INFORME PROYECTO TA-TE-TI

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

CONTENIDOS

TDA_Lista

Estructura utilizada para representar la lista

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

Operaciones del TDA

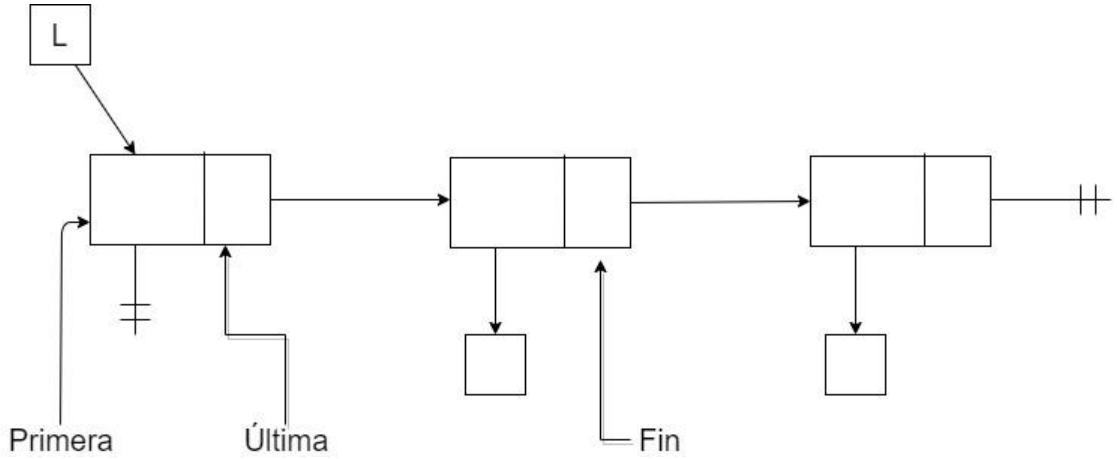
- **Void crear_lista(tLista *l)**
  - **¿Qué hace? : Inicializa una lista vacía**
  - **¿Cómo lo hace? : Recibe por parámetro un puntero a una tLista (un puntero a una estructura Nodo). Guarda lugar en memoria para una estructura Nodo y asigna a su elemento y a su puntero a la siguiente posición como nulos, creando el nodo centinela**
- **Void l_insertar(tLista l, tPosicion p, tElemento e)**
  - **¿Qué hace? : Inserta el elemento 'e' en la lista 'l' antes 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 con 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. Guardo el siguiente a la posición 'p', actualizo el siguiente de 'p' como el siguiente de la posición guardada anteriormente, actualizo el siguiente de la posición guarda a NULL, elimino el elemento de la posición guardada 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 y destruye la lista.**
  - **¿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 nodo de la misma. Por último asigna NULL a la lista.**
- **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' al menos que el siguiente de 'p' sea l_fin(l).**
- **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, al menos que 'p' sea l_primera(l).**
- **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 NULL y tiene un contador, cuando deja de recorrer la lista, devuelve el contador.**

Operaciones Auxiliares

- **Void l_destruirAux (tPosicion aux, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Función auxiliar utilizado en l_destruir(l,fEliminar) que se encarga de recorrer la lista y eliminar cada uno de sus nodos.**
  - **¿Cómo lo hace? : Asigna como NULL al puntero a su siguiente, eliminando el elemento con la función fEliminar y asignándole NULL al puntero a su elemento. Por ultimo libera el espacio de memoria que tenía la posición a eliminar.**

Explicación Posición Indirecta



- **Primera** apunta al nodo centinela de la lista ya que al utilizar posición indirecta, cuando se pida recuperar el elemento de primera, esta accederá al siguiente nodo y luego al elemento en dicha nodo.
- **Última** apunta al antepenúltimo nodo de la lista ya que al utilizar posición indirecta, cuando se pida recuperar el elemento de última, esta accederá al penúltimo nodo de la lista y luego al elemento de dicho nodo.
- **Fin** apunta al anteúltimo nodo de la lista ya que al utilizar posición indirecta, cuando se pida recuperar el elemento de fin, este accederá al último nodo de la lista y luego al elemento de dicho nodo.

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 NULL.**
- **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 NULL.**
- **tNodo a_insertar(tArbol a, tNodo np,tNodo nh,tElemento e)**
  - **¿Qué hace? : Inserta en el árbol '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 su lista de hijos. Si nh es NULL, lo agrega al final de la lista de hijos de np. Si nh no es NULL 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 árbol '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' con la función fNodoEliminar(n,fEliminar).**
- **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 la función eliminar(n,fEliminar) que recorre el árbol en profundidad y elimina cada nodo del mismo utilizando la función fNodoEliminar(n,fEliminar). Por último, asigna a la raíz de 'a' como NULL, libera el espacio de memoria que tenía reservado 'a' y pone a 'a' como NULL.**

- **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 NULL a la raíz de 'a'. Si 'n' no es la raíz de 'a', asigna a 'n' como la raíz de 'sa', recorre la lista de hijos del padre de 'n' y elimina a 'n' de dicha lista.**

Operaciones Auxiliares

- **Void fNodoEliminar(tNodo n, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Elimina el nodo 'n'.**
  - **¿Cómo lo hace? : Eliminar el elemento del nodo 'n', luego destruye su lista de hijos y la asigna como NULL, asigna su padre como NULL y por ultimo libera el espacio de memoria que tenía reservado 'n'.**
- **Void eliminar(tNodo n, void (*fEliminar)(tElemento))**
  - **¿Qué hace? : Función auxiliar utilizada en a_destruir(a,fEliminar) que recorre el arbol 'a' en profundidad y elimina cada uno de sus nodos.**
  - **¿Cómo lo hace? : Recorre la lista de hijos de 'n', se llama recursivamente a si misma con cada hijo de 'n' y así sucesivamente. Por ultimo elimina 'n' con la función fNodoEliminar(n,fEliminar).**

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? : Guarda lugar en memoria para una estructura partida, asigna a modo_partida el valor de 'modo_partida', inicializa el estado de la partida como partida en juego, asigna a turno_de el valor 'comienza' y guarda lugar en memoria para una estructura tablero e inicializa su grilla. Por ultimo guarda el nombre de jugador1 como' j1_nombre' y jugador2 como 'j2_nombre'.***
  - ***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? : Si el movimiento hecho por el jugador actual no es válido retorna movimiento error.***
  Si el movimiento es válido, se actualiza el lugar en la grilla del tablero de 'p' que el jugador ingreso a través de 'mov_x' y 'mov_y', y se actualiza el turno_de de 'p'. Actualiza el estado de 'p' con la función chequearEstadoDePartida(p) y se fija si el estado de la partida, si la partida está en juego retorna movimiento ok sino retorna movimiento error.

- ***void finalizar_partida(tPartida * p)***
  - ***¿Qué hace? : Finaliza la partida***
  - ***¿Cómo lo hace? : Libera el espacio de memoria del tablero de 'p' y le asigna NULL. Luego se libera el espacio de memoria de 'p' y se le asigna NULL.***

- ***Operaciones Auxiliares***
  - ***Void chequearEstadoDePartida(tPartida p)***
    - ***¿Qué Hace? : Actualiza el estado de la partida 'p'.***
    - ***¿Cómo lo hace? : Se fija todas las posibles combinaciones ganadoras, luego si alguna de esas combinaciones está compuesta por tres fichas iguales se fija a que jugador pertenecen dichas fichas y actualiza el estado de la partida como gano jugador1 o gano jugador2.***
      Si no encontró ninguna combinación de fichas iguales, se fija que el tablero de 'p' no esté completo, si está completo actualiza el estado de la partida como partida empate y si no lo está, deja el estado de la partida como partida en juego.

IA

- ***Funcionalidad:*** ***Permite la creación de una nueva IA, encontrar cual es el mejor movimiento próximo a realizar y destruir la IA.***
- ***Responsabilidad:*** ***Realizar las mejores jugadas que pueda para ganar o para que el adversario pierda.***
- ***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? : Guarda lugar en memoria para una estructura búsqueda_adversaria, guarda lugar para una estructura estado, luego inicializa la grilla de estado con la grilla del tablero de la partida 'p'. Asigna, a la utilidad del estado, IA no termino, inicializa los valores que representan a los jugadores max y min, inicializa el árbol de búsqueda adversaria y crea su raíz. Y por último ejecuta la función ejecutar_min_max(b).***
  - ***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? : Guarda el estado en la raíz del árbol de 'b' y guarda la lista de hijos de ese estado. Recorre la lista de hijos y compara sus utilidades, encontrando a su mejor sucesor. Por ultimo utiliza la función diferencia_estados(e,e,x,y) para computar la diferencia entre estados.***

- ***void destruir_busqueda_adversaria(tBusquedaAdversaria * b)***
  - ***¿Qué hace?: Libera el espacio asociado a la estructura correspondiente para la búsqueda adversaria.***
  - ***¿Cómo lo hace?: Destruye el arbol_busqueda de 'b', libera el espacio de memoria que tenía 'b' y le asigna NULL.***

- ***Operaciones Auxiliares***
  - ***Void fEliminar(tElemento t)***
    - ***¿Qué hace?: Libera el espacio de memoria de 't'.***
    - ***¿Cómo lo hace?: Libera el espacio de memoria que tenía ´t´.***
  - ***Int MAX (int valor1,int valor2)***
    - ***¿Qué hace?: Retorna el valor más grande entre 'valor1' y 'valor2'.***
    - ***¿Cómo lo hace?: Compara 'valor1' con 'valor2' y retorna el mayor.***
  - ***Int MIN (int valor1,int valor2)***
    - ***¿Qué hace?: Retorna el valor más chico entre 'valor1' y 'valor2'***
    - ***¿Cómo lo hace?: Compara 'valor1' con 'valor2' y retorna el menor.***
  - ***Static void ejecutar_min_max(tBusquedaAdversaria b)***
    - ***¿Qué hace? : Ordena la ejecución del algoritmo Min-Max para la generación del árbol de b´.***
    - ***¿Cómo lo hace? : Ejecuta el método crear_sucesores_min_max().***
  - ***Static void crear_sucesores_min_max(tArbol a, tNodo n, int es_max, int alpha, int beta, int j_max, int j_min)***
    - ***¿Qué hace? : Implementa la estrategia del algoritmo Min_Max con podas Alpha-Beta a partir del estado almacenado en 'n'.***
    - ***¿Cómo lo hace? : Se compara el valor de utilidad del estado almacenado en 'n' con 'j_max', utilizando la función valor_utilidad(estado, j_max), y se guarda en 'utilidad'. Si la 'utilidad' es distinta de IA_NO_TERMINO, la utilidad del estado de 'n' pasa a ser 'utilidad'.***
      Si no, si 'n' es un nodo max se recorre su lista de sucesores obtenida con la función estados_sucesores(estado, j_max) y se llama

recursivamente a la función con cada estado sucesor de 'n'. Luego se compara 'alpha' con la mejor utilidad obtenida hasta el momento utilizando MAX (v1, v2) y se guarda en 'alpha'. Si 'beta' es menor/igual a 'alpha' se termina la ejecución, sino cuando se termina de recorrer la lista de sucesores la utilidad del estado de 'n' pasa a ser 'alpha'.

Si 'n' es un nodo min, se realizan los mismos pasos que con max pero se compara 'beta' con la mejor utilidad obtenida hasta el momento utilizando MIN (v1, v2) y se guarda en 'beta'.

Por último, se destruye la lista de sucesores de 'n'.

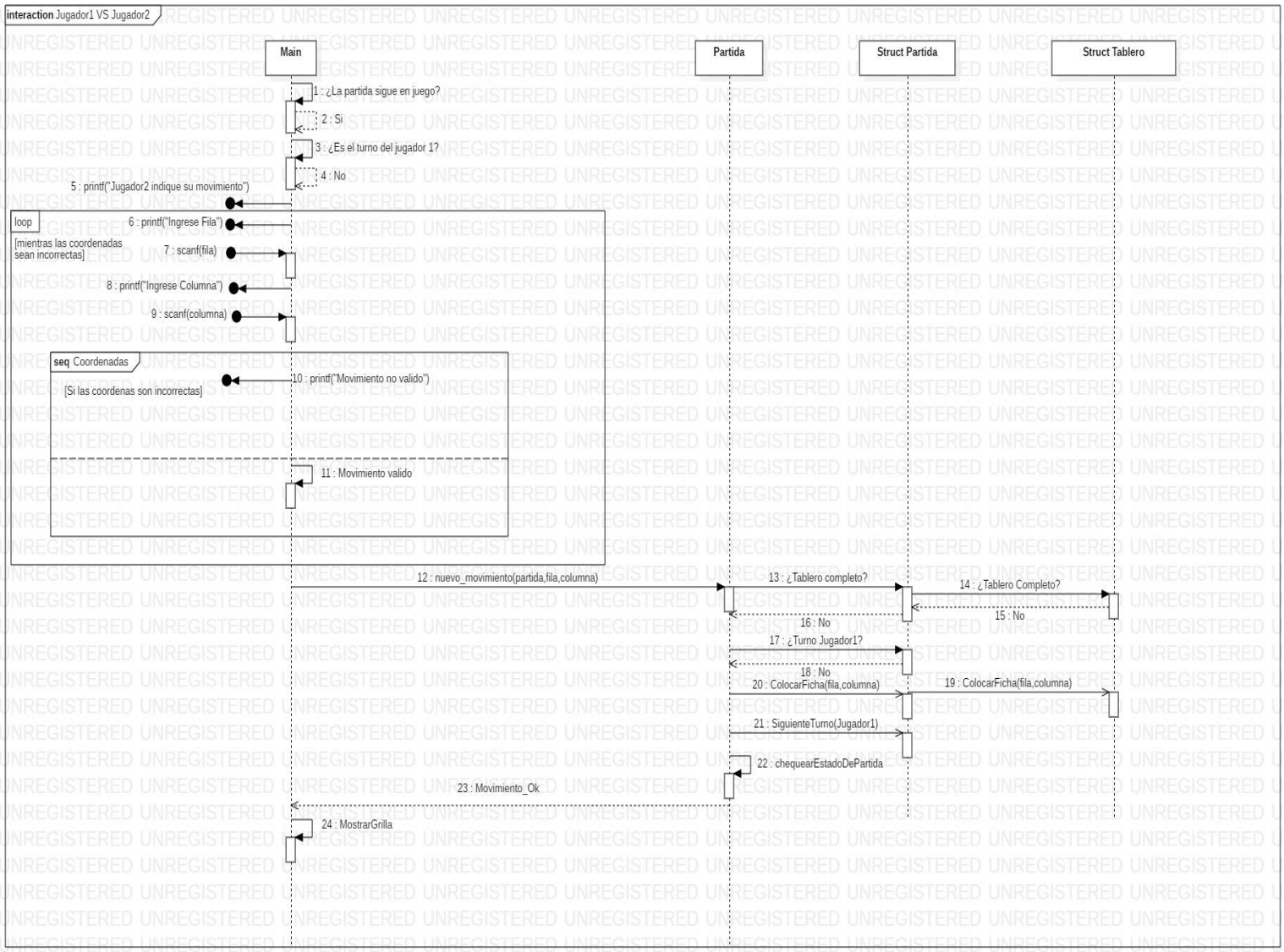
- ***Static int valor_utilidad(tEstado e, int j_max)***
  - ***¿Qué hace? :*** ***Computa la utilidad del estado 'e' respecto del j_max y la retorna.***
  - ***¿Cómo lo hace? :*** ***Se fija todas las posibles combinaciones ganadoras, luego si alguna de esas combinaciones está compuesta por tres fichas iguales se fija a que jugador pertenecen dichas fichas y retorna gano jugador1 o gano jugador2. Si no encontró ninguna combinación de fichas iguales, se fija que la grilla de 'e' no esté completa, si está completa retorna partida empate y si no lo está retorna partida no termino.***
  -
- ***Static tLista estados_sucesores (tEstado e, int ficha_jugador)***
  - ***¿Qué hace? :*** ***Computa una lista con estados sucesores al estado 'e' y la retorna.***
  - ***¿Cómo lo hace? :*** ***Crea una nueva lista vacía, recorre la grilla de 'e' y cuando encuentra un lugar sin una ficha del j1 o j2, crea un estado nuevo clonando 'e' con la función clonar_estado(e), añade la 'ficha_jugador' en la grilla del nuevo estado en el lugar disponible anteriormente encontrado y la inserta en la lista. Por ultimo retorna la lista.***
- ***Static tEstado clonar_estado(tEstado e)***
  - ***¿Qué hace? :*** ***Crea un nuevo estado que resulta de la clonación del estado 'e' y lo retorna.***
  - ***¿Cómo lo hace? :*** ***Guarda lugar en memoria para una estructura estado, recorre la grilla de 'e' y la copia en la grilla del nuevo estado, asigna la utilidad de 'e'***

a la utilidad del nuevo estado y por ultimo lo retorna.

- ***Static void diferencia_estados (tEstado ant, tEstado nuevo, int *x, int *y)***
  - ***¿Qué hace? : Computa la diferencia que existe entre dos estados.***
  - ***¿Cómo lo hace? : Recorre la grilla de 'ant' y cuando encuentra un lugar en 'nuevo' que es distinto al lugar en 'ant', asigna el valor 'i' en 'x' y asigna el valor 'j' en 'y'.***

Relación entre Modulos

- **Situación:** Comenzó el Jugador1, termino de realizar su 3er movimiento (5 jugada de la partida) y es el turno del Jugador2.



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(Aleatorio).
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.
7. Se tendrá la opción de terminar el juego presionando la tecla ‘0’ o de jugar una nueva partida presionando la tecla ‘1’.

Conclusiones

Si en paso 1, 3, 4 y 7 no se pasan los números que se especifican en el modo de ejecución, el programa termina con falla.