Carátula

Rios Matias.

Rychert Alan.

TDA Lista

Para representar la lista se utilizan celdas simplemente enlazadas y una celda centinela de encabezamiento, haciendo uso del concepto de posición indirecta, es decir, la posición se denota mediante un puntero a una celda que contiene un puntero a la celda que tiene el elemento deseado, y la lista es un puntero a una celda vacía que representa el centinela.

Dibujitos pongo después.

Operaciones del TDA:

void crear\_lista(tLista\* l):

Inicializa una lista vacía reservando y asignándole un lugar en la memoria dinámica. Como la lista está vacía, se inicializan los campos del struct en nulo.

tPosicion l\_primera(tLista l):

Recupera y retorna la primer posición de la lista. Si la lista está vacía, primera(l), ultima(l) y fin(l) son iguales. La primer posición es el resultado de castear a tPosicion la lista recibida por parámetro (el centinela).

tPosicion l\_ultima(tLista l):

Recupera y retorna la ultima posición de la lista. Si la lista está vacía, primera(l), ultima(l) y fin(l) son iguales. Para conseguir la ultima posición se recorren todas las posiciones de la lista hasta encontrarla.

void l\_destruir(tLista\* l,void (\*fEliminar)(tElemento)):

Destruye la lista l y elimina cada una de sus celdas y sus respectivos elementos. Los elementos son eliminados mediante la función fEliminar parametrizada, la cual debe ser implementada por el cliente de la lista.  
Al destruir la lista se liberan todos los espacios de la memoria dinámica que fueron reservados al momento de crear la lista o de insertar nuevos elementos.

void l\_insertar(tLista l, tPosicion p, tElemento e):

Inserta el elemento e en la lista l, en la posición p. Esto quiere decir que luego de insertar, el elemento contenido en la posición p ahora será e, y el elemento antes contenido en ella se encontrará en la siguiente de p (debido a que se trabaja con posición indirecta).

void l\_eliminar(tLista l, tPosicion p, void (\*fEliminar)(tElemento)):

Elimina la celda P de L, liberando el espacio en la memoria dinámica que había sido reservado al momento de insertar o crear la lista. El elemento almacenado en la posición P es eliminado mediante la función fElimina parametrizada, que debe ser implementada por el cliente de la lista. Si P es fin(l), finaliza indicando LST\_POSICION\_INVALIDA.

tElemento l\_recuperar(tLista l, tPosicion p):

Recupera y retorna el elemento contenido en la posición P (es decir, el elemento que se encuentra en la celda siguiente de la celda apuntada por P, debido a que se trabaja con posición indirecta). Si P es fin(l), finaliza indicando LST\_POSICION\_INVALIDA.

tPosicion l\_siguiente(tLista l, tPosicion p):

Recupera y retorna la posición siguiente a P en la lista, es decir, la celda siguiente a la celda apuntada por P. Si P es fin(l), finaliza indicando LST\_POSICION\_INVALIDA.

tPosicion l\_anterior(tLista l, tPosicion p):

Recupera y retorna la posición anterior a P en la lista, recorriendo la lista hasta encontrar la posición cuya siguiente posición es P. Si P es igual a l\_primera(l), finaliza indicando LST\_NO\_EXISTE\_ANTERIOR.

tPosicion l\_fin(tLista l):

Recupera y retorna la posición fin de l. Si la lista está vacía, l\_primera(l), l\_ultima(l) y l\_fin(l) son iguales. Para conseguir la posición fin, se recorre toda lista hasta encontrar una posición cuya siguiente posición sea nula.

int l\_longitud(tLista l):

Retorna la longitud actual de la lista, la cual es computada recorriendo toda la lista y aumentando un Contador por cada posición que la compone.

Modulos:

Partida.c: el módulo partida.c consiste en dos tipos de registros y diferentes operaciones con las cuales se pueden manipular los mismos para representar lógicamente una partida de Ta-Te-Ti. Los registros son partida y tablero.  
Una partida es un registro que contiene distintos campos:  
tres campos que contienen enteros (modo\_partida,turno\_de, y estado), un campo que contiene un struct (tablero), y dos campos que contienen caracteres(nombre\_jugador\_1 y nombre\_jugador\_2).  
Un tablero es un registro que contiene como único campo una matriz de enteros de 3 filas por 3 columnas en la cual se verá reflejado el estado actual de la grilla de la partida.

Operaciones del módulo partida:

void nueva\_partida(tPartida \* p, int modo\_partida, int comienza, char \* j1\_nombre, char \* j2\_nombre):

inicializa una nueva partida, recibiendo por parámetro el modo de la partida (Usuario contra Usuario o Usuario contra Agente IA), jugador que comienza la partida (jugador 1, jugador 2, o al azar), y el nombre de ambos jugadores. Para esto se reservan los espacios necesarios en memoria dinámica, se crea un struct partida en ese espacio y se inicializan los campos de la partida con los datos parametrizados. Tambien es reservado espacio en memoria dinámica para poder inicializar el campo tablero en la partida.

int nuevo\_movimiento(tPartida p, int mov\_x, int mov\_y):

Actualiza, si corresponde, el estado de la partida, teniendo en cuenta que el jugador coloca una ficha en la fila x, columna y, retornando un entero que representa si la jugada fue valida o no dependiendo de lo que corresponde.

void finalizar\_partida(tPartida \* p):

Finaliza la partida referenciada por P, liberando toda la memoria utilizada, es decir, la reservada para el tablero y la reservada para la partida. También lo apuntado por P es seteado en nulo.

Main.c: el módulo main.c se encarga de la ejecución propiamente dicha del juego, y consiste en las siguientes operaciones que permiten el desarrollo del juego:

Operaciones:

int main():

Permite al usuario ingresar los datos necesarios para iniciar una partida de Ta-Te-Ti, es decir, el modo de juego, quien comienza jugando, y el/los nombres del/de los jugadores. Luego, crea la partida teniendo en cuenta lo ingresado por el usuario.

void humanoVsHumano(char jugador1[50],char jugador2[50],int jugadorInicial):

Desarrolla una partida de Ta-Te-Ti en modo PART\_MODO\_USUARIO\_VS\_USUARIO haciendo el control de turnos correspondientes para permitir que los jugadores realicen las jugadas que desean, ingresando por teclado la fila y columna donde desean insertar una ficha. No permite avanzar de turno en caso de que la jugada no sea válida, solicitando que se ingrese nuevamente una jugada hasta que la misma sea valida, y luego de cada jugada, controla el estado de la partida, para, en caso de que haya finalizado, indicar quien ganó o si hubo empate, y si no, continuar con la partida. Luego de cada jugada realizada muestra el estado del tablero por pantalla.

void humanoVsMaquina(char jugador1[50],int jugadorInicial):

Desarrolla una partida de Ta-Te-Ti en modo PART\_MODO\_USUARIO\_VS\_AGENTE\_IA, haciendo el control de turnos correspondientes para permitir que se desarrollen los turnos como corresponde. Si el usuario intenta realizar una jugada incorrecta no avanza de turno, solicitando que se ingrese nuevamente una jugada hasta que la misma sea valida, luego de esto pide al Agente IA que realice su movimiento y controla el estado de la partida, para, en caso de que haya finalizado, indicar quien ganó o si hubo empate, y si no, continuar con la partida. Luego de cada turno (ya sea del jugador o del Agente IA) muestra el estado del tablero por pantalla.

int esTerminal(tTablero t):

Recorre la grilla hasta encontrar un componente de la misma que este vacio. Si alguna está vacía quiere decir que no es terminal y retorna 0, si no, retorna 1, indicando que el estado es terminal.

int verificarGanador(tTablero e):

Controla si algún jugador ganó, analizando cada posible situación en la que se produciría Ta-Te-Ti, si las 3 fichas en las posiciones en las que se produce Ta-Te-Ti pertenecen al mismo jugador retorna 1, indicando que hay un ganador, sino, retorna 0.

void imprimirTablero(tTablero tablero):

Muestra por pantalla el tablero de la partida.