INFORME PROYECTO TA-TE-TI

Integrantes: Aitor Ortuño(113603) -Nahuel Maika Fanessi(113642)

TDA_Lista

Estructura utilizada para representar la lista

Se utiliza una lista simplemente enlazada con celda centinela utilizando posición indirecta.

Operaciones del TDA

- **Void crear_lista(tLista *l)**
  - **¿Qué hace? : Inicializa una lista vacía**
  - **¿Cómo lo hace? : Recibe por parámetro un puntero a una tLista (un puntero a una estructura Nodo). Guarda lugar en memoria para una estructura Nodo y asigna a su elemento y a su puntero a la siguiente posición como nulos, creando el nodo centinela**
- **Void l_insertar(tLista l, tPosicion p, tElemento e)**
  - **¿Qué hace? : Inserta el elemento 'e' en la lista 'l' después de 'p'.**
  - **¿Cómo lo hace? : Recibe por parámetro una tLista (puntero a una estructura Nodo), una tPosicion (puntero a una estructura Nodo) y un tElemento (puntero a void). Guarda lugar en memoria para una estructura Nodo y asigna a su elemento el tElemento 'e' y a su siguiente el siguiente de p, actualizando luego el siguiente de p como la nueva posición creada.**
- **Void l_eliminar(tLista l, tPosicion p, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Elimina el elemento en la posición 'p' de la lista 'l' con la función fEliminar pasada por parámetro.**
  - **¿Cómo lo hace? : Recibe por parámetro una tLista, una tPosicion y una función fEliminar. Actualizo el siguiente de 'p' como el siguiente de su siguiente, elimino el elemento del siguiente a la posición 'p' con la función fEliminar y por ultimo libero el espacio de memoria que tenía la posición a eliminar.**
- **Void l_destruir(tLista l, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Destruye cada elemento de la lista 'l' con la función fEliminar.**
  - **¿Cómo lo hace? : Guarda la primer posición de la lista y llama a un método recursivo el cual se encarga de recorrer la lista 'l' y eliminar desde atrás hacia adelante cada elemento de la misma asignando como nulo a su siguiente, eliminando el elemento con la función fEliminar y asignándole nulo, y por ultimo liberando el espacio de memoria que tenía la posición a eliminar.**
- **tElemento l_recuperar(tLista l, tPosicion p)**
  - **¿Qué hace? : Recupera el elemento que se encuentra en la posición 'p' en la lista 'l'.**

- **¿Cómo lo hace? : Devuelve el elemento del siguiente de 'p' ya que utiliza posición indirecta.**

- **tPosicion l_primera(tLista l)**
  - **¿Qué hace? : Retorna la primera posición de la lista 'l'.**
  - **¿Cómo lo hace? : Devuelve el nodo centinela de la lista 'l' ya que utiliza posición indirecta.**
- **tPosicion l_ultima(tLista l)**
  - **¿Qué hace? : Retorna la última posición de la lista 'l'**
  - **¿Cómo lo hace? : Recorre la lista 'l' y devuelve la anteúltima posición de la misma ya que utiliza posición indirecta.**
- **tPosicion l_fin(tLista l)**
  - **¿Qué hace? : Retorna la posición fin de la lista 'l'**
  - **¿Cómo lo hace? : Recorre la lista 'l' y devuelve la última posición de la misma ya que utiliza posición indirecta.**
- **tPosicion l_siguiente(tLista l, tPosicion p)**
  - **¿Qué hace? : Retorna la posición siguiente a 'p'**
  - **¿Cómo lo hace? : Devuelve el siguiente de 'p'.**
- **tPosicion l_anterior(tLista l,tPosicion p)**
  - **¿Qué hace? : Retorna la posición anterior a 'p'**
  - **¿Cómo lo hace? : Recorre la lista 'l' hasta encontrar el anterior a 'p' y lo devuelve.**
- **int l_longitud(tLista l)**
  - **¿Qué hace? : Retorna la longitud de 'l'**
  - **¿Cómo lo hace? : Recorre la lista 'l' mientras que el siguiente del actual no sea nulo y tiene un contador, cuando deja de recorrer la lista, devuelve el contador.**

Operaciones Auxiliares

TDA_Arbol

Estructura utilizada para representar el árbol

Se utiliza una representación de nodos enlazados donde cada nodo mantiene una referencia con su nodo padre, con una lista de nodos, la cual es su lista de hijos, y con su propio elemento, que es su rotulo.

Operaciones del TDA

- **Void crear_arbol(tArbol *a)**
  - **¿Qué hace? : Inicializa un árbol vacío**
  - **¿Cómo lo hace? : Recibe por parámetro un puntero a un tArbol (un puntero a una estructura Árbol). Guarda lugar en memoria para una estructura Árbol y asigna a su raíz como nula.**
- **Void crear_raíz(tArbol a, tElemento e)**
  - **¿Qué hace? : Inserta el elemento 'e' en el árbol 'a'.**
- **¿Cómo lo hace? : Recibe por parámetro una tArbol y un tElemento (puntero a void). Guarda lugar en memoria para una estructura Nodo y asigna a su elemento el tElemento 'e', inicializa la lista de hijos y asigna a su padre como nulo.**
- **tNodo a_insertar(tArbol a, tNodo np,tNodo nh,tElemento e)**
  - **¿Qué hace? : Inserta en el arbol 'a' en la lista de hijos de 'np' y como hermano izquierdo de 'nh' el elemento 'e'.**
  - **¿Cómo lo hace? : Guarda lugar en memoria para una estructura Nodo y asigna a su elemento el tElemento 'e', asigna a su padre como np e inicializa la lista de hijos. Si nh es nulo, lo agrega al final de la lista de hijos de np. Si nh no es nulo, recorre la lista de hijos de np y lo agrega en la posición anterior a nh.**
- **Void a_eliminar(tArbol a, tNodo n, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Elimina del arbol 'a' el nodo 'n' con la función fEliminar.**
  - **¿Cómo lo hace? : Si 'n' es la raíz de 'a' y no tiene hijos, elimino la raíz, si tiene un hijo, elimino la raíz y su hijo pasa a ser la nueva raíz. Si 'n' no es raíz, agrego los hijos de 'n' a la lista de hijos de su padre en la posición anterior a 'n' , luego elimino 'n' de la lista de hijos de su padre y por ultimo elimino el nodo 'n'.**
- **Void a_destruir(tArbol *a, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Destruye el árbol 'a' eliminando cada uno de sus nodos.**
  - **¿Cómo lo hace? : Llama recursivamente a una función privada 'eliminar' que recorre el árbol en profundidad y elimina cada nodo del árbol utilizando una función llamada 'fNodoEliminar' que usa la función fEliminar para eliminar el elemento del nodo, luego asigna su lista de**

hijos como nulo, asigna su padre como nulo y por ultimo libera el espacio de memoria reservado para el nodo que elimina.

- **tElemento a_recuperar(tArbol a, tNodo n)**
  - **¿Qué hace? : Retorna el elemento del nodo 'n' en el árbol 'a'**
  - **¿Cómo lo hace? : Retorna el elemento del nodo 'n'.**
- **tNodo a_raiz(tArbol a)**
  - **¿Qué hace? : Retorna la raíz del árbol 'a'**
  - **¿Cómo lo hace? : Retorna la raíz de 'a'.**
- **tLista a_hijos(tArbol a, tNodo n)**
  - **¿Qué hace? : Retorna la lista de hijos del nodo 'n' en el árbol 'a'.**
  - **¿Cómo lo hace? : Retorna la lista de hijos de 'n'.**
- **Void a_sub_arbol(tArbol a l, tNodo n, tArbol *sa)**
  - **¿Qué hace? : Asigna como raíz del árbol 'sa' al nodo 'n' y elimina del árbol 'a' a 'n' y todos sus hijos.**
  - **¿Cómo lo hace? : Inicializa el árbol 'sa', si 'n' es la raíz de 'a', 'n' pasa a ser la raíz de 'sa' y le asigna nulo a la raíz de 'a'. Si 'n' no es la raíz de 'a', asigna a 'n' como la raíz de 'sa' y recorre la lista de hijos del padre de 'n' y lo elimina de dicha lista.**

Operaciones Auxiliares

Juego TA-TE-TI

Introducción

Se desarrolló el juego TA-TE-TI y permite 2 modos de juego:
-Jugador1 VS Jugador2
-Jugador Vs Maquina

Módulos Desarrollados

Partida

- ***Funcionalidad:*** ***Permite la creación de una nueva partida, realizar un nuevo movimiento y finalizar la partida.***
- ***Responsabilidad:*** ***Mantener actualizado el estado de la partida, ya sea partida en juego, gana algún jugador o hay empate.***
  También controla que los nuevos movimientos realizados sean válidos.
- ***Fuentes: partida.c y partida.h***
- ***Estrategia Utilizada:***
- ***Operaciones***
  - ***void nueva_partida(tPartida * p, int modo_partida, int comienza, char * j1_nombre, char * j2_nombre)***
    - ***¿Qué hace? : Inicializa una nueva partida, indicando que modo de partida va a ser, que jugador comienza y los nombres de ambos jugadores.***
    - ***¿Cómo lo hace? :***
  - ***int nuevo_movimiento(tPartida p, int mov_x, int mov_y)***
    - ***¿Qué hace? : Actualiza el movimiento hecho por el jugador de turno (si dicho movimiento es válido), el estado de la partida, a que jugador le corresponde el siguiente turno y retorna el estado actual de la partida.***
    - ***¿Cómo lo hace? :***
  - ***void finalizar_partida(tPartida * p)***
    - ***¿Qué hace? : Finaliza la partida***
    - ***¿Cómo lo hace? : Libera el espacio de memoria al cual referencia p.***
- ***Operaciones Auxiliares***

➢ IA

- ***Funcionalidad:*** ***Permite la creación de una nueva IA, encontrar cual es el mejor movimiento próximo a realizar y destruir la IA.***
- ***Responsabilidad:***
- ***Fuentes:*** ***ia.c y ia.h***
- ***Estrategia Utilizada:*** ***MIN_MAX***
- ***Operaciones***
  - ***void crear_busqueda_adversaria(tBusquedaAdversaria * b, tPartida p)***
    - ***¿Qué hace? : Inicializa una búsqueda adversaria, clonando el estado del tablero de la partida, inicializando un nuevo árbol de búsqueda adversaria y luego hacer uso del algoritmo min_max para generar el árbol.***
    - ***¿Cómo lo hace? :***
  - ***void proximo_movimiento(tBusquedaAdversaria b, int * x, int * y)***
    - ***¿Qué hace? : Computa y retorna el próximo movimiento a realizar por el jugador MAX utilizando el árbol de búsqueda adversaria creado siguiendo el algoritmo min_max.***
    - ***¿Cómo lo hace? :***
  - ***void destruir_busqueda_adversaria(tBusquedaAdversaria * b)***
    - ***¿Qué hace? : Libera el espacio asociado a la estructura correspondiente para la búsqueda adversaria***
    - ***¿Cómo lo hace? : Libera el espacio de memoria al cual referencia p.***
- ***Operaciones Auxiliares***

Modo de Ejecución

Modo de Uso

1. Elegir modo de juego seleccionando ‘1’ (Jugador1 VS Jugador2) o ‘2’ (Jugador VS Maquina).
2. Dependiendo del modo se pedirá únicamente el nombre del Jugador o Jugadores.
3. Elegir quien comienza, ‘1’ (Jugador1), ‘2’ (Jugador2) o 3(Random).
4. Cuando sea el turno de cualquier jugador que no sea la maquina deberá seleccionar una fila (entre 1 y 3) y una columna (entre 1 y 3).
5. Repetir el paso 4 hasta que la partida finalice con un empate o con un ganador.
6. Saldrá un mensaje dependiendo de cómo haya finalizado la partida y se deberá apretar cualquier tecla para terminar el juego.

Conclusiones

Respecto al código, quedo sin solucionar un problema que es, cuando termina la partida esta finaliza correctamente pero lanza un error de que se liberó un espacio de memoria que ya había sido liberado o que no tiene nada. Estuvimos tratando de encontrar el error, pero luego de varias horas de debugging no lo pudimos solucionar.
Además no pudimos hacer las librerías correspondientes.

Respecto a la documentación, por falta de tiempo nuestra no pudimos terminar de documentar completamente como nos gustaría. Falta definir las operaciones auxiliares utilizadas en la IA, PARTIDA, TDA_ARBOL y TDA_LISTA, así como algún diagrama de flujo o pseudocódigo para representar como se relacionan las distintas funciones entre ellas.