ESINAIT: DOCUMENTACION EXTERNA

Realizado por:

Juan Antonio González Cantero
Jesús Márquez Delgado
Claudia Silvestre Muñoz
Ignacio Caballero Marín

1.Descripción Funcional

Al ejecutar el programa, se mostrará una pantalla de inicio con las opciones de iniciar sesión o salir. Si se elige iniciar sesión, si se introduce un usuario registrado, podrá seguir con la ejecución del programa, en caso negativo, se ofrece la opción de registrar un usuario nuevo.

1. Si el usuario no es administrador se le mostrarán las siguientes opciones:

    1.1. **Entrar/Cambiar de usuario** si se desea cambiar de usuario e iniciar se sesion con otro, volviendo así al inicio de sesión del principio pero sin cerrar la sesión actual.
    1.2. **Jugar/Continuar partida** muestra el menú de partida, el cual está compuesto de las siguientes opciones.
        1.2.1. **Unirse**, te añade a la lista de espera.
        1.2.2. **Salir de la lista**, te elimina de la lista de espera.
        1.2.3. **Salir**, vuelve al menú principal.
    1.3. **Comprar objeto**, el cual muestra la lista de objetos que se pueden comprar y el dinero actual, a continuación ofrece la opción de elegir un objeto de la lista para comprarlo y añadirlo a la mochila.
    1.4. **Ver perfil**, muestra los datos del usuario.
    1.5. **Amigos**, muestra el menú de amigos, con las opciones de añadir o eliminar alguno de estos.
    1.6. **Salir del sistema**, cierra sesion y vuelve al menú de inicio.
    1.7. **Ver mochila**, muestra los objetos que tiene el usuario en mochila.
2. Si el usuario es administrador además de las anteriores se mostrarán las siguientes opciones:
    2.1. Dentro de Jugar/Continuar partida se le muestra la opción **iniciar partida**, que únicamente funciona si el número de jugadores en espera es igual o superior al mínimo preestablecido en la configuración.
    2.2. **Configuración**, muestra la configuración del sistema y permite modificarla.
    2.3. **Administrador**, muestra un menú para promover o eliminar administradores.
    2.4. **Menu Admin Objetos**, te muestra las opciones de administrador de objetos que son:
        2.4.1. **Registrar Objeto:** Registra un objeto nuev**o.**
        2.4.2. Borrar Objeto: Elimina un objeto del vector de estructuras.
        2.4.3. Modificar Objeto: Modifica los atributos de un objeto.

Una vez se inicia la partida se muestran las siguientes opciones:

1. **Ver/Usar Mochila,** te muestra tus objetos en mochila y te permite sacar uno para usarlo.
2. **Usar objeto/Disparar**, en caso de ser un armate muestra una lista de oponentes al alcance y te da la opción de elegir un objetivo al que disparar. Si no, si es un escudo te aumenta el escudo, y si es una poción, venda, etc te aumenta la vida hasta un máximo de 100. Tras ser usados se eliminan de la mochila y además consume una acción.
3. **Mover Jugador**, te da opción de moverte hacia arriba, abajo, izquierda o derecha además de la opción de salir si no se desea mover. Si el usuario elige moverse, aumentará su coordenada en la distancia de paso a la dirección elegida y aumentará un turno.
4. **Ver objetos cercanos**, muestra los objetos al alcance para recoger, y permite recogerlos en el caso que se quiera.
5. **Ver oponentes cercanos,** muestra los oponentes que están al alcance del arma elegida.
6. **Ver posición actual**, muestra las coordenadas actuales en el mapa
7. **Ver posición y límites del mapa**, muestra el centro de la tormenta y los límites al norte,sur,este y oeste.
8. **Finalizar turno**, pasa al turno del siguiente jugador.
9. **Volver al menú principal**, sale del menú de partida.

2.División del Problema

Para la división del ejercicio optamos por separar los datos en diferentes módulos(usuarios.c, tienda.c, mochila.c, mapa.c, tormenta.c y configuración.c), además de estos también dividimos los menús en dos partes, el menú principal en un main.c y el menú de partida en un módulo partida.c.

- **Usuarios.c**, el cual tiene todas las funciones encargadas de la gestión de los usuarios del sistema. Realizado por Jesús Márquez Delgado.

- **Tienda.c**, en este módulo se reúnen todas la funciones de gestión de objetos, además de los menús para la tienda y una función para comprar objetos en esta. Realizado por Juan Antonio González Cantero

- **Mochila.c,** módulo encargado de almacenar los objetos en mochila de los usuarios, además de ofrecer opciones de gestión como borrar de la mochila o insertar, etc. Realizado por Claudia Silvestre Muñoz.

- **Mapa.c**, modulo que inicializa tanto la posición usuarios como la de objetos en el mapa cuando se inicia la partida, estableciendo los limites de dicho mapa para no colocara nadie fuera de este. Realizado por Ignacio Caballero Marín y Jesús Márquez Delgado.

- **Tormenta.c**, módulo que complementa el mapa generando las tormentas de la partida y aportando la función de eliminar todo elemento fuera de la tormenta. Realizado por Ignacio Caballero Marín y Jesús Márquez Delgado.

- **Configuración.c**, módulo que tiene registrada la configuración del sistema , para ser utilizadas en los diferentes módulos. Realizado por Claudia Silvestre Muñoz.

- **Partida.c**, contiene todos los menús para la partida, además de utilizar todos los módulos para su correcto funcionamiento. Realizado por Jesús Márquez Delgado

3.Documentación por módulo

- **Módulo Usuario**
    - **Sección de importación**
        - Incluye el fichero de cabecera de “configuracion.h” y las librerias “stdio.h”,”stdlib.h”,”string.h”
    - **Sección de exportación**
        - Índice usuario inicia sesión
        - Índice de un usuario concreto
        - Número de jugadores en juego
        - Número de jugadores en espera
    - **Funciones**
        - **cargar_usuarios(usuarios **u):** Inicializa los vectores de estructuras de usuarios con los datos del fichero, si este existe.
        - **guardar_usuario(usuario *u):** Guarda en un fichero los datos de los vectores de estructuras de usuario.
        - **cargar_amigos(amigo **a):** Inicializa los vectores de estructuras de amigos con los datos del fichero.
        - **guardar_amigos(amigo *a):** Guarda en un fichero los datos de los vectores de estructuras de amigos.

- **iniciar_sesion(usuario *u,configuracion c):** Recibe la el vector de estructuras usuario y la estructura configuracion, pide el nick del usuario y la contraseña, comprueba que el usuario exite y que la contraseña es correcta y devuelve el índice del usuario. En caso de no existir el usuario, pregunta si el usuario quiere registrarse como un jugador nuevo.
- **confirmar_usuario(usuario *u,char *usuario):** Recibe el vector de estructuras de usuario y una cadena con un nick, comprueba que el nick pertenece a un usuario y devuelve 0 si existe o -1 si no le pertenece a ningún usuario
- **crear_usuario(usuario **u, configuracion c):** Recibe el vector de estructuras de usuario y la estructura configuracion, pide un nick para crear el un nuevo usuario, si este existe se cancela el registro, si no, pide que se termine el registro y se crea una nueva estructura usuario
- **ver_perfil(usuario *u,int i):** Muestra los datos del usuario del que se recibe el índice.
- **lista_amigos(amigo *a,char *nick):** Muestra la lista de amigos del usuario dado.
- **m_amigos(amigos **a,char *nick):** Entra al menú de amigos con el vector de estructuras amigos por referencia y la cadena con el nick del usuario. En este se muestran las opciones de amigos.
- **aniadir_amigo(usuario *u,amigos **a,char *nick):** Recibe el vector de estructura por referencia de amigos y por valor de usuario y una cadena con el nick del usuario. Pide el nombre del amigo y si existe, se crea una nueva estructura amigos con ambos nicks.
- **comparar_amigo(char *nick,char *nickamig,amigo *a):** Compara si el usuario tiene el nickamig agregado, si está agregado devuelve 0, si no -1.
- **borrar_amigo(usuario *u,amigo **a,char *nick):** Busca si el usuario ha agregado ya al otro usuario y si es asi borra la estructura donde está almacenada.
- **m_admin(usuario **u):** Muestra el menú de configurar administradores.
- **indice_usuario(usuario *u,char id[100]):** Devuelve el índice del usuario que recibe como cadena id.
- **njugadores_EE(usuario *u):** Devuelve el número de usuarios en línea registrados.
- **njugadores_EJ(usuario *u):** Devuelve el número de usuarios en línea registrados.

        - **inicializar_vida_escudo(usuario **u):** Inicializa la vida a 100 y el escudo a 0 de todos los usuarios.

- **Módulo Partida**

    - **Sección de importación**
        - Incluye los ficheros de cabecera “tormenta.h”,”tienda.h”,”usuario.h”,”mapa.h” y “Mochila.h” y las librerías “string.h”, “stdio.h”, “stdlib.h”, “time.h” y “math.h”
    - **Funciones**
        - **lobby(usuario **u,configuracion c,int *indice,Elemento **jm,mochila **m,objetos *o,tormenta **t):** Si está en partida, muestra las opciones de partida, si el estado es “GO”, imprime “has muerto” y si no es ninguna de estas, muestra el menú de espera, con opciones entrar lista, salir lista y salir si es jugador o las mismas mas iniciar partida si es admin, opciones para unirte a la lista de espera e iniciar la partida.
        - **op_usuario_partida(usuario *u,configuracion c):** Muestra las opciones de usuario en partida.
        - **op_admin_partida(usuario *u,configuracion c):** Muestra las opciones de administrador en partida.
        - **jugadores_espera(usuario *u, configuracion c):** Muestra la lista de jugadores en espera.
        - **terminar_partida_jugadores(usuario **u,Elemento **jm):**
        - **movimiento(Elemento **jm,int indice,configuracion c, int *accion):** Muestra un menú con los movimientos posibles para el usuario, según la elección amplía o disminuye las coordenadas de la posición y aumenta el número de acciones.
        - **id_objeto(objeto *o,mochila *m,usuario *u,int indice):** Devuelve el id del objeto en el vector de estructuras de objetos del id del vector de estructuras de mochila
        - **usar_arma(ususario **u,Elemento **jm, mochile **m, int *turno, int alcance, int danio, int *ido.objetos *o,int *accion):** Le introduces el id del objeto y le da la opción de atacar a un jugador al alcance, si le ataca se le reduce la vida y si esta llega a 0, el jugador atacado pasa a estado “GO”.
        - **mostrar_jugadores_cercanos(Elemento *jm,int idm,int alcance):** Muestra una lista de jugadores al alcance del arma en mano.
        - **mostrar_objetos_cercanos(Elemento **jm,int *idm,configuracio c,mochila **m,usuario *u,int indice,int**

***accion):** Muestra una lista de objetos al alcance de recoger y da la opcion de recogerlos si se quiere.

- **turno_jugador(Elemento *jm,char *nick,int indice):** Busca un nick de usuario en la lista de elementos y devuelve su índice
- **usar_objeto(usuario **u,objetos *o,mochila **m,int ido,int indice):** Recibe el id de un objeto accesorio y si es un escudo, aumenta el porcentaje de escudo de este y si no lo es, aumenta el porcentaje de vida a la vida del usuario.
- **estar_partida(usuario **u,Elemento *jm,int indice):** Devuelve 0 si el usuario recibido con un índice está en el vector de estructuras Elemento.

- **Módulo Tienda**
  - **Sección de importación**
    - Incluye los ficheros de cabecera de “mochila.h”, “configuracion.h”, “usuario.h”, “stdin.h”, “stdlib.h” y “string.h”
  - **Funciones**
    - **void cargar_objetos(objetos **obj):** Inicializa los vectores de estructuras de objetos con los datos del fichero, si este existe.
    - **void lista_objetos(objetos *obj):** Muestra por pantalla la lista de objetos existentes.
    - **void comprar_objetos(mochila **moch,objetos *obj,usuario **usua,int iuser,configuracion c):** Se muestra la lista de objetos disponibles para su compra y permite al usuario comprarlos si dispone de dinero suficiente. Tras comprarlo el objeto se guarda en su mochila.
    - **void registrar_objetos(objetos **obj):** Permite a un usuario con permisos de administrador registrar un objeto nuevo. Comprueba mediante el ID del objeto introducido si el objeto ha sido registrado con anterioridad.
    - **void borrar_objetos(objetos **obj):** Permite a un usuario con permisos de administrador borrar un objeto que haya sido registrado. Comprueba mediante el ID introducido si el objeto que se desea borrar existe.
    - **void modificar_objetos(objetos **obj):** Permite a un usuario con permisos de administrador modificar los parámetros de un

objeto que haya sido registrado con anterioridad. Comprueba mediante el ID introducido si el objeto que se desea modificar existe.

- **void guardar_objetos(objetos *obj):** guarda los cambios producidos en la estructura objetos en el fichero “objetos.txt”

- **void m_admin_tienda(objetos *obj):** Muestra las opciones de administrador de objetos y tienda (borrar,modificar y registrar).

- **Módulo Mochila**

  - **Sección de importación:** Incluye los ficheros de cabecera “usuario.h” y “configuracion.h” correspondientes a los módulos Usuario y Configuracion. También incluye las librerías ”stdio.h”, “string.h”, “stddef.h” y “stdlib.h”.
  - **Sección de exportación:** Exporta el índice del objeto que quiere utilizar un usuario durante su turno.
  - **Funciones:**
    - **void cargarficheroMochila (mochila **eMochila):** Recibe el vector eMochila sin inicializar y carga en él la información del fichero “Mochilas.txt”. Por lo tanto, carga en la mochila los objetos de todos los usuarios.
    - **void guardarficheroMochila (mochila *eMochila):** Recibe el vector eMochila con los objetos actualizados después de haber sido cogidos y usados y los vuelca en el fichero “Mochilas.txt”.
    - **void leerMochila (mochila *eMochila, int j, usuario *u):** Muestra por pantalla los objetos que posee el usuario que tiene el turno.
    - **int usarMochila (mochila *eMochila, int j, usuario *u):** Esta función sirve para seleccionar un objeto que se quiere utilizar durante el turno de un determinado usuario. Se introduce el nombre del objeto y la función devuelve la posición de ese objeto en el vector eMochila para en la función Partida ver el daño, beneficio o alcance de ese objeto.
    - **void eliminarobjeto (mochila **eMochila, int i):** Este procedimiento es para después de ser utilizado un objeto, por lo que se recibe el vector eMochila y la posición del objeto a incrementar una unidad de la mochila.
    - **void guardar_mochila (mochila **eMochila, char *objeto, int j, usuario *u, configuracion c):** Este procedimiento se utiliza para guardar un objeto en la mochila después de recogerlo en

el mapa o después de comprarlo en la tienda durante el turno de un usuario. La mochila de cada usuario tiene un máximo que viene indicada en la configuración por lo que si se alcanza ese máximo se le preguntará al usuario si quiere cambiarlo por otro objeto que ya está en la mochila.

- **Módulo Configuración**

    - **Sección de importación:** Incluye los ficheros de cabecera “stdio.h”, “string.h” y “stdlib.h”.
    - **Funciones:**
        - **void cargar_configuracion(configuracion *c):** Este procedimiento recibe el vector c sin inicializar y carga en él la información del fichero “configuracion.txt”. Por lo tanto, carga la configuración del juego.
        - **void guardar_configuracion(configuracion c):** Recibe el vector c con los datos de configuración después de la partida por si ha habido modificaciones durante la partida y los vuelca en el fichero “configuracion.txt”.
        - **void mostrar_configuracion(configuracion *c):** En este procedimiento se muestra la configuración actual y se pide un nuevo valor en el caso que el administrador deseara cambiar un valor o está la opción Salir por si finalmente el administrador no quiere modificar nada.

- **Módulo mapa**

Sección de importación

Incluye los ficheros de cabecera “usuario.h”, “tienda.h”, “configuración.h” que corresponden a sus respectivos módulos, así como las librerias “stdio.h”, “stdlib.h”, “math.h”, “time.h” y “string.h”.

Sección de exportación

Se exporta el tipo registro “Elemento” con sus respectivas variables, un entero llamado “nusuarios” que guarde el número de usuarios usados en el módulo, así como 5 funciones: “generar_mapa”, “guardar_mapa”, “borrar_elemento”, “cargar_mapa” y “elementos_mapa” con su respectivo funcionamiento el cual se explicará más adelante.

Funciones

Void aleatorio (int* x, int* y, configuración c)

Se pasa por referencia dos enteros así como la estructura del módulo “configuración” y mediante la operación incluida en la librería “stdlib.h” rand() asignamos un número a cada posición limitada al radio del mapa. Esta función es esencial para generar tanto jugadores como objetos en el mapa.

Void generar_mapa (Elemento **vector, usuariou, objetos *o, configuración c)**

En esta función se crea el mapa, de manera que se generen los personajes, los objetos y todo lo necesario para iniciar la partida.

Se definen 5 enteros y un vector dinámico, se le asigna a la variable njugadores el valor resultante de la llamada a la función njugadores_EE((*u)) que se encuentra en el módulo “usuarios” la cual contiene el número de jugadores que hay.

En el primer bucle se procede a pasar los jugadores que se encuentren en lista de espera a incluirlos en el juego, así como asignar el tamaño al vector donde se guardarán posteriormente.

Los dos siguientes bucles son para dar valores a cada estructura de tipo Elemento en el que se guarda la información respectiva a cada jugador u objeto con respecto al mapa. Por lo que el en primero se guarda que es un jugador, su nickname y sus posiciones de manera aleatoria. En de define el tipo como objeto, se escribe qué objeto es según la nomenclatura usada en otros módulos como “mochila” o “tienda” y finalmente se les asigna posiciones al igual que en el anterior.

Void guardar_mapa (Elemento *vector, int índice)

Esta función se centra en guardar en ficheros toda la información resultante guardada durante la partida en el vector de estructuras. Para ello, se abre el fichero, si no existe se crea, y se guarda toda la información introducida de cada jugador u objeto anteriormente separando cada apartado de cada elemento por el carácter ‘/’. Finalmente se cierra el fichero.

Void borrar_elemento (Elemento **vector, int id)

En esta función se comprueba si un elemento ya no se encuentra en el mapa y lo elimina del vector en el que están contenidos.

Para ello comprueba si la id pasada en la llamada es distinta que la del elemento que se desea comprobar y si es así la sustituye, cambiando el valor de todas las

variables en la estructura y disminuyendo en 1 la cantidad de elementos en la variable “nelementos”. Finalmente reajusta el tamaño del vector.

Void elementos_mapa (Elemento *e)

Esta función imprime por pantalla cuando el jugador los solicite todos los elementos disponibles en el mapa, tanto si es un objeto como si es un enemigo, así como su posición en coordenadas.

Para ello se recibe el vector en el que están todos los elementos y se imprime todas las posiciones de este.

Void cargar_mapa (Elemento **e)

La finalidad de esta función es leer el fichero “mapa.txt” para que el jugador pueda continuar la partida donde lo dejó la última vez sin perder el progreso.
Para ello se lee el fichero con la información ya guardada, y se va sobreescribiendo cada uno de los apartados de la estructura por los del contenido del fichero, así con cada uno de los elementos del vector “e”. Para el correcto funcionamiento de la función se ha usado una función “strtok” para que al leer la cadena del fichero en la cual están separados los datos de cada elemento por el carácter ‘/’ se extraiga cada uno de los datos para así guardarlo en la estructura.

Int n_jugadores (Elemento *jm)

La función te devuelve un entero en el que están todos los elementos que son jugadores en el mapa.

Para ello recorre el vector y ve que elemento tiene como tipo “Jugador” y suma uno al contador. Finalmente lo devuelve.

Void borrar_mapa (Elemento **jm)

La función elimina todo lo relativo al mapa y redimensiona el vector donde está todo a 0.

Módulo tormenta

Sección de importación

Se incluye solamente el fichero de cabecera “configuración.h”, así como las librerias “stdio.h”, “string.h”, “stdlib.h” y “math.h”.

Sección de exportación

Se exporta un registro “tormenta”, un entero para contar el número de tormentas que se generan y tres funciones: “cargar_tormenta”, “guardar_tormenta” y “generar_tormentas”.

Funciones

Void cargar_tormenta (tormenta **t)

Esta función se encarga de leer el fichero “tormenta.txt” y guardar sus elementos en la estructura generada anteriormente de tipo “tormenta”.

Para ello abre el fichero como lectura y, con el uso de la función strtok, va extrayendo de cada cadena sus elementos separados por el carácter ‘/’ para después introducirlos en cada una de las posiciones del vector. Se corresponderá una línea por cada posición y sustituirá los elementos vacíos de la estructura por los que se guardaron antes de terminar la partida y mediante la función “guardar_tormenta” se registraron en el fichero. Finalmente se cierra el fichero.

Void guardar_tormenta (tormenta *t)

La finalidad de esta función es guardar en el fichero “tormenta.txt” toda la información respectiva a la tormenta, la posición de cada uno de sus ojos, así como su diámetro y el tiempo que tarda en disminuir este.

Para ello vamos a abrir el fichero “tormenta.txt” como escritura e imprimimos en el fichero cada elemento del vector tormenta en el que está un registro con sus correspondientes datos y los separamos entre sí con el carácter ‘/’.
Finalmente se cierra el fichero.

Void generar_tormentas (tormenta **t, configuración c)

Esta función es la principal del módulo ya que es la encargada de generar la tormenta que se irá reduciendo a lo largo de la partida.

Para ello vamos a pasarle en la llamada el vector “tormenta” y la estructura de configuración “c”, iniciamos una comparación para que los datos de la primera vez que se genera la tormenta sean diferentes que el resto. En la primera vez que se genera la posición predeterminada va a ser el centro del mapa, x=0 e y=0, y al resto se le asignará lo ya inicializado como el tiempo o el diámetro.

En el resto de las veces que se genera esta las posiciones de los ojos van a ser aleatorias. Va a haber un total de 5 tormentas generadas a lo largo de la partida. Al inicio de cada bucle hay que tener en cuenta de que la variable “ntormentas” se aumentará en uno y este valor será el usado para aumentar de manera dinámica el tamaño del vector “t”.

Void aleatorio_tormenta (int* x, int* y, tormenta *t, int índice)
El propósito principal de esta función es generar las posiciones aleatorias de los ojos de la tormenta.

Por lo que se usa la función rand() limitando, esta generación al radio de la tormenta anterior de manera que cada centro creado se encuentre dentro de los límites de la tormenta.

Int distancia (int x1, int y1, int x2, int y2)

Esta función te devuelve la distancia de entre dos elementos del mapa recibiendo sus coordenadas. Para ello usa la ecuación de distancia entre dos puntos.

Int distancia_e (Elemento *jm, char *u1, char *u2)

La finalidad de esta función es ver la distancia que hay entre el jugador que lo solicita y todos los demás objetos del mapa.

Para ello recorre el vector y va guardando la posición de cada uno de los elementos por pares y hace la ecuación de distancia.

Void fuera_tormenta (tormenta *t, int índice, Elemento **jm, usuario **u)

La función comprueba que todos los jugadores estén dentro de los límites actuales de la tormenta, cambia su estado a “Game Over” y elimina el elemento del vector. Para ello se usan funciones de otros módulos.

4.Proceso de instalación

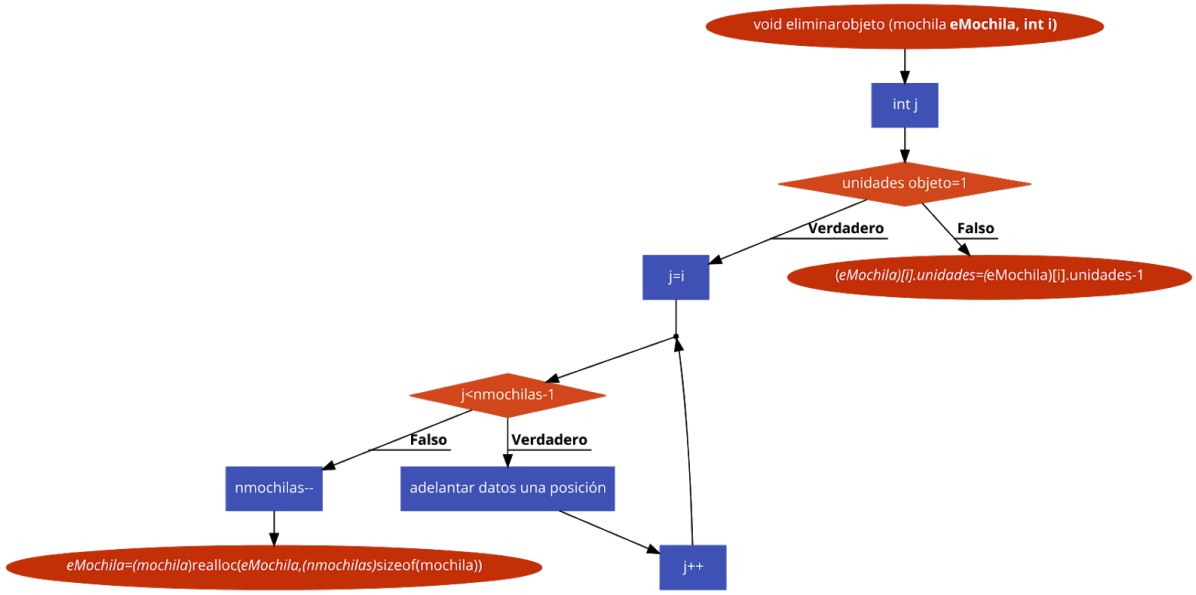
Para la instalación del programa únicamente se necesita el ejecutable, también se puede complementar con los datos en los ficheros si se precisa de ellos.

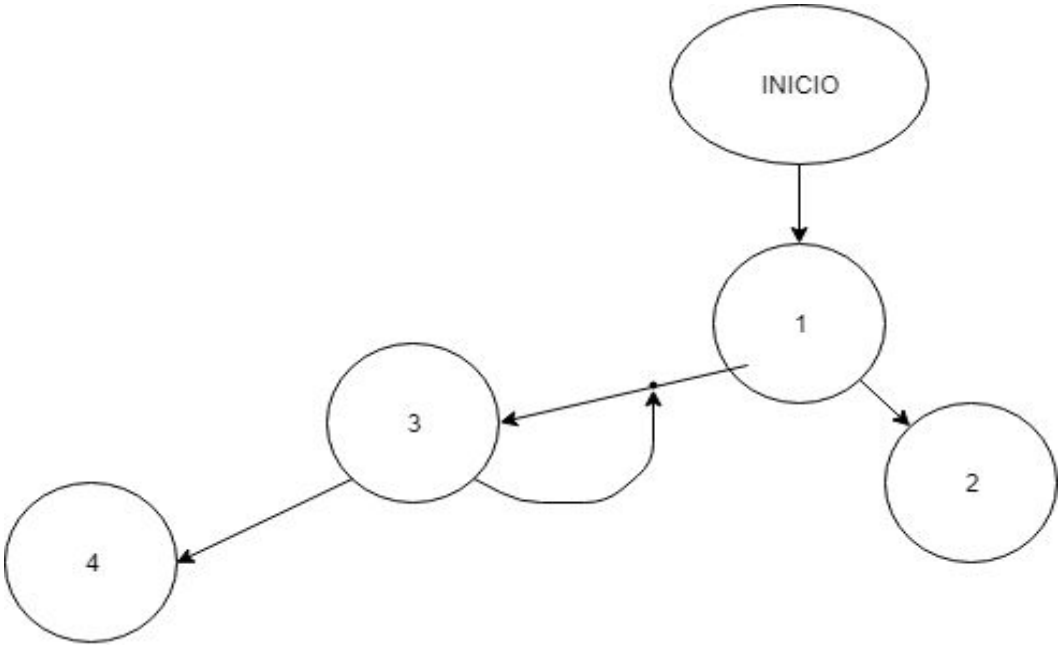
5.Casos de Prueba

MODULO MOCHILA

PRUEBAS DE CAJA BLANCA

- **Procedimiento eliminarobjeto**





La complejidad ciclomática de cualquier función viene dada por:
C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1
Entonces,
C = 3
C = 3

Ruta Nº	
1	1,3,4
2	1,2
3	1,3,3,4

PRUEBAS DE CAJA NEGRA

- Funcion guardar_mochila

Vamos a realizar la prueba de caja negra en este fragmento de código del procedimiento void guardar_mochila (mochila **eMochila, char *objeto, int j, usuario *u, configuracion c):

```
if (cont==c.tam_mochila)
        {
        printf ("La mochila esta llena. Desea reemplazar el objeto por otro? s/n\n"); //porque el maximo de objetos diferetes es 7
        leerMochila(*eMochila,j,u);
        fflush(stdin);
        scanf("%c",&op);

        if (op=='s')
        {
        while (cont1==0)
        {
                printf ("Introduce el objeto que desea reemplazar: \n");
                fflush(stdin);
                fgets(objetoelegido,6,stdin);
                fixstring(objetoelegido);
```

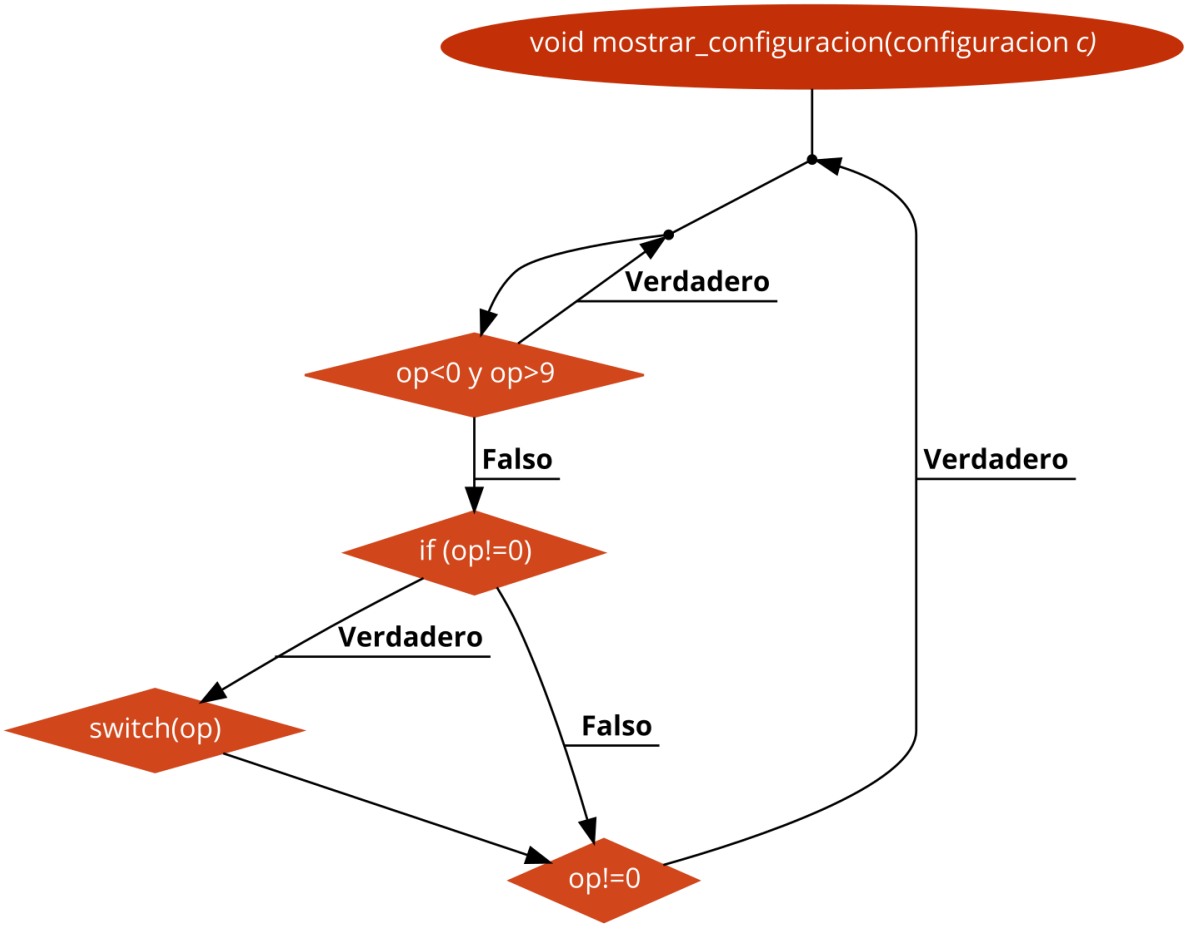
```
		for (i=0;i<nmochilas && cont1==0;i++)
		{
		if (strcmp((*eMochila)[i].idusu,u[j].nick)==0) //usuario actual
	{
		for (i=0;i<nmochilas;i++)
		{
		if (strcmp((*eMochila)[i].idobj,objetoelegido)==0) //para saber en que posicion esta el objeto que se reemplaza
		{
			strcpy((*eMochila)[i].idobj,objeto); //el objeto nuevo reemplaza el que ya estaba
			(*eMochila)[i].unidades=1;
			cont1++;
		}
		}
		}
		}
		if (cont1==0)
		{
		printf ("La ID introducida no es correcta. Introduzca otra: \n");
		} else printf ("El objeto se ha reemplazado correctamente.\n");
	}
	}
	}
```

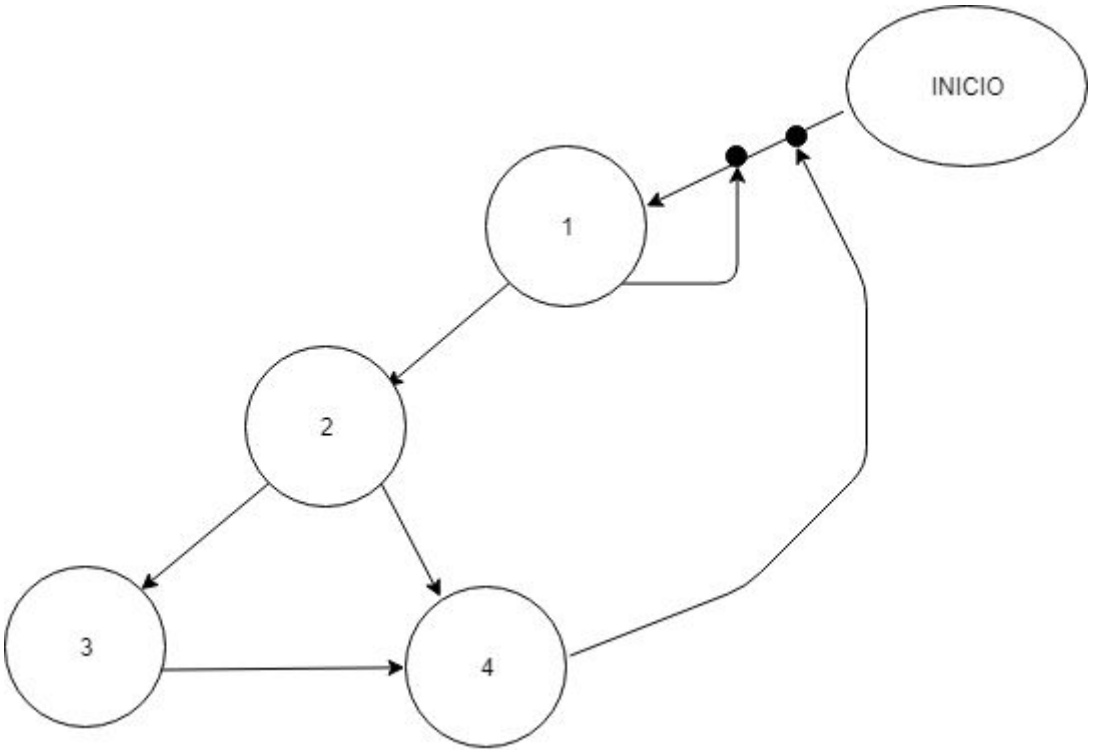
ENTRADA	SALIDA
Combinación de caracteres igual que un objeto que está en la mochila	El objeto introducido se reemplaza por el objeto nuevo que se ha conseguido
Combinación de caracteres distinta a la del objeto introducido	Sigue en el bucle y pregunta por otro objeto
Introduzco un número	Sigue en el bucle y pregunta por otro objeto

MÓDULO CONFIGURACIÓN

PRUEBAS DE CAJA BLANCA

- **Procedimiento mostrar_configuracion**





La complejidad ciclomática de cualquier función viene dada por:

C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1
Entonces,
C = 5
C = 5

Ruta Nº	
1	1,2,3,4
2	1,2,4
3	1,1,2,4
4	1,1,2,3,4
5	1,2,3,4,1,2,4

PRUEBAS DE CAJA NEGRA

- Funcion mostrar_configuracion

Vamos a realizar la prueba de caja negra en este fragmento de código del procedimiento void mostrar_configuracion(configuracion *c):

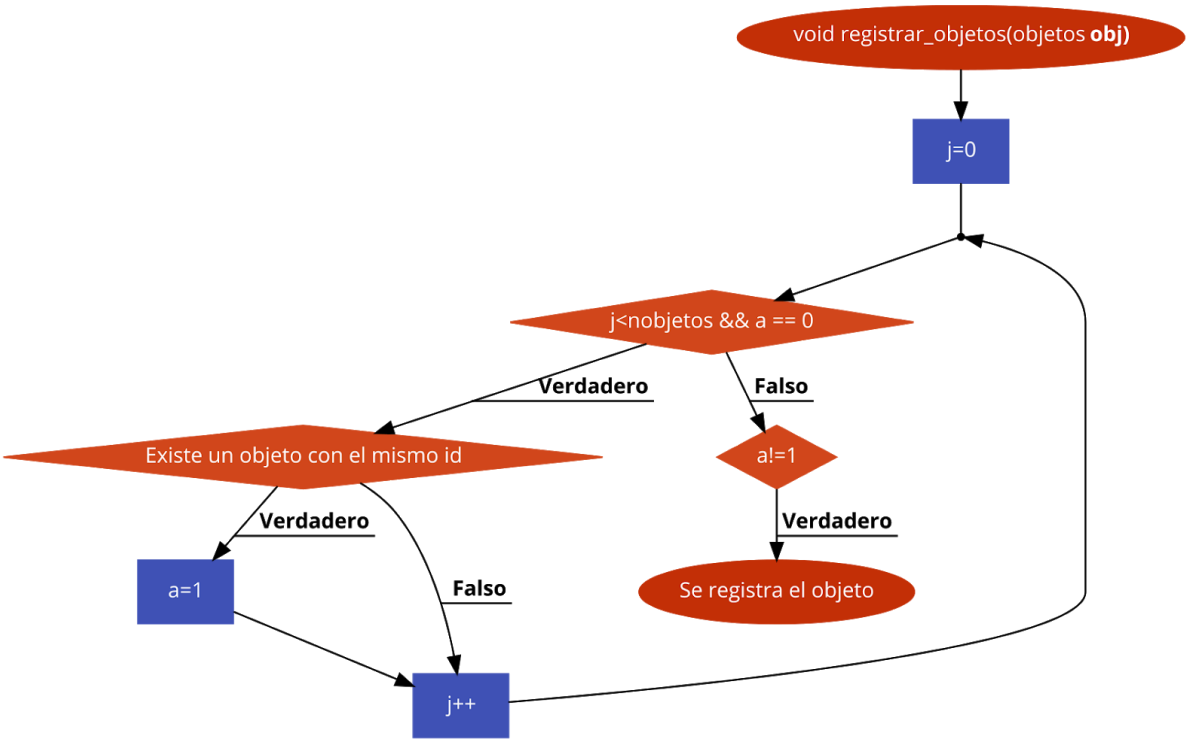
```
do{
                system("cls");
                printf(" |CONFIGURACION|\n");
                printf("
|RAD.MAPA|DIST.RCG|DIST.FSC|DIN.BSE|MIN.JGD|DIST.PASO|TAM.MCH|PRT.NIVEL|N
.ACCN|\n");
                printf(" 1.Radio Mapa: %d\n",(*c).radio_mapa);
                printf(" 2.Distancia Recoger: %d\n",(*c).dist_recoger);
                printf(" 3.Distancia A.Fisico: %d\n",(*c).dist_fisico);
                printf(" 4.Dinero Base: %d\n",(*c).dinero_defecto);
                printf(" 5.Minimo Jugadores: %d\n",(*c).min_jugadores);
                printf(" 6.Distancia Paso: %d\n",(*c).dist_paso);
                printf(" 7.Tamanio Mochila: %d\n",(*c).tam_mochila);
                printf(" 8.Victorias Nivel: %d\n",(*c).partidas_nivel);
                printf(" 9.Numero Acciones: %d\n",(*c).n_acciones);
                printf("\n 1-9.Modificar\n 0.Salir\n Opcion: ");
                scanf("%d",&op);
        }while(op<0&&op>9);
```

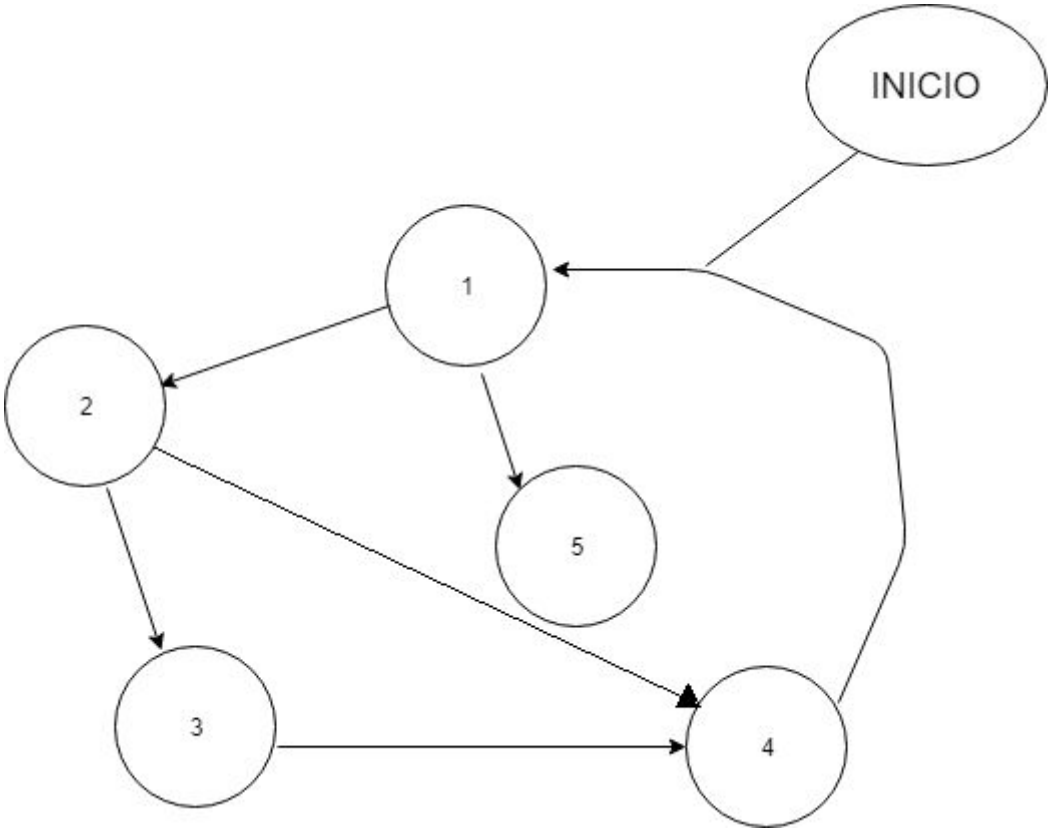
ENTRADA	SALIDA
Introduzco número mayor que 0 y menor que 10	El parámetro asociado al número introducido se cambia por un valor introducido posteriormente
Introduzco número introducido es 0	Se sale de configuración
Introduzco valor diferente de un número	Se repite el bucle y se pide un valor nuevo para op

MÓDULO OBJETOS

PRUEBAS DE CAJA BLANCA

- **Funcion registrar_objeto**





La complejidad ciclomática de una función viene dada por:
C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1

Entonces,
C = 3
C = 3

Ruta Nº	
1	1,2,3,4,1,5
2	1,5
3	1,2,4,1,5

PRUEBAS DE CAJA NEGRA

- Funcion registrar_objetos

Vamos a realizar las pruebas de caja negra de la siguiente función:

```
void registrar_objetos(objetos **obj){

   char nuevo[20];
   int j,a=0;

   printf("Introduzca el ID del objeto que desea registrar: ");
   fflush(stdin);
   gets(nuevo);

   for(j=0;j<nobjetos && a == 0;j++){

      if(strcmp(nuevo,(*obj)[j].item_ID)==0){

         printf("Ya hay un objeto registrado con ese mismo ID.\n");
         a=1;

      }

   }
   if(a!=1){
```

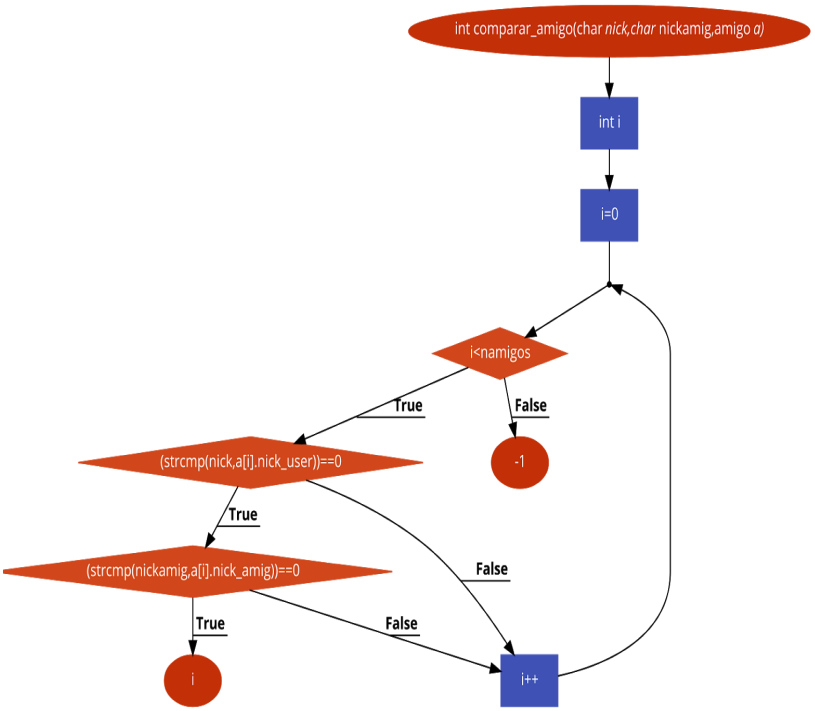
```
        nobjetos = nobjetos + 1;
        *obj = (objetos*)realloc((*obj),nobjetos*sizeof(objetos));
        strcpy((*obj)[nobjetos-1].item_ID,nuevo);

        printf("Introduce la descripcion del objeto: ");
        fflush(stdin);
        gets((*obj)[nobjetos-1].descripcion);

        printf("Introduce que tipo de objeto es:  ");
        fflush(stdin);
        gets((*obj)[nobjetos-1].tipo);

        printf("Introduce el coste del objeto:  ");
        scanf("%d",&(*obj)[nobjetos-1].coste);

        printf("Introduce el alcance del objeto:  ");
        scanf("%d",&(*obj)[nobjetos-1].alcance);

        printf("Introduce el porcentaje de daño si es un arma o de reposicion si es un
escudo: ");
        scanf("%d",&(*obj)[nobjetos-1].porcentaje_d_e);

        printf("Objeto registrado.\n");

    }

}
```

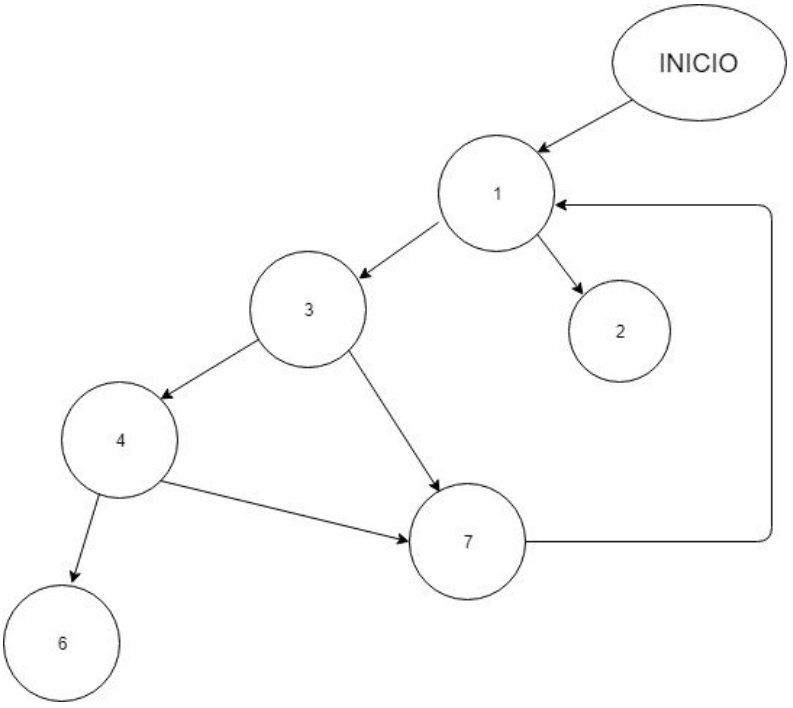
ENTRADA	SALIDA
Introduzco un ID ya existente	El procedimiento imprime que ya existe un objeto con ese ID por pantalla y finaliza.
Introduzco un ID no existente	El programa pide por pantalla una serie de parámetros para registrar el nuevo objeto como descripcion,coste,etc

MÓDULO USUARIO

PRUEBAS DE CAJA NEGRA

- **Función comparar_amigo**





La complejidad ciclomática de una función viene dada por:
C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1
Entonces,
C = 4
C = 4

Nº Rutas	
1	1-2
2	1-3-4-6
3	1-3-7-1-2
4	1-3-4-7-1-2

PRUEBAS DE CAJA NEGRA

- **Funcion m_amigos**:

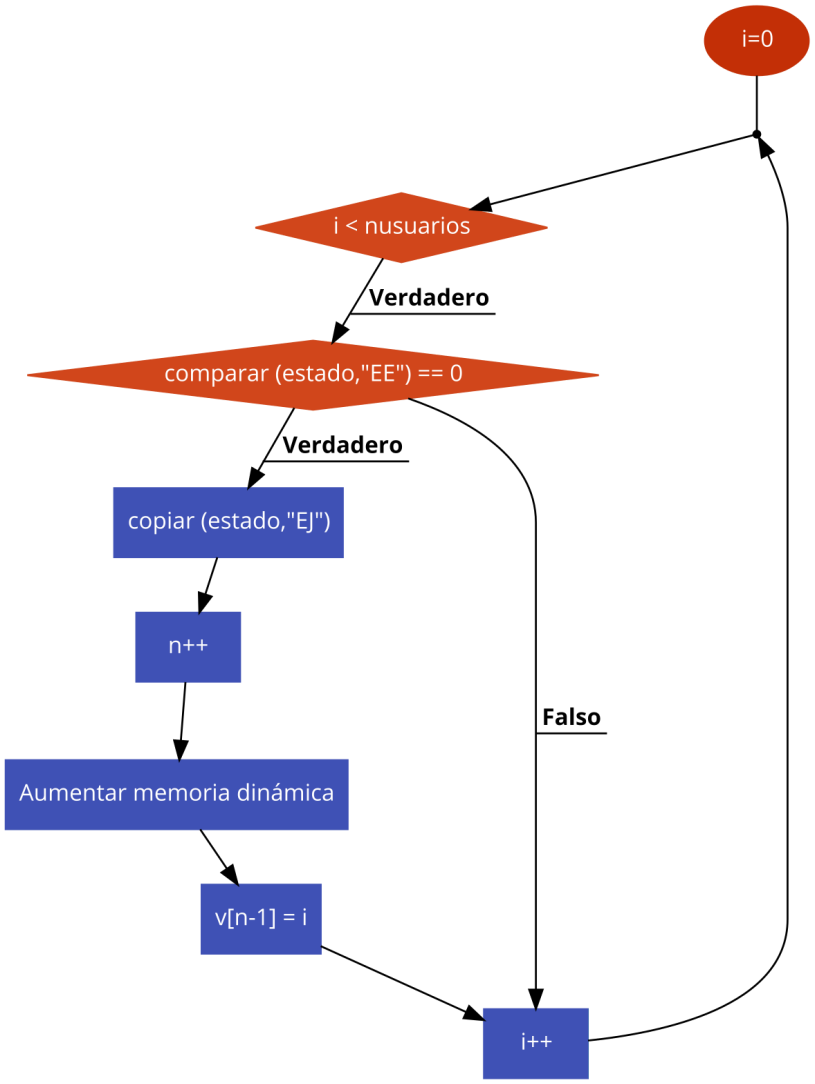
```
do
	{
		do
		{
			lista_amigos((*a),nick);
			printf(" 1.Aniadir amigo\n");
			printf(" 2.Borrar amigo\n");
			printf(" 0.Salir\n");
			printf("\tOpcion: ");
			scanf("%d",&op);
		}while(op<0&&op>2);
		switch(op)
		{
			case 1: aniadir_amigo(u,&(*a),nick);break;
			case 2: borrar_amigo(u,&(*a),nick);break;
		}
	}while(op!=0);
```

ENTRADA	SALIDA
Introduce op=1	Va al case 1: aniadir amigo
Introduce op=2	Va al case 2: borrar amigo
Introduce op > 2	Vuelve a pedirte el op
Introduce op < 0	Vuelve a pedir el op
introduce 0	Sale del do{ }while()

MÓDULO MAPA

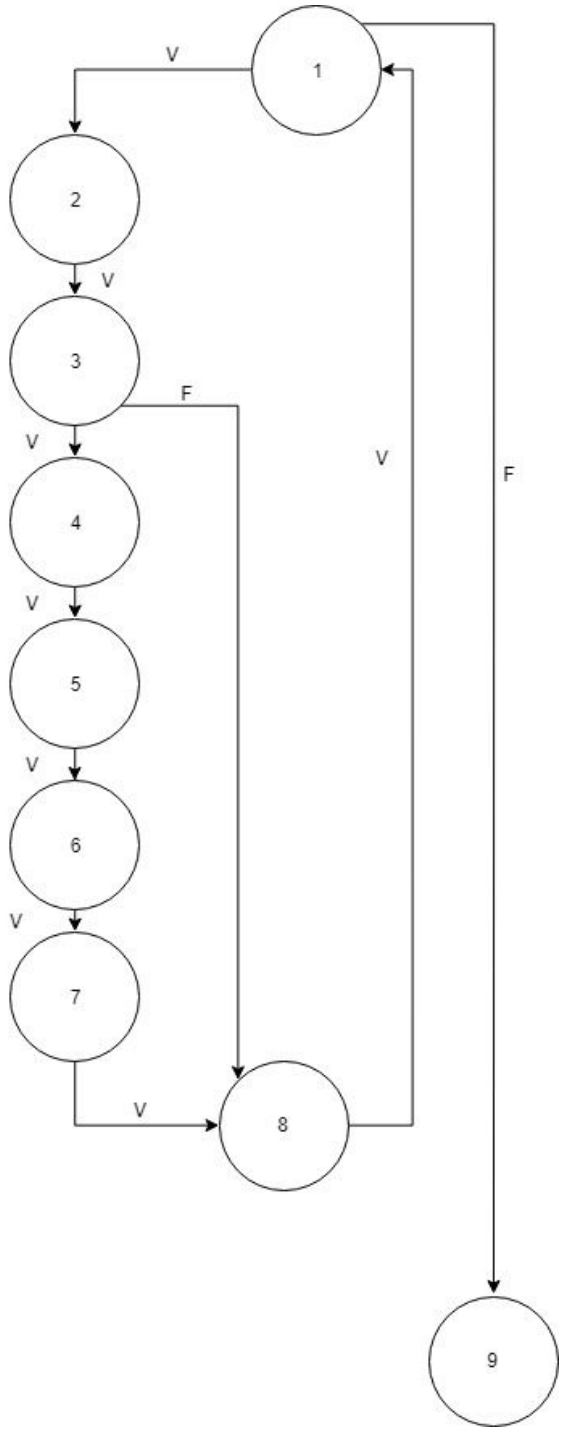
Pruebas de caja blanca

Las pruebas de caja las vamos a hacer según el procedimiento generar_mapa:



La complejidad ciclomática de una función viene dada por:
C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1
Entonces,
Si tiene 9 aristas y 8 nodos:
C= 9-8+2=3
C= 2+1=3

El grafo correspondiente:



Ruta nº	
1	1, 2, 3, 4, 5, 6, 7, 8, 1, 9
2	1, 2, 3, 8, 1, 9
3	1, 3

Prueba de caja negra
Vamos a hacer las pruebas de este segmento con otra función, en este caso vamos a usar el procedimiento borrar_elemento:

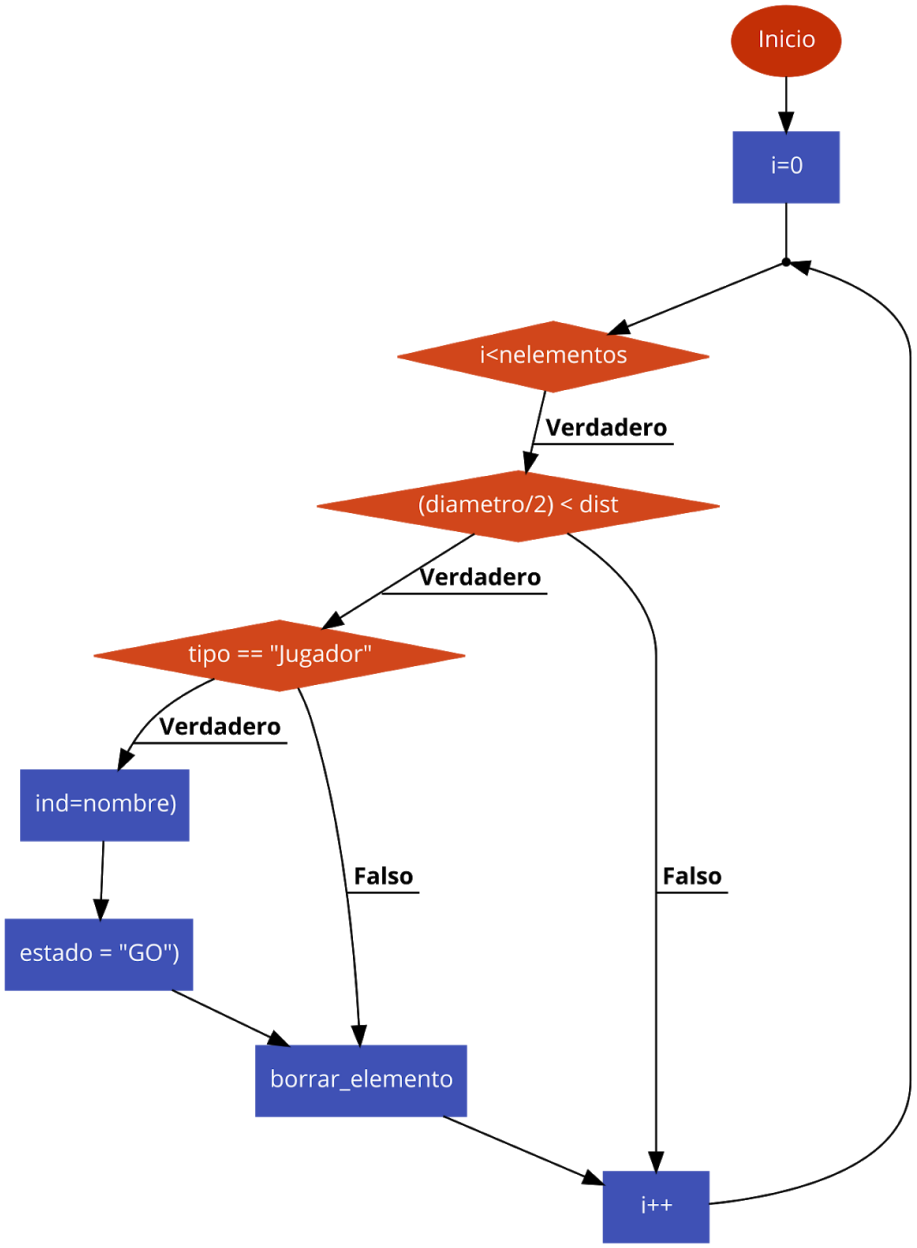
```
if(id!=nelementos-1)
{
        strcpy((*vector)[id].nombre,(*vector)[nelementos-1].nombre);
        strcpy((*vector)[id].tipo,(*vector)[nelementos-1].tipo);
        (*vector)[id].posx = (*vector)[nelementos-1].posx;
        (*vector)[id].posy = (*vector)[nelementos-1].posy;
}
nelementos--;
(*vector) = (Elemento*) realloc ((*vector),nelementos*(sizeof(Elemento)));
```

Entrada	**Salida**
Introducimos un valor para id igual que nelementos-1	Valor de nelementos disminuye en 1 y la dimensión del vector de decrementa en 1. La estructura no se ve modificada.
Introducimos un valor para id distinto de nelementos-1	El registro se ve modificado cambiando el elemento del vector con posición de la id dada por el de la posición nelementos-1. Valor de nelementos se disminuye en 1 y la dimensión del vector se decrementa en 1.

Tras realizar la prueba nos damos cuenta de que la variable id puede tener el valor que se desee que se va a eliminar siempre un elemento y sustituirlo en el registro, solo en el caso de que sea igual a nelementos-1 no se vería modificado.

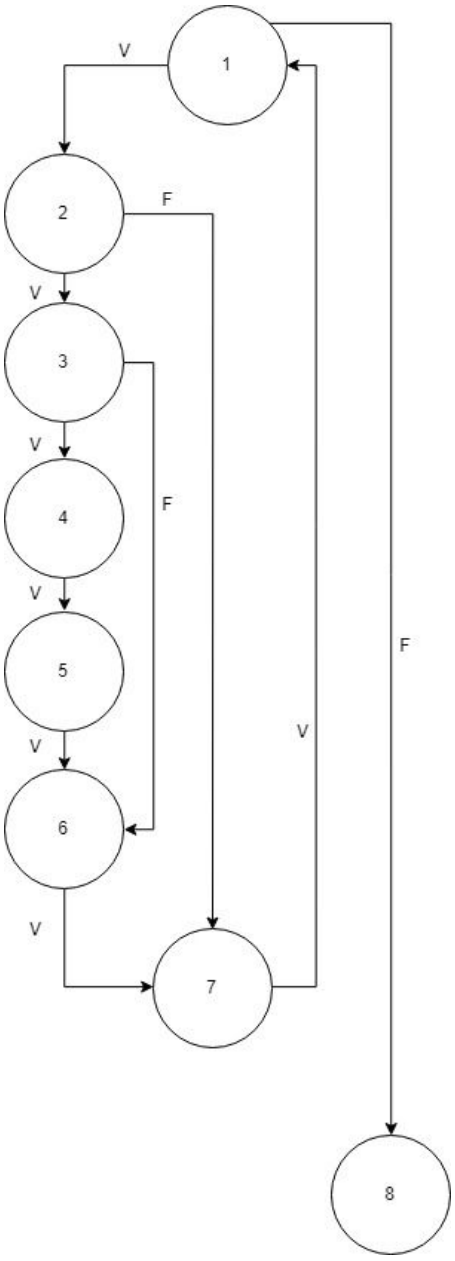
MÓDULO TORMENTA

PRUEBAS DE CAJA BLANCA
Las pruebas de la caja blanca las haremos con el procedimiento fuera_estructura:



La complejidad ciclomática de una función viene dada por:
C=Número de aristas - Número de nodos +2
C=Número de nodos predicados +1
Entonces, tiene 8 nodos y 10 aristas por lo que
C= 10-8+2 = 4
C= 3+1 = 4

Y el grafo sería:



Ruta nº	
1	1, 2, 3, 4, 5, 6, 7, 1, 8
2	1, 2, 7, 1, 8
3	1, 2, 3, 6, 7, 1, 8
4	1, 8

PRUEBAS DE CAJA NEGRA

Vamos a hacer la prueba de caja negra con el funcion distancia_e:

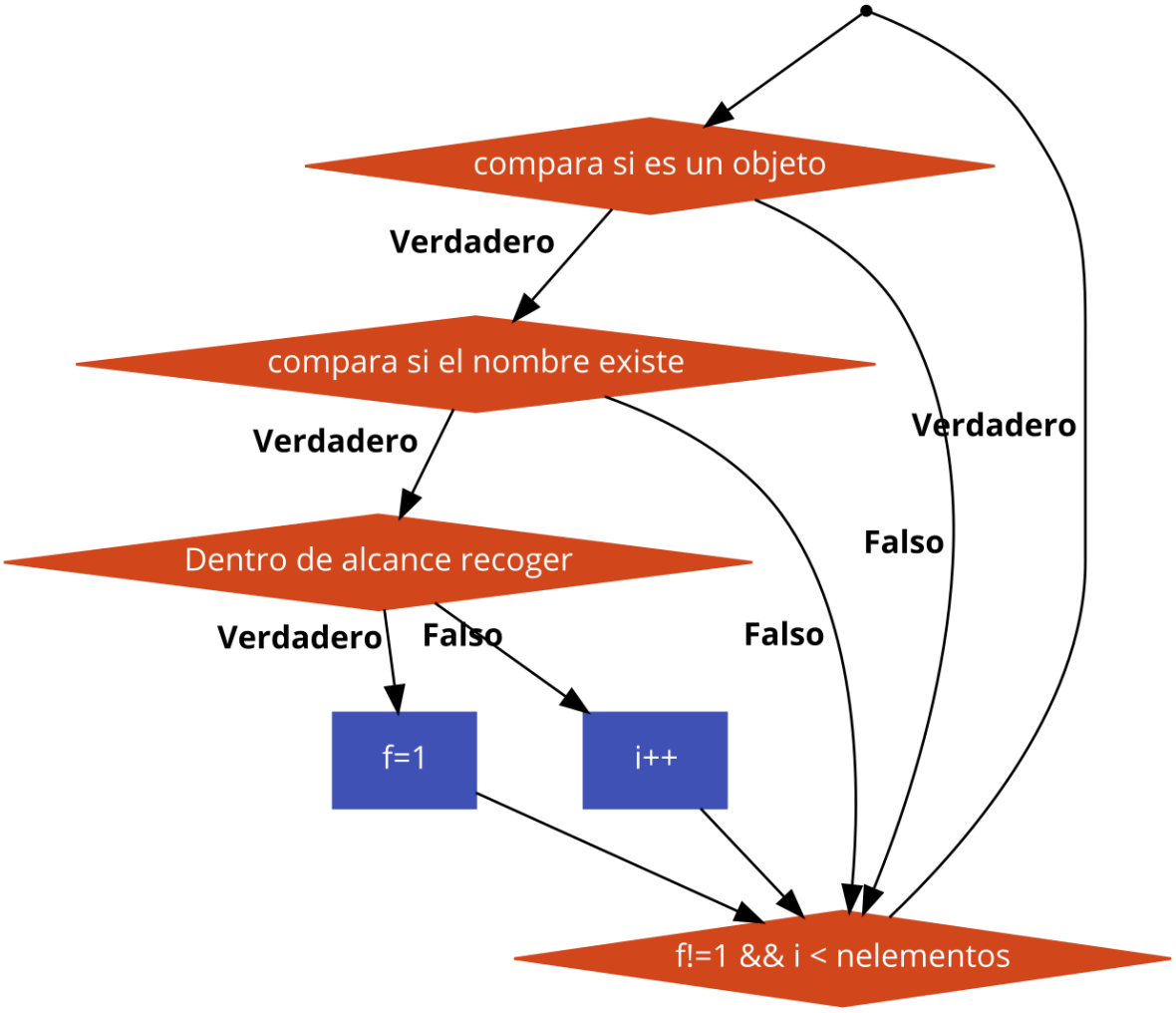
```
int x,y,dist,i,id1,id2;
        for(i=0;i<nelementos;i++){
                if(strcmp(jm[i].nombre,u1)==0) id1=i;
                if(strcmp(jm[i].nombre,u2)==0) id2=i;
        }
        x=jm[id1].posx-jm[id2].posx;
        y=jm[id1].posy-jm[id2].posy;
        dist=sqrt(pow(x,2)+pow(y,2));
        return dist;
```

Entrada	**Salida**
Le pasas u1 y u2 con una cadena de caracteres que sean nicks de los jugadores .	La distancia entre ambos jugadores
Le pasas u1 y u2 con dos cadenas que no tienen ninguna relacion con algun nick de algun jugador.	No te devuelve nada ya que no tiene un índice para el vector

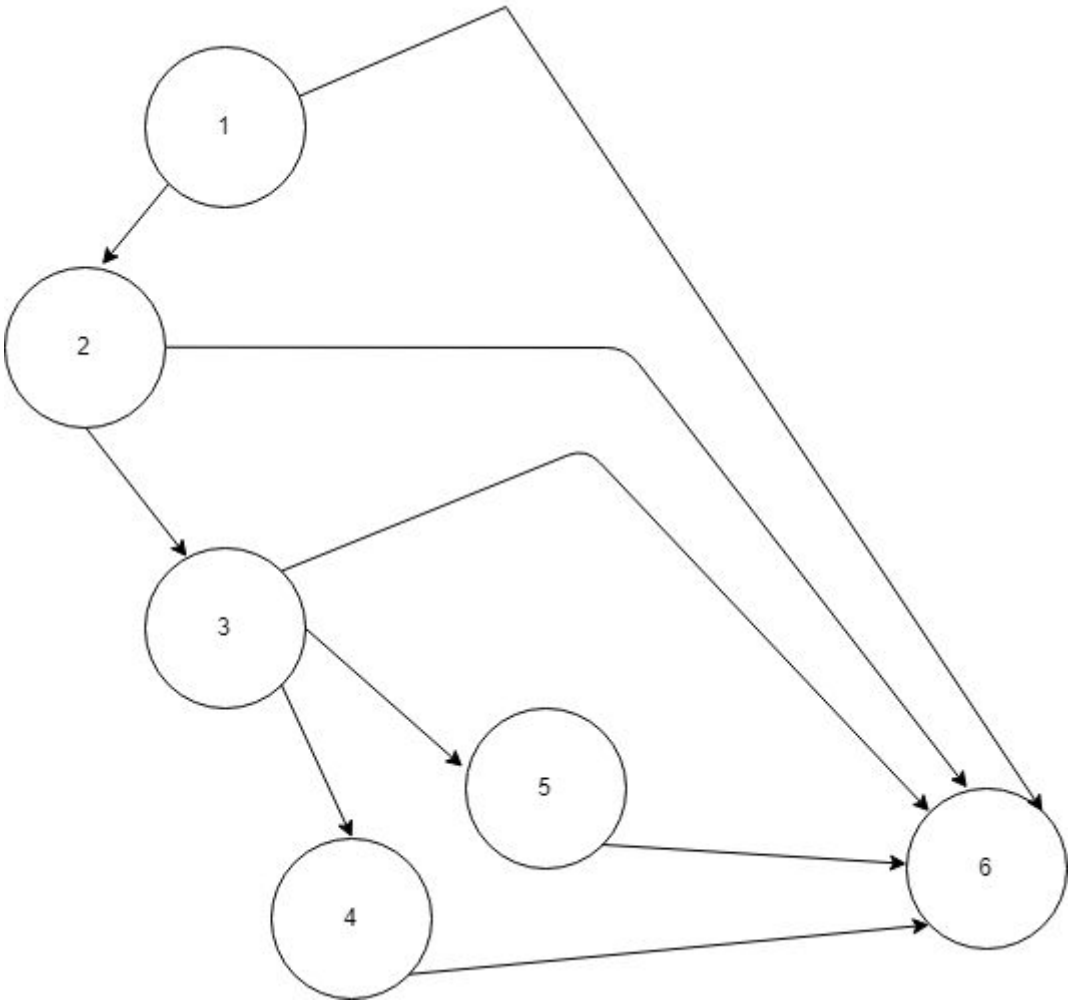
Después de hacer todas las pruebas de caja negra tenemos que sólo si u1 y u2 son nicknames de jugadores en el juego te devuelve un valor entero.

MÓDULO PARTIDA

PRUEBAS DE CAJA BLANCA



La complejidad ciclomática de una función viene dada por:
C=Número de aristas - Número de nodos +2 = 9 - 6 + 2 = 5
C=Número de nodos predicados +1 = 4 + 1 = 5



Nº Rutas	
1	**1-2-3-4-6**
2	**1-2-3-5-6**
3	**1-2-3-6**
4	**1-2-6**
5	**1-6**

PRUEBAS DE CAJA NEGRA

```
int turno_jugador(Elemento *jm,char *nick,int indice)
{
	int i;
	do
	{
		if(strcmp(nick,jm[i].nombre)==0)
		{
			return i;
		}
		i++;
	}while(i<nelementos);
	return indice;
}
```

Entrada	Salida
cadena nombre contenida en Elemento	Devuelve el índice de este
ultima cadena nombre de Elemento	Devuelve el último índice
cadena no contenida en Elemento	Devuelve el indice recibido