersDw")
    ([&listaUsers]()
        {
            std::vector<crow::json::wvalue> temp = listaUsers.bubbleSortDW();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });
```

Método de reporte de Artículos:

Este método se encarga de realizar un request hacia la lista circular de Items, y llama a los métodos de ordenamiento y retorna la lista ordenada en forma ascendente y descendente, para su posterior visualización en formato HTML.

```
CROW_ROUTE(app, "/itemsUp")
    ([&listItem]()
        {
            std::vector<crow::json::wvalue> temp = listItem.bubbleSortUP();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });


CROW_ROUTE(app, "/itemsDw")
    ([&listItem]()
        {
            std::vector<crow::json::wvalue> temp = listItem.bubbleSortDW();
            crow::json::wvalue final = std::move(temp);
            return crow::response(std::move(final));
        });
```

**La clase Main.py**

Esta es la clase principal en el lenguaje de programación Python, y se encargara del manejo de la interfaz grafica así como de la matriz dispersa para que el usuario pueda visualizar las operaciones que realiza dentro de C++ por medio de la interfaz de la librería TKinter de Python.

La variable global base_url:

Esta variable representa el puerto local en el cual esta alojado nuestra API en C++, esta es la URL por medio de la cual interactuaremos en todo momento con nuestra base de datos y con las estructuras ya definidas.

```
base_url = "http://localhost:5000"
```

**Matriz Dispersa:**

Para la implementación de la estructura de la matriz dispersa se implementaron distintas estructuras de datos, tales como un nodo y una lista doblemente enlazada para las cabeceras y también un nodo interno. Tales métodos se describen a continuación como parte de la estructura principal de la matriz dispersa.

Nodo Encabezado:

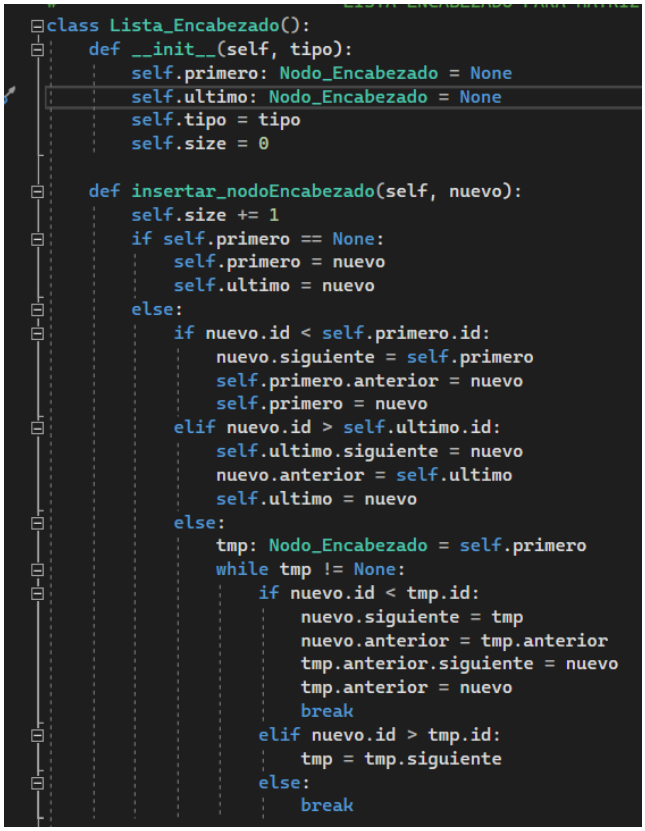
Este nodo se encarga de Ocupar las posiciones en memoria de las cabeceras de la matriz disperasa.

```
class Nodo_Encabezado():
    def __init__(self, id):
        self.id: int = int(id)
        self.siguiente = None
        self.anterior = None

        self.acceso = None
```

Lista Encabezado:

Este nodo se encarga de asignar las posiciones de los nodos encabezado para que se ordenen y se generen según el tamaño y el requerimiento de espacio de la matriz dispersa.

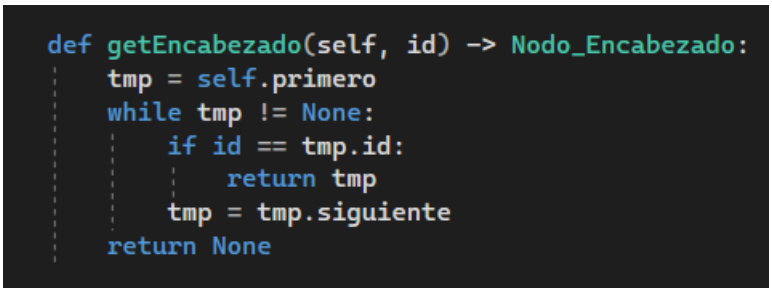


```
class Lista_Encabezado():
    def __init__(self, tipo):
        self.primero: Nodo_Encabezado = None
        self.ultimo: Nodo_Encabezado = None
        self.tipo = tipo
        self.size = 0

    def insertar_nodoEncabezado(self, nuevo):
        self.size += 1
        if self.primero == None:
            self.primero = nuevo
            self.ultimo = nuevo
        else:
            if nuevo.id < self.primero.id:
                nuevo.siguiente = self.primero
                self.primero.anterior = nuevo
                self.primero = nuevo
            elif nuevo.id > self.ultimo.id:
                self.ultimo.siguiente = nuevo
                nuevo.anterior = self.ultimo
                self.ultimo = nuevo
            else:
                tmp: Nodo_Encabezado = self.primero
                while tmp != None:
                    if nuevo.id < tmp.id:
                        nuevo.siguiente = tmp
                        nuevo.anterior = tmp.anterior
                        tmp.anterior.siguiente = nuevo
                        tmp.anterior = nuevo
                        break
                    elif nuevo.id > tmp.id:
                        tmp = tmp.siguiente
                    else:
                        break
```

Método obtener Encabezado:

Este método facilita la consulta de Nodo Encabezado, si existe el Nodo lo retorna, y si no existe se crea uno nuevo, este es esencial para la creación correcta de la matriz dispersa



```
def getEncabezado(self, id) -> Nodo_Encabezado:
    tmp = self.primero
    while tmp != None:
        if id == tmp.id:
            return tmp
        tmp = tmp.siguiente
    return None
```

Nodo Interno:

Este método se encarga de obtener la asignación de un valor en memoria de un nodo interno de la matriz dispersa, estos nodos son los que se encargaran de sostener el tipo de dato almacena y las principales características de este y por medio del cual implementaremos la logia del tablero y la ubicación y características de los barcos.

```
class Nodo_Interno():
    def __init__(self, x, y, caracter, color):
        self.caracter = caracter
        self.coordenadaX = int(x)
        self.coordenadaY = int(y)
        self.color = color

        self.arriba = None
        self.abajo = None
        self.derecha = None
        self.izquierda = None
```

Matriz Dispersa:

Esta es la clase principal de la matriz dispersa y se encarga de manejar toda la lógica de recorrido y asignación de nodos cabecera, listas cabeceras y nodos internos de tal forma que se genere correctamente la matriz dispersa y todas las características que esta conlleva. La matriz dispersa es capa de administrar la memoria dinámicamente de forma visual, de modo que tanto los nodos cabeceros como los nodos internos se crea únicamente a medida que son solicitados y esos son asignados en su posición únicamente cuando la posición solicitada no existe.

```
class MatrizDispersa():
    def __init__(self, capa):
        self.capa = capa
        self.filas = Lista_Encabezado('fila')
        self.columnas = Lista_Encabezado('columna')
```

Método de Tamaño:

Este método se encarga de definir el tamaño que tendrá la matriz dispersa por medio de las listas de Cabeceras debido a que según se creen los nodos cabeceros de crearan también los espacios en memoria que definen los límites de la matriz.

```python
def size(self,size):
    global matrix_size
    matrix_size = size

    for i in range(1,matrix_size+1):
        nodo_X = self.filas.getEncabezado(i)
        nodo_Y = self.columnas.getEncabezado(i)

        if nodo_X == None:
            nodo_X = Nodo_Encabezado(i)
            self.filas.insertar_nodoEncabezado(nodo_X)

        if nodo_Y == None:
            nodo_Y = Nodo_Encabezado(i)
            self.columnas.insertar_nodoEncabezado(nodo_Y)
```

Método de Inserción:

Este método se encarga de definir la inserción del nodo interno y se almacena como nodo interno de ambas listas cabeceras, las cuales son filas y columnas.

```python
def insert(self, pos_x, pos_y, caracter, color):
    nuevo = Nodo_Interno(pos_x, pos_y, caracter, color)
    nodo_X = self.filas.getEncabezado(pos_x)
    nodo_Y = self.columnas.getEncabezado(pos_y)

    if nodo_X == None:
        nodo_X = Nodo_Encabezado(pos_x)
        self.filas.insertar_nodoEncabezado(nodo_X)

    if nodo_Y == None:
        nodo_Y = Nodo_Encabezado(pos_y)
        self.columnas.insertar_nodoEncabezado(nodo_Y)
```

Este método deriva en la inserción por fila y la inserción por columna las cuales se puede verificar como sigue:

```python
if nodo_X.acceso == None:
    nodo_X.acceso = nuevo
else:
    if nuevo.coordenadaY < nodo_X.acceso.coordenadaY:
        nuevo.derecha = nodo_X.acceso
        nodo_X.acceso.izquierda = nuevo
        nodo_X.acceso = nuevo
    else:
        tmp : Nodo_Interno = nodo_X.acceso
        while tmp != None:
            if nuevo.coordenadaY < tmp.coordenadaY:
                nuevo.derecha = tmp
                nuevo.izquierda = tmp.izquierda
                tmp.izquierda.derecha = nuevo
                tmp.izquierda = nuevo
                break;
            elif nuevo.coordenadaX == tmp.coordenadaX and nuevo.coordenadaY == tmp.coordenadaY:
                break;
            else:
                if tmp.derecha == None:
                    tmp.derecha = nuevo
                    nuevo.izquierda = tmp
                    break;
                else:
                    tmp = tmp.derecha
```

```
    if nodo_Y.acceso == None:
        nodo_Y.acceso = nuevo
    else:
        if nuevo.coordenadaX < nodo_Y.acceso.coordenadaX:
            nuevo.abajo = nodo_Y.acceso
            nodo_Y.acceso.arriba = nuevo
            nodo_Y.acceso = nuevo
        else:
            tmp2 : Nodo_Interno = nodo_Y.acceso
            while tmp2 != None:
                if nuevo.coordenadaX < tmp2.coordenadaX:
                    nuevo.abajo = tmp2
                    nuevo.arriba = tmp2.arriba
                    tmp2.arriba.abajo = nuevo
                    tmp2.arriba = nuevo
                    break;
                elif nuevo.coordenadaX == tmp2.coordenadaX and nuevo.coordenadaY == tmp2.coordenadaY:
                    break;
                else:
                    if tmp2.abajo == None:
                        tmp2.abajo = nuevo
                        nuevo.arriba = tmp2
                        break
                    else:
                        tmp2 = tmp2.abajo
```

Método de Inserción automática de barcos:

Este conjunto de métodos se encarga de cuantificar la cantidad de barcos necesarios según el tamaño, este mismo método se repite en cada tipo de barco, ya sea la porta avión, el submarino, el destructor y el buque. El método Genera un numero aleatorio en cada coordenada, de tal forma que se consigue una coordenada de X y Y aleatoria desde la cual se empieza a evaluar las posibles posiciones donde se podrá situar la unidad.

```
def add_protaviones(self):

    global matrix_size, rec_pa

    row = random.randint(1,matrix_size)
    col = random.randint(1,matrix_size)

    pivote = self.get_By_Pos(row,col)

    tmp: Box =  self.boxes_head


    #derecha
    tmpr1 = self.get_By_Pos(row,col+1)
    tmpr2 = self.get_By_Pos(row,col+2)
    tmpr3 = self.get_By_Pos(row,col+3)

    #iquierda
    tmpl1 = self.get_By_Pos(row,col-1)
    tmpl2 = self.get_By_Pos(row,col-2)
    tmpl3 = self.get_By_Pos(row,col-3)

    #arriba
    tmpu1 = self.get_By_Pos(row+1,col)
    tmpu2 = self.get_By_Pos(row+2,col)
    tmpu3 = self.get_By_Pos(row+3,col)

    #abajo
    tmpd1 = self.get_By_Pos(row-1,col)
    tmpd2 = self.get_By_Pos(row-2,col)
    tmpd3 = self.get_By_Pos(row-3,col)
```

Este método valida que las posiciones aproximadas de todas las 4 direcciones principales no sean nulas saliendo dela matriz y que al estar dentro de la matriz no contengan ningún color.

```
while tmp != None:
    if (tmp.posX == pivote.posX and tmp.posY == pivote.posY and tmp.color == withe):
        op = random.randint(1,4)

        if(op == 1):
            if(tmpr1 != None and tmpr2 != None and tmpr3 != None):
                if(tmpr1.color == withe and tmpr2.color == withe and tmpr3.color == withe):
                    tmp.color = Maroon
                    tmpr1.color = Maroon
                    tmpr2.color = Maroon
                    tmpr3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 2):
            if(tmpl1 != None and tmpl2 != None and tmpl3 != None):
                if(tmpl1.color == withe and tmpl2.color == withe and tmpl3.color == withe):
                    tmp.color = Maroon
                    tmpl1.color = Maroon
                    tmpl2.color = Maroon
                    tmpl3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 3):
            if(tmpu1 != None and tmpu2 != None and tmpu3 != None):
                if(tmpu1.color == withe and tmpu2.color == withe and tmpu3.color == withe):
                    tmp.color = Maroon
                    tmpu1.color = Maroon
                    tmpu2.color = Maroon
                    tmpu3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        elif(op == 4):
            if(tmpd1 != None and tmpd2 != None and tmpd3 != None):
                if(tmpd1.color == withe and tmpd2.color == withe and tmpd3.color == withe):
                    tmp.color = Maroon
                    tmpd1.color = Maroon
                    tmpd2.color = Maroon
                    tmpd3.color = Maroon
                    rec_pa += 1
                else:
                    pass
        else:
            self.add_protaviones()

    tmp = tmp.next
```

Es un método recursivo que por lo que al no encontrar una posición desde la coordenada aleatoria se llama a si mismo para generar una nueva coordenada e intentar nuevamente la inserción.