ESTRUCTURA DE DATOS

Trabajo Final Integrador

RULETA MORTAL

ALFARO JOSUE DARIO 3490

Facultad de Ingeniería
Universidad Nacional de Jujuy

2024

Contenido

Introducción 2
Análisis del Problema 2
Diseño de la Solución 3
Tipos de datos y estructuras de datos utilizadas 3
Declaración de Constantes 3
Consideraciones de constantes auxiliares: 3
Declaración de Cadena y Punteros a Archivo 3
Declaración de librerías 3
Sección: Estructuras usadas para JUGADORES 4
Sección: Estructuras usadas para PALABRAS 4
Sección: Estructuras usadas para RULETA DE PALABRAS 5
Sección: Estructuras usadas para TURNOS DE JUGADORES 5
Diseño TOP-DOWN del sistema 7
Implementación 8
Consideraciones y aportes que ayudarán a entender la lógica 8
Insertar Jugador 8
Insertar Palabra 9
Generar Ruleta 10
Gestión Jugar 11
Elegir Jugadores para jugar la partida 12
Cargar turnos de jugadores 12
Calcular puntaje de jugador 13
Verificar Ganador 13
Modulo Jugar 14
Pruebas, Observaciones, Consideraciones y casos de error que encontré en el desarrollo 15

Ruleta Mortal

Introducción

Este informe tiene por objetivo explicar el funcionamiento detallado de los módulos más importantes de la "Ruleta Mortal (El poder de las palabras)", mostrando en cada fase el sistema desarrollado desde el planteo del problema, pasando brevemente por el análisis, el diseño propuesto, la codificación de los módulos principales y las pruebas necesarias para el correcto funcionamiento del sistema.

Análisis del Problema

El problema que queremos solucionar es plantear un sistema que pueda proveer una plataforma de juego para la ruleta y gestione turnos de cada jugador, almacene palabras registradas para la ruleta en un banco de palabras y los jugadores registrados en un banco de jugadores de forma ordenada, junto con un cuadro de honor de los jugadores que ganaron, además, al finalizar el juego, guardar el puntaje del ganador y su puntaje más alto.

Para lo cual el sistema utilizará distintas entidades:

JUGADOR

Apellido, Nombre, Nickname (único), Mejor Puntaje, Puntaje total, Cantidad de Partidas ganadas.

PALABRAS

Palabra, Definición, Sinónimos (si es que tiene).

RULETA DE PALABRAS

Ruleta con palabras seleccionadas aleatoriamente.

TURNOS DE JUGADORES

Nickname, Vidas, Puntos, Palabras adivinadas, Cantidad de palabras adivinadas, Longitud de palabra más larga

Diseño de la Solución

Tipos de datos y estructuras de datos utilizadas

Declaración de Constantes

```
const int MAX_CHAR = 50;
const int MAX_SINONIMOS=5;

const int MAX_VECTOR_AUX_RULETA=150;
const int MAX_VECTOR_AUX_JUGADORES=150;
```

Tamaño max. de las cadenas

Cantidad max. de sinónimos por palabra

Define el tam. max. del vector aux. para generar la ruleta

Define el tam. max. del vector aux. para la cola de turnos

Consideraciones de constantes auxiliares:

MAX_VECTOR_AUX_RULETA

Al momento de generar la ruleta de palabras permitimos que el usuario pueda generar una ruleta desde 5 a N (Máximo de palabras registradas en el archivo), Por lo cual antes de registrar palabras validamos que el total de palabras existentes sea menor al MAX_VECTOR_AUX_RULETA , de modo que en la ruleta ingrese la totalidad de palabras registradas.

MAX_VECTOR_AUX_JUGADORES

Al momento de generar la cola de turnos de jugadores el tamaño de la cola debe poder abarcar la totalidad de los jugadores registrados, Por lo cual antes de registrar jugadores validamos que el total de jugadores existentes sea menor al MAX_VECTOR_AUX_JUGADORES, de modo que en la cola de turnos ingrese la totalidad de jugadores del archivo.

Declaración de Cadena y Punteros a Archivo

```
typedef char tcad[MAX_CHAR];
//PUNTERO A ARCHIVO
typedef FILE *parchivo;
```

Declaramos un vector de caracteres de tamaño MAX_CHAR

Declaramos el puntero a un archivo de tipo FILE

Declaración de librerías

```
#include "util.hpp"
#include "jugadores.hpp"
#include "palabra.hpp"
#include "jugar.hpp"
#include "ranking.hpp"
```

Se definieron en este orden porque cada una utiliza estructuras definidas en el orden que fueron definidas.

Declaradas en el main.

Sección: Estructuras usadas para JUGADORES

UBICACIÓN: jugadores.hpp

```
typedef struct jugador
{

    tcad apellido;
    tcad nombre;
    tcad nickname;
    float mejor_puntaje;
    float puntaje_total;
    int partidas_ganadas;

};
```

Representa una entidad de tipo jugador dentro del sistema, con su apellido, nombre, nickname, mejor puntaje, puntaje total y partidas ganadas

```
typedef struct tjugador *pjugador; //Puntero
typedef struct tjugador //nodo jugador
{
    pjugador izq; //puntero izq
    pjugador der; //puntero der
    jugador jug; //registro de tipo jugador
};
```

Representa un ABB de tipo Jugador, se utiliza para mantener el orden del archivo "jugadores.txt" al momento de insertar jugadores nuevos.

Sección: Estructuras usadas para PALABRAS

UBICACIÓN: palabra.hpp

```
typedef tcad vpalabras[MAX_SINONIMOS];
```

Un vector de cadenas de MAX_SINONIMOS

```
typedef struct tsinonimo{
    vpalabras datos;
    int ocup;
};
```

Representa una estructura de sinonimos con su vector de vpalabras y su ocupado.

```
typedef struct palabra
{
    tcad cad;              //cadena
    tcad def;              //definicion
    tsinonimo sin;         //sinonimos
};
```

Representa una entidad de tipo Palabra para la cual se registra su cadena, su definicion y una lista de Sinonimos

```
typedef struct tpalabra *ppalabra;
typedef struct tpalabra
{
    ppalabra izq;
    ppalabra der;
    palabra pal;
};
```

Representa un ABB de tipo Palabra, se utiliza para mantener el orden del archivo “palabras.txt” al momento de insertar palabras nuevas.

Sección: Estructuras usadas para RULETA DE PALABRAS

UBICACIÓN: palabra.hpp

```
typedef palabra v_banco_palabras[MAX_VECTOR_AUX_RULETA];
```

Representa un vector Auxiliar para guardar las palabras del archivo “palabras.txt” y poder generar con palabras aleatorias la ruleta

```
typedef struct tnodo *pnodo;

typedef struct tnodo{
    palabra dato;
    pnodo ant;
    pnodo sig;
};

typedef struct truleta{
    pnodo inicio;
    pnodo fin;
};
```

Representa una estructura de tipo RULETA de palabras, generada con una lista doblemente enlazada con punteros inicio y final, se usa para iniciar la partida.

Sección: Estructuras usadas para TURNOS DE JUGADORES

UBICACIÓN: jugar.hpp

```
typedef struct tnodo_cadena *pnodo_cadena;

typedef struct tnodo_cadena
{
    pnodo_cadena sig;
    tcad dato;
};
```

Representa una lista auxiliar para poder guardar los jugadores para la partida (No repetidos), al iniciar la partida se rellena la cola de turnos con ella.

```
typedef struct tnodo_p *pnodo_p;

typedef struct tnodo_p {
    palabra dato;    //Por cada n
    pnodo_p ant;
    pnodo_p sig;
};

typedef struct tpila {
    pnodo_p inicio;
    int cont;
};
```

Representa una pila de Palabras creada con listas doblemente enlazadas, se utiliza para apilar palabras ganadas de cada jugador

```
typedef struct turno_jugador
{
    tcad    nick;
    tpila   pila;
    float   puntos;
    int     vidas;
    int     cant_adivinadas;
    int     longitud_max_palabra;
};
```

Representa un registro de tipo Turno de un jugador dentro de la partida, por cada turno registramos el Nick del jugador, su pila de palabras, puntos ganados, vidas, cant. de palabras adivinadas y tamaño de la palabra más larga

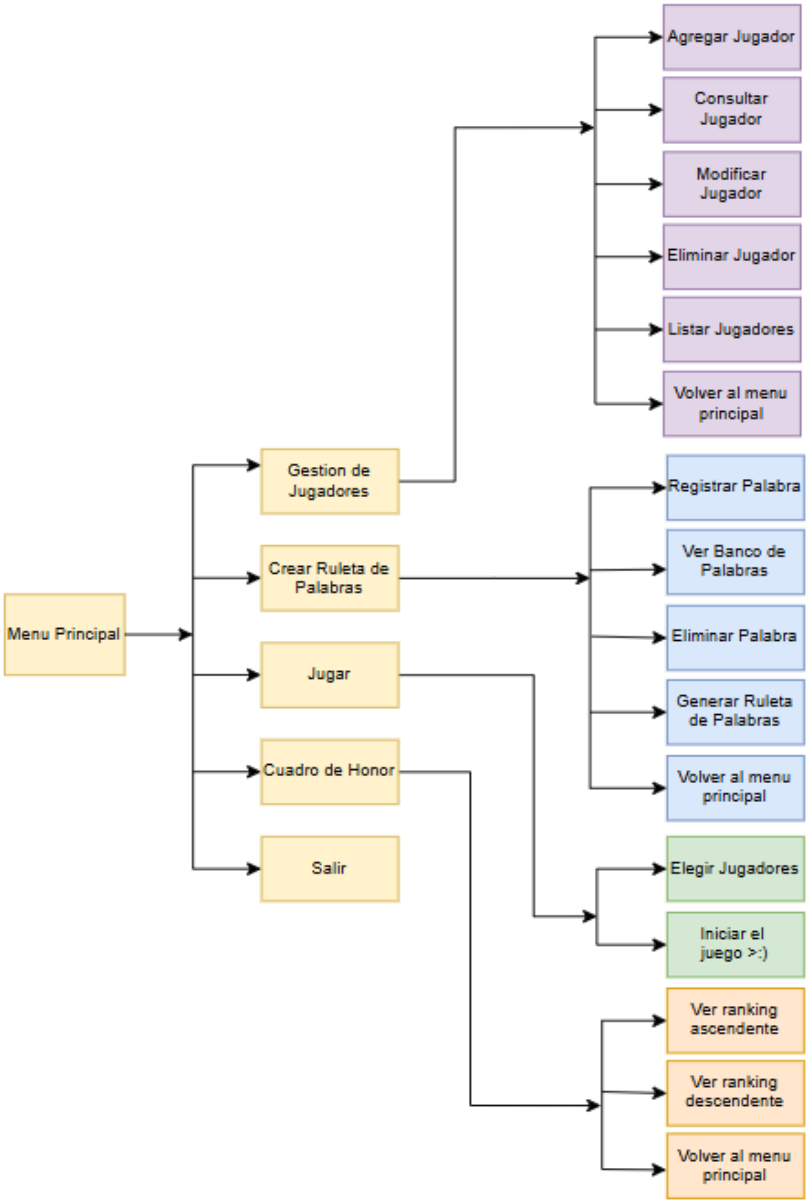
```
typedef turno_jugador tcont_turnos[MAX_VECTOR_AUX_JUGADORES];
```

Definimos un contenedor de datos de tipo turno_jugador, con tamaño del auxiliar.

```
typedef struct tcola
{
    tcont_turnos datos;
    int frente;
    int final;
    int cantidad;
};
```

Definimos la cola de jugadores, que prioriza Almacenamiento de memoria con el contenedor de turnos, su frente, final y cantidad de registros.

Diseño TOP-DOWN del sistema



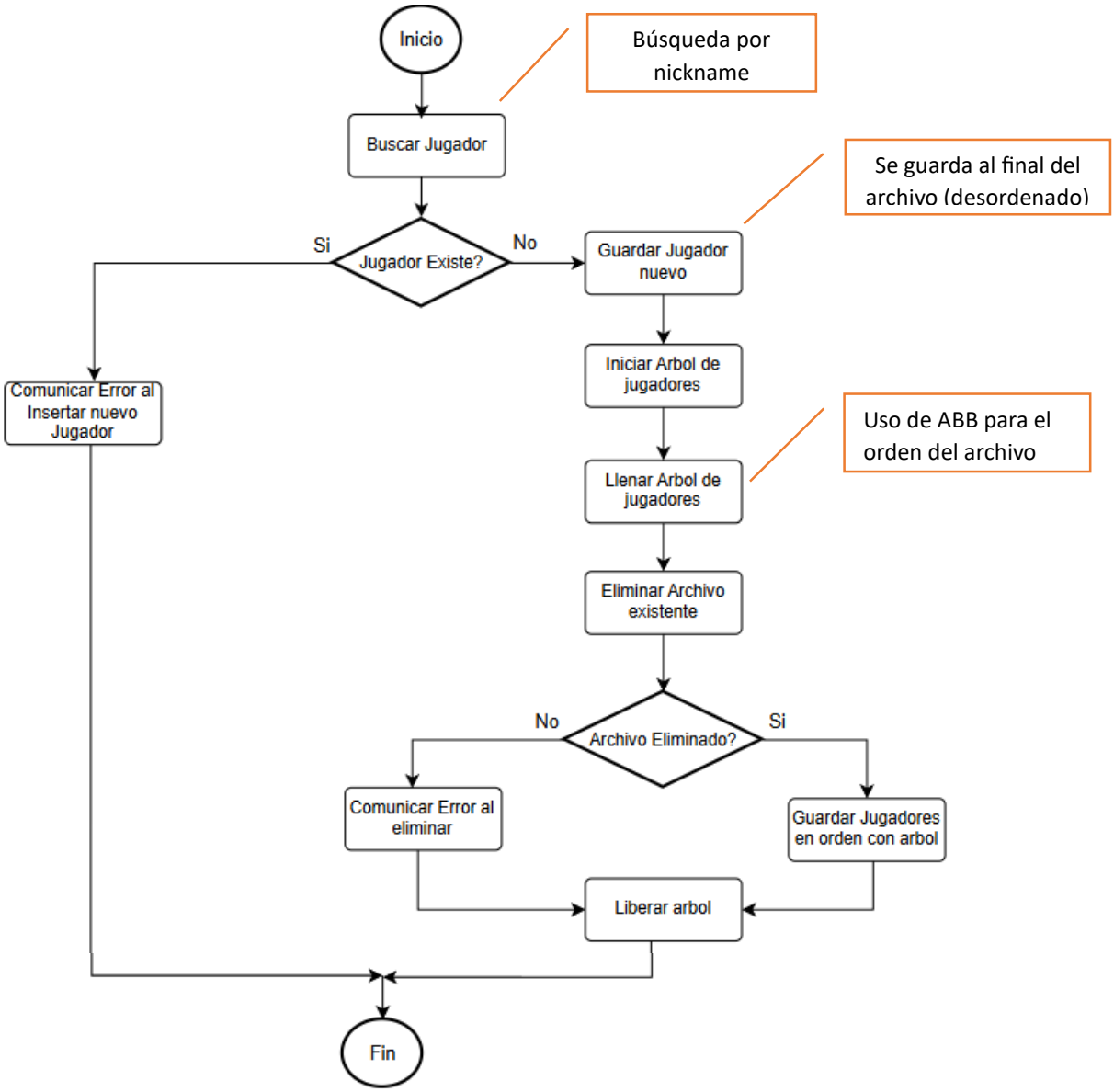
Observación:

Para el módulo de **crear ruleta de palabras** se optó por agregar opciones nuevas, como **ver banco de palabras** (Permite ver todas las palabras registradas) y **eliminar palabra** (Permite eliminar una palabra del banco de palabras)**,** para que el sistema permita gestionar las palabras guardadas en el banco.

Implementación

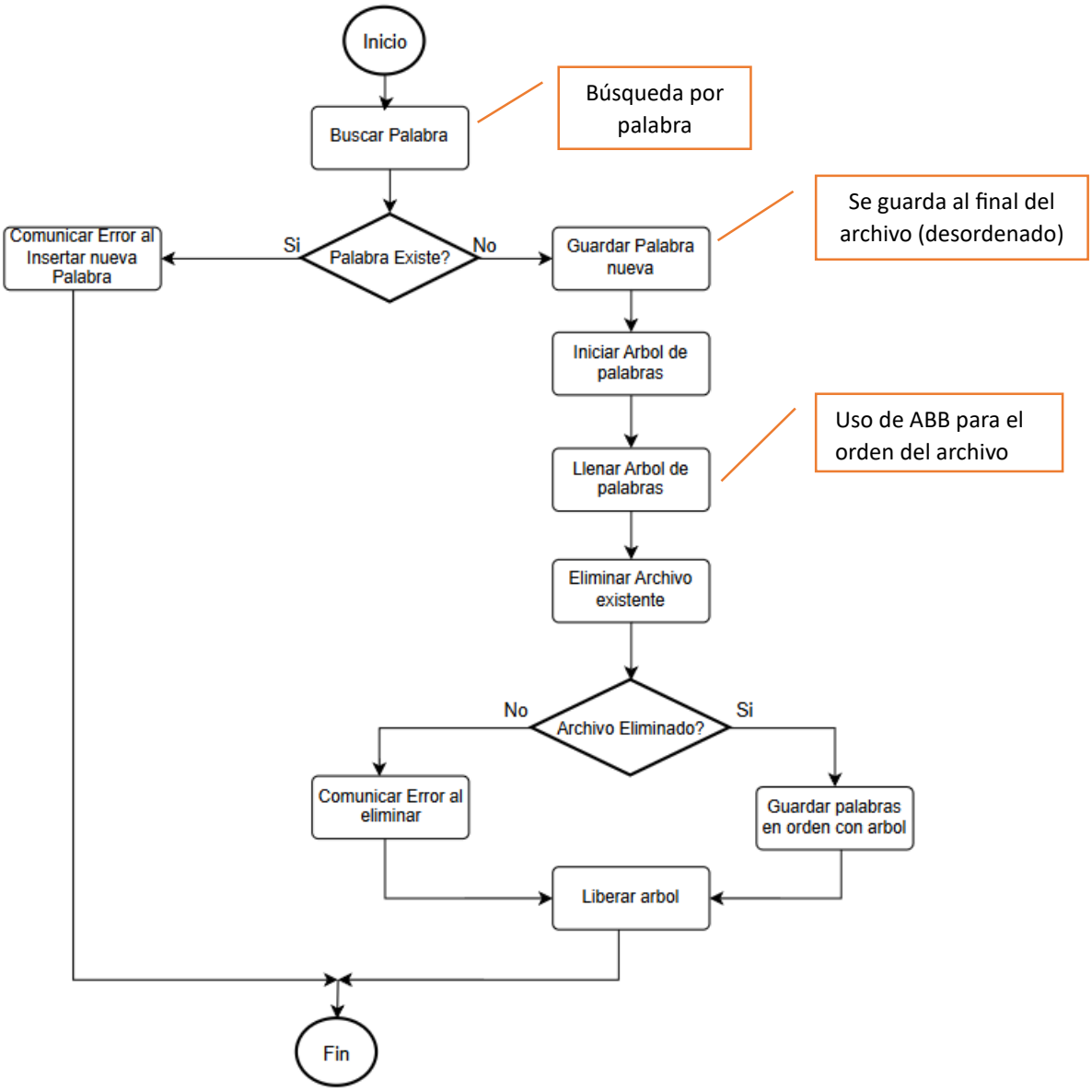
Consideraciones y aportes que ayudarán a entender la lógica

Insertar Jugador



Insertar Palabra

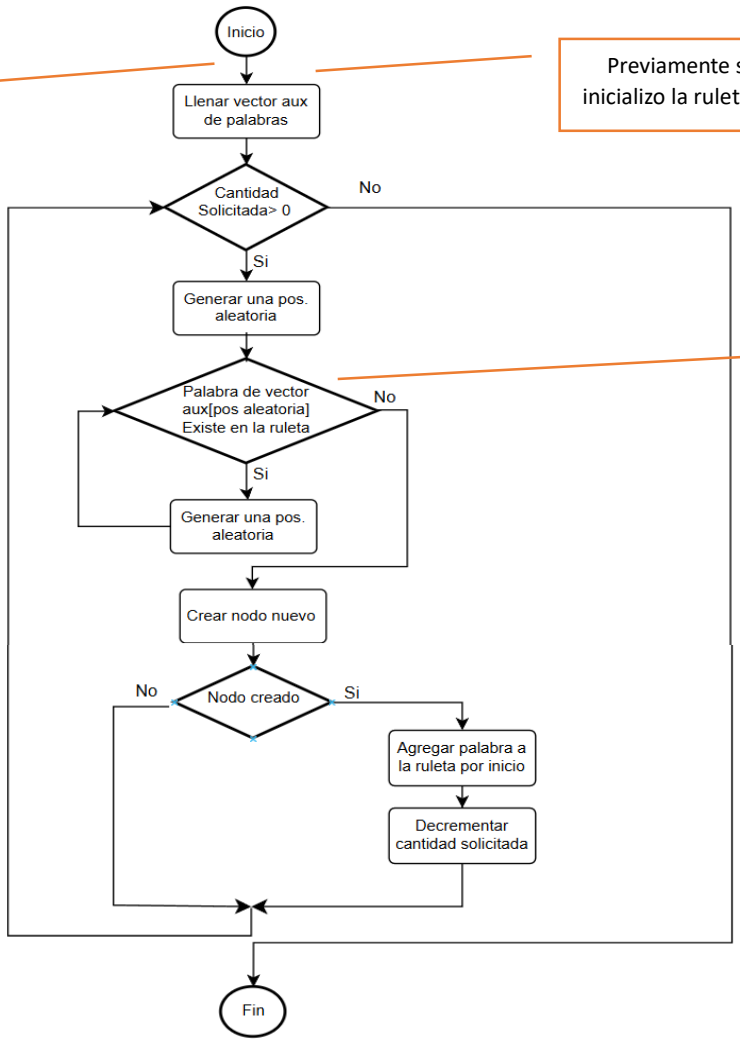
Sigue la misma estructura de insertar jugador.



Generar Ruleta

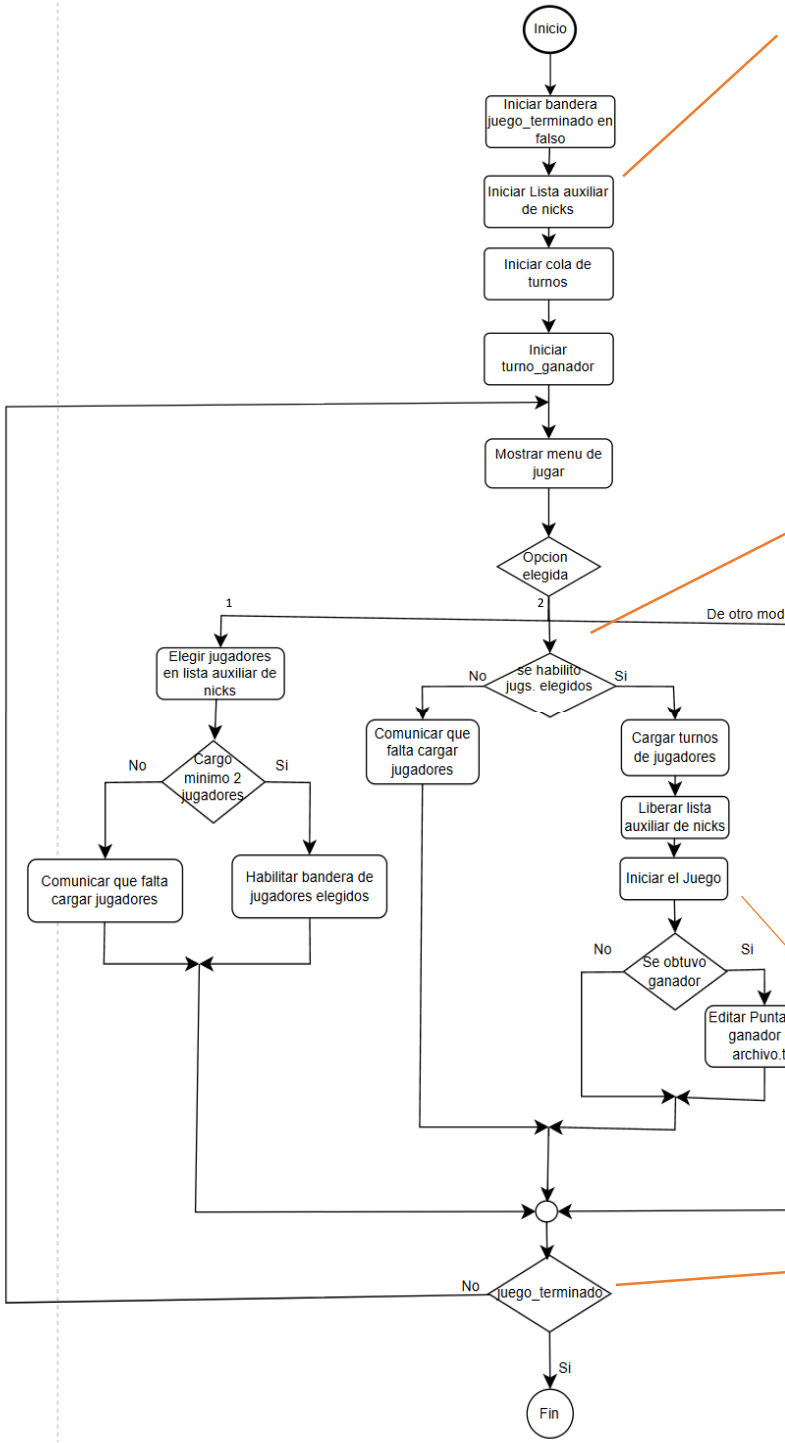
El usuario ya ingresó una cantidad solicitada de palabras para generar la ruleta entre 5 y cant_máx_aux_vec_rul…

Previamente se generó e inicializo la ruleta (en el main)



Va consultando si la palabra existe en la ruleta, si existe genera otra

Gestión Jugar

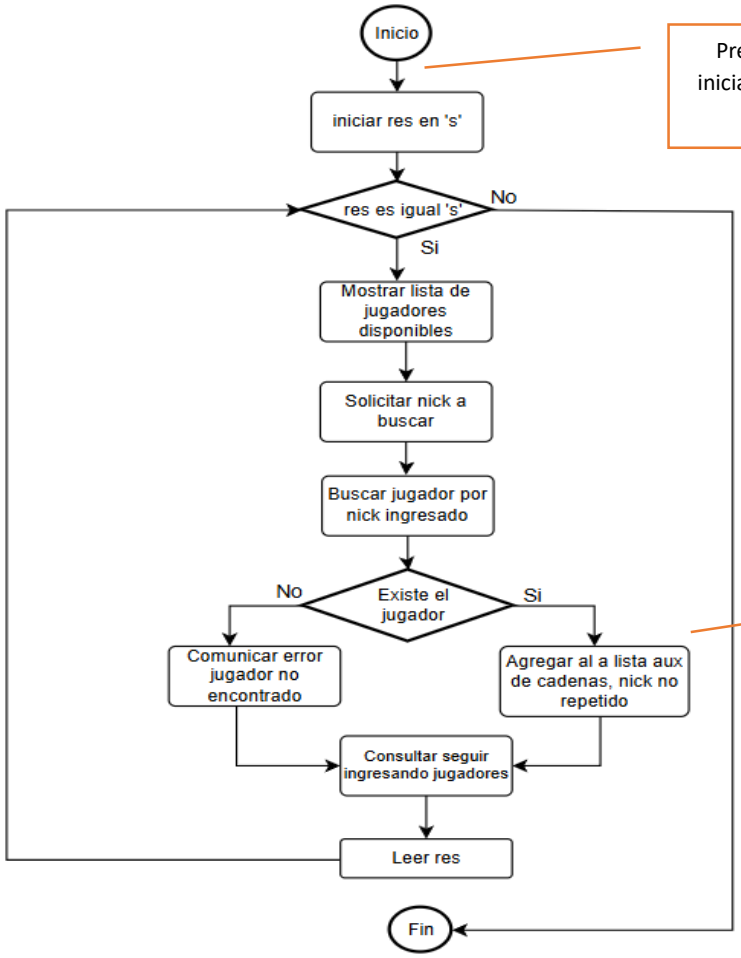


En el modulo de gestión jugar se inicializa la cola de turnos de jugadores y la lista auxiliar para cargar nicks no repetidos, posterior carga se libera la lista

Solo podemos jugar si al menos hay cargados 2 jugadores en la lista auxiliar

Variable juego_terminado se actualiza al finalizar el juego

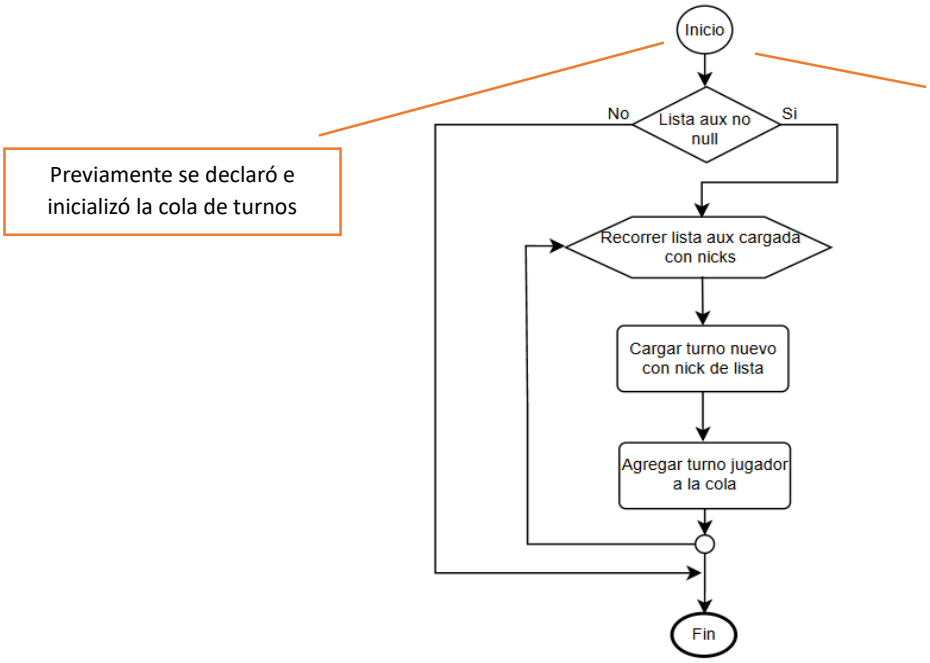
Elegir Jugadores para jugar la partida



Previamente se generó e inicializo la lista aux (modulo jugar)

Crea un nodo, si el jugador existe comunica que está en la lista y elimina el nodo, caso contrario se inserta

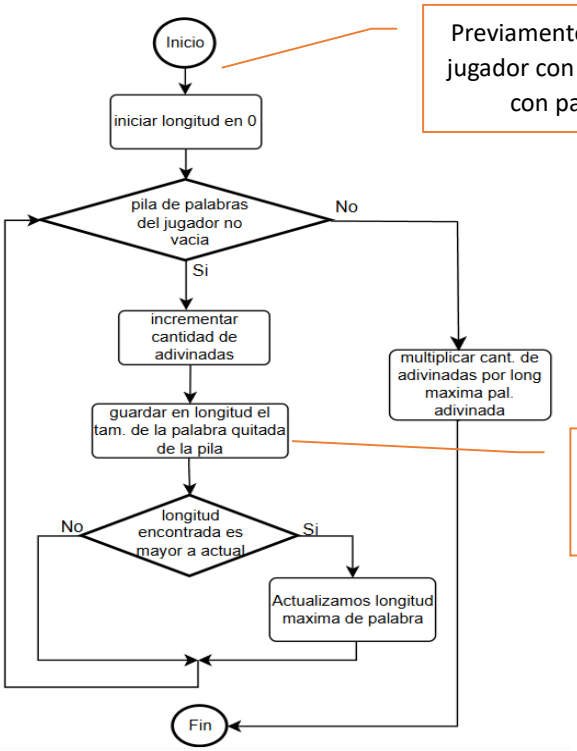
Cargar turnos de jugadores



Previamente se declaró e inicializó la cola de turnos

Previamente se Cargó la lista auxiliar

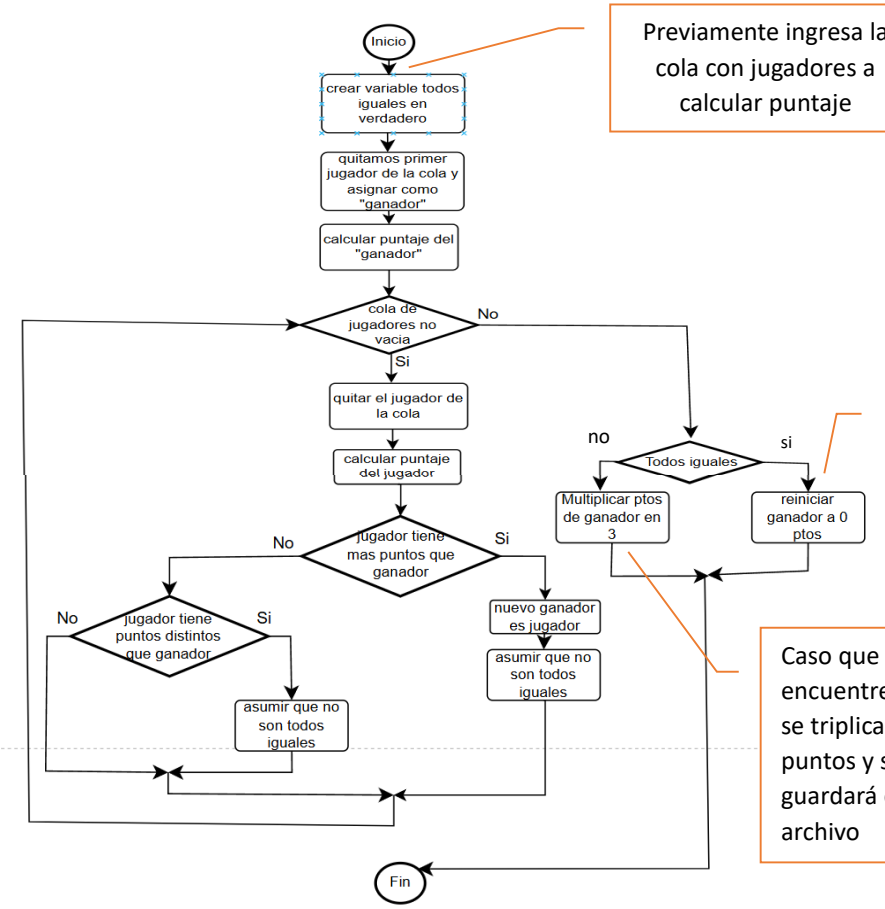
Calcular puntaje de jugador



Previamente ingresa un jugador con pila cargada con palabras

Por cada longitud calculada, quitamos la palabra de la pila

Verificar Ganador



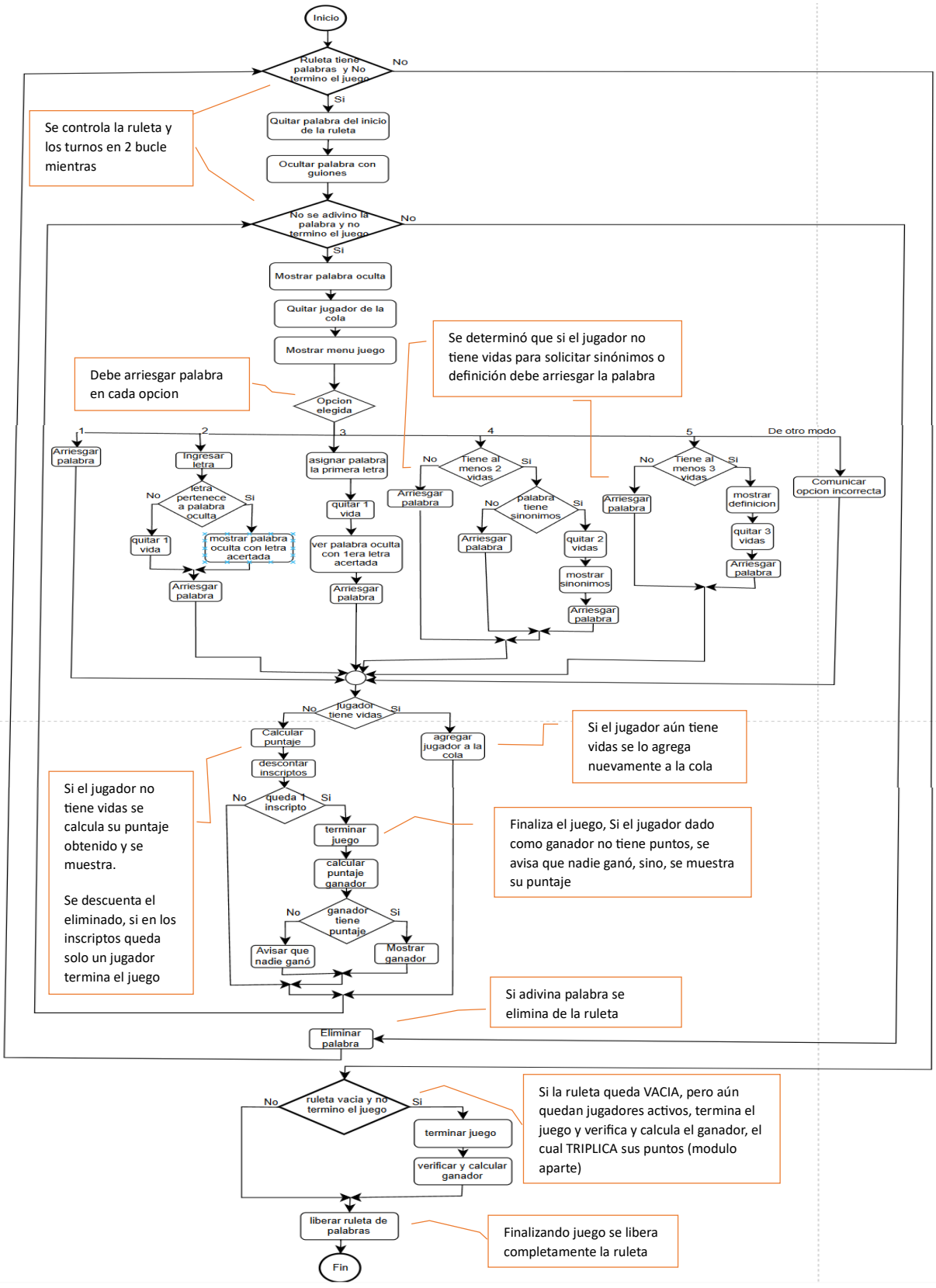
Previamente ingresa la cola con jugadores a calcular puntaje

Al reiniciar puntos a 0 se asume que nadie gano, por tener todos el mismo puntaje, y no se modifica puntaje en el archivo

Caso que se encuentre ganador se triplican sus puntos y se guardará en el archivo

Modulo Jugar

Utiliza ruleta de palabras y cola de turnos de jugadores y determina el ganador, y módulos anteriores diseñados



Pruebas, Observaciones, Consideraciones y casos de error que encontré en el desarrollo

Durante el desarrollo del programa se fueron dando ciertos casos de error y consideraciones que se arreglaron en el código, las cuales serán detalladas a continuación.

ERROR: Inserción no liberaba correctamente el arbol si surge error en guardado

```
Modulo que permite insertar el nuevo jugador siguiendo una secuencia de pasos para poder hacer la i
ordenado
*/
void insertar_jugador(parchivo fp, jugador j)
{
    pjugador arboljug; //Declaramos el arbol de jugadores para ordenar

    if(!search_file_jugador(fp, j.nickname)) //buscamos que no exista el nickname en el archivo
    {
        save_file_jugador(fp, j); //si no existe se guarda el nuevo nickname

        iniciar_arb_jugadores(arboljug); //inicializamos el arbol

        llenar_arbol_jugadores(fp, arboljug); //llenamos el arbol con los jugadores con insercion s

        if(remove("jugadores.txt") == 0) //eliminamos el archivo actual (desordenado)
        {

            guardar_jugador_en_orden(fp, arboljug);  //recorremos el arbol en orden y vamos rellena
            cout<<"Se agregó jugador al banco"<<endl;
            liberar_arbol(arboljug); //una vez se carga archivo en orden se libera el arbol
        }
        else
        {
            cout << "error al eliminar" << endl;
        }
    }
    else
        cout << "Jugador ya existe" << endl;
}
```

SOLUCION: Mover el módulo de liberación al final del proceso para que siempre libere el árbol correctamente

```
void insertar_jugador(parchivo fp, jugador j)
{
    pjugador arboljug; //Declaramos el arbol de jugadores para ordenar

    if(!search_file_jugador(fp, j.nickname)) //buscamos que no exista el nickname en el a
    {
        save_file_jugador(fp, j); //si no existe se guarda el nuevo nickname

        iniciar_arb_jugadores(arboljug); //inicializamos el arbol

        llenar_arbol_jugadores(fp, arboljug); //llenamos el arbol con los jugadores con i

        if(remove("jugadores.txt") == 0) //eliminamos el archivo actual (desordenado)
        {

            guardar_jugador_en_orden(fp, arboljug);  //recorremos el arbol en orden y vam
            cout<<"Se agregó jugador al banco"<<endl;
        }
        else
        {
            cout << "error al eliminar" << endl;
        }
        liberar_arbol(arboljug); //una vez se finaliza el proceso se libera el arbol
    }
    else
        cout << "Jugador ya existe" << endl;
}
```

Error: Se eligió MALA ESTRUCTURA DE PILA de palabras en turnos de jugador (solicitaba pila dinámica).

```
typedef palabra tcont_palabras[MAX_VECTOR_AUX_RULETA];

typedef struct tpila
{
    tcont_palabras datos;
    int cima;
};
```

```
void iniciar_pila(tpila &pila)
{
    pila.cima = -1;
}

bool pila_vacia(tpila pila)
{
    return pila.cima == -1;
}

bool pila_llena(tpila pila)
{
    return pila.cima == MAX_VECTOR_AUX_RULETA - 1;
}

void agregar_pila(tpila &pila, palabra valor)
{
    if(pila_llena(pila))
        cout << "Pila de palabras llena" << endl;
    else
    {
        pila.cima++;
        pila.datos[pila.cima] = valor;
    }
}

palabra quitar_pila(tpila &pila)
{
    palabra extraido;
    if(pila_vacia(pila))
        strcpy(extraido.cad, "@");
    else
    {
        extraido = pila.datos[pila.cima];
        pila.cima--;
    }

    return extraido;

}

palabra tope_pila(tpila pila)
{
    palabra extraido;
    if(pila_vacia(pila))
        strcpy(extraido.cad, "@");
    else
    {
        extraido = pila.datos[pila.cima];
    }

    return extraido;
}
```

SOLUCION: Se tuvo que REESTRUCTURAR LA PILA Y LOS MODULOS Y FUNCIONES que la ocupaban por una pila dinámica implementada con listas dobles.

```
typedef struct tnodo_p *pnodo_p;

typedef struct tnodo_p {
    palabra dato;
    pnodo_p ant;
    pnodo_p sig;
};

typedef struct tpila {
    pnodo_p inicio;
    int cont;
};
```

```
void iniciar_pila(tpila &pila)
{
    pila.inicio=NULL;
    pila.cont=0;
}

//Creación del nodo
void crear(pnodo_p &nuevo, palabra valor)
{
    nuevo=new tnodo_p;
    if(nuevo!=NULL)
    {
        nuevo->dato=valor;
        nuevo->ant=NULL;
        nuevo->sig=NULL;
    }
}

bool pila_llena(tpila lis) {
    return lis.cont == MAX_VECTOR_AUX_RULETA
}

bool pila_vacia(tpila pila)
{
    return pila.cont==0;
}
```

Se rehicieron operaciones para la pila dinámica de palabras…

ERROR: Al momento de quitar el jugador de la cola o de finalizar el juego, no se terminó de limpiar su pila de palabras.

```
            default:
                cout << "Opción incorrecta" << endl;
                system("pause");
            }

            if (jugador.vidas > 0)
                agregar_cola(turnos_jugadores, jugador);
            else
            {
                system("cls");
                cout << jugador.nick << " quedo fuera del juego." << endl;
                cantidad_jugadores--;
                if (cantidad_jugadores == 1)
                {
                    juego_terminado = true;

                    ganador = quitar_cola(turnos_jugadores);

                    calcular_puntaje(ganador);

                    if(ganador.puntos>0){
                        cout << "El ganador del juego es: " << ganador.nick << " con: "<< ganador.puntos<<" puntos !!" << endl;


                    }else
                      cout << "\nNingún jugador adivino las palabras." << endl;

                }
            }
        }
        delete palabra;
        palabra = NULL;
    }
    /*
```

SOLUCION Usar modulo calcular puntaje para ir desapilando palabras y eliminando cada nodo mientras no esté vacia la pila

```
        }else
        {
            system("cls");

            calcular_puntaje(jugador);

            cout << jugador.nick << " quedo fuera del juego con "<<jugador.puntos<<" puntos obtenidos"<< endl;

            jugadores_inscriptos--;
            if (jugadores_inscriptos == 1)
            {
                juego_terminado = true;

                ganador = quitar_cola(turnos_jugadores);

                calcular_puntaje(ganador);

                if(ganador.puntos>0){
                    cout << "El ganador del juego es: " << ganador.nick << " con: "<< ganador.puntos<<" puntos !!" << en

                }else
                   cout << "\nNingún jugador adivino las palabras." << endl;

            }
        }
    }
```

```
palabra quitar_pila(tpila &lis){
    pnodo_p extraido, i;
    palabra ext;

    if (pila_vacia(lis))//(lis.inicio==NULL)
        strcpy(ext.cad, "@");
    else
    {
        if(lis.inicio->sig==NULL){
            extraido=lis.inicio;
            lis.inicio=NULL;
        }else{
            for(i=lis.inicio;(i->sig)->sig!=NULL;i=i->sig);
            extraido=i->sig;
            i->sig=NULL;
            extraido->ant=NULL;
        }
        lis.cont--;
        ext=extraido->dato;
        delete(extraido);
        extraido=NULL;
    }
    return ext;
}
```

SE ENCONTRO VARIABLE INUTILIZADA “nuevo” al momento de cargar palabras, ya que no se realizó la operación de modificación de palabras, la variable no es requerida

```
/**
 La operacion de carga permite cargar manualmente los campos, por cada campo va validando que cumpla min. de caracteres
 Utiliza una validacion de libreria util (verificacion_cadena) para verificar que sea cadena valida
 */
void cargar_palabra(palabra &p, bool nuevo)  //nuevo true = agregar , false = modificar
{
    bool valido = false;
    int minimoCaracteres = 3;
    char res;

    system("cls");

    do
    {
        cout << "Ingrese Palabra: ";
```

Solución, debe eliminarse la variable del modulo y eliminarse de todas las llamadas al modulo de crear palabra donde se invoque la variable

```
/*
 La operacion de carga permite cargar manualmente los cam
 Utiliza una validacion de libreria util (verificacion_ca
 */
void cargar_palabra(palabra &p)
if(contar_palabras_banco(f)<MAX_VECTOR_AUX_RULETA){
    cargar_palabra(nuevo);
    insertar_palabra(f, nuevo);
}else{
    cout<<"Se superó el maximo de palabras insertables"<<
}
```

Error al momento de limpiar la palabra luego de la inserción similar al de JUGADOR, NO libera correctamente el árbol de palabras

```
save_file_pal(fp, p); //si no existe se guarda

iniciar_arb_palabras(arbolpal); //inicializamo

llenar_arbol_pal(fp, arbolpal); //llenamos el

if(remove("palabra.txt") == 0) //eliminamos el
{

    guardar_palabra_en_orden(fp, arbolpal);
    cout<<"Se agregó palabra al banco"<<endl;
    liberar_arbol(arbolpal); //una vez se carg
}
else
{
    cout << "error al eliminar" << endl;
}
```

Solución mover el liberar árbol de palabras al final del proceso

```
if(!search_file_pal(fp, p.cad)) //buscamos que no exista palabra en el archivo
{
    save_file_pal(fp, p); //si no existe se guarda nueva palabra

    iniciar_arb_palabras(arbolpal); //inicializamos el arbol

    llenar_arbol_pal(fp, arbolpal); //llenamos el arbol con los palabras con inser

    if(remove("palabra.txt") == 0) //eliminamos el archivo actual (desordenado)
    {

        guardar_palabra_en_orden(fp, arbolpal);  //recorremos el arbol en orden y
        cout<<"Se agregó palabra al banco"<<endl;
    }
    else
    {
        cout << "error al eliminar" << endl;
    }
    liberar_arbol(arbolpal); //una vez se finaliza el proceso se libera el arbol
```

ERROR se encontró al momento de crear la pila y verificar que esté llena, si la pila esta llena el nodo quedó creado, pero no se liberó su memoria

```
void agregar_pila(tpila &lis, palabra valor){
    pnodo_p i, nuevo;

    crear(nuevo,valor);
    if(pila_llena(lis))
        cout<<"Pila llena"<<endl;
    else{
        if(lis.inicio==NULL)
            lis.inicio=nuevo;
        else
        {
            for(i=lis.inicio;i->sig!=NULL;i=i->sig);
            i->sig=nuevo;
            nuevo->ant=i;
        }
        lis.cont++;
    }
}
```

Solucion Se agrega delete al nodo creado de modo que la memoria quede liberada si no se pudo agregar a la pila.

```
void agregar_pila(tpila &lis, palabra valor){
    pnodo_p i, nuevo;

    crear(nuevo,valor);
    if(pila_llena(lis)){
        cout<<"Pila llena"<<endl;
        delete(nuevo);
        nuevo=NULL;
    }
    else{
        if(lis.inicio==NULL)
            lis.inicio=nuevo;
        else
```