Documentación del proyecto de *Organización de Computadoras - Ta-Te-Ti*

Nicolás Dato y Nicolás Medina

03/11/2019

Índice

1. Definiciones y especificación de requerimientos

1.1. Definición general del proyecto de software

Este proyecto se basa en la creación del famoso juego *Ta-Te-Ti*, el cual cuenta con 3 modos de juego, juagador vs jugador, jugador vs computadora y computadora vs computadora.

El modo jugador vs jugador consite simplemente en turnos donde cada jugador realizara una jugada y se notificará si esta se realizó exitosamente, en caso de que no fuera así (como intentar ubicar una ficha en una parte del tablero en el cual ya había una) se pedirá realizar la jugada nuevamente. Las jugadas se ejecutaran decidiendo la posición en la grilla, es decir, decidiendo el lugar mediante cordenadas x e y comenzando a contar desde 0.

Si se desea ejecutar uno de los dos modos donde participa la computadora, cuando sea su turno se ejecutará el algoritmo de **busqueda adversaria MIN-MAX** donde se buscará la mejor jugara, dadas las características del *Ta-Te-Ti* esto significa que la computadora nunca pierde, sólo gana o empata.

Cualquier persona que sepa las reglas del juego puede ser capaz de jugar, lo unico que tendrá que hacer es indicar x e y para colocar la ficha en la posición que se quiera.

1.2. Especificación de requerimientos del proyecto

La implementacion del proyecto se ha realizado en el lenguaje C, el modo de visualización del menú y tablero se ha realizado en la consola. Se ha implementado los *TDA LISTA* y *TDA ARBOL* los cuales se encontrara en una libreria dinamica denominada **libliar** y esta se utilizara para el algoritomo de **busqueda adversaria MIN-MAX** el cual es implementado en un *TDA IA* . También se hizo uso de un *TDA PARTIDA* para el manejo del tablero, turnos, etc. Por ultimo, el programa principal se encargara de de manejar tanto el *TDA PARTIDA* como el *TDA IA* e irá mostrando el estado actual del tablero.

El usuario al iniciar el programa tendrá la opción de iniciar una nueva partida o de salir del sistema, en caso de iniciar una nueva partida tendrá que elejir en qué modo es el que desea jugar y luego decidir los turnos, también se pedirá ingresar el nombre de los jugadores.

El proyecto es un desarrollo original cumpliendo las pautas de implementacion dada por la catedra de la materia *organización de computadoras*, aún así dado el desarrollo de clases y de estructura de datos aprendido en materias

anteriores los *TDA* implementados en el lenguaje C pueden ser reutilizados en proyectos/desarrollos posteriores.

1.3. Especificación de los procedimientos

1.3.1. Procedimientos de desarrollo

Como se dijo anteriormente, la implementación se realizó en el lenguaje C y se ha usado como IDE y compilador de dicho lenguaje el dado por la catedra llamado **Code::Blocks y MinGW**. Se han usado varias librerias de este lenguaje, las primeras dos son *stdlib.h* y *stdio.h* las cuales son las librerias primordiales para el manejo de memoria y de I/O. Una de las otras librerias utilizadas fue *time.h* utilizada en el caso de que el usuario decida que empieza un jugador aleatorio. Otra libreria utilizada fue la de *string.h* usada para copiar cadena de caracteres de una manera mas sencilla.

Para abarcar este proyecto se decidió primero la implementación del *TDA LISTA* para luego poder implementar el *TDA ARBOL* y así estar en condiciones de encarar el algoritmo de **busqueda adversaria MIN-MAX**. Aun así primero decidimos completar el *TDA PARTIDA* antes de desarrollar *TDA IA*.

Una vez con todos los *TDA* completados se pasó a la implementación del programa principal, donde se implementa un menu facilitando la experiencia del usuario.

Ya con todo finalizado se implementĺa libreria **libliar** con los *TDA LISTA* y *TDA ARBOL*.

1.3.2. Procedimientos de compilación y ejecución

Para el uso del mismo, el usuario, el cual es así mismo un desarrollador, contará con el archivo comprimido **.zip** conteniendo todos los archivos .cpb, .h y .c para la compilacón y ejecución del programa.

Dado que se genera una librería (*libliar*), se requiere abrir 2 proyectos con *Code::Blocks*, tp1.cbp y libliar.cbp. En las propiedades del proyecto tp1 se puede configurar que tp1 tiene de dependencia el proyecto libliar[1], para que al compilar tp1 también se compile libliar.

Para compilar el proyecto, primero que hay compilar libliar, que va a generar los archivos *bin\Debug\libliar.a* y *bin\Debug\libliar.dll*.

[1]Lamentablemente no se logró hacer que esta configuración quede guardada para que al abrir el proyecto ya esté la dependencia agregada

Con libliar compilado, se puede compilar el proyecto tp1, el cual va a linkearse con *bin\Debug\libliar.a*[2]

Una vez compilado el proyecto tp1, en *bin\Debug* se genera el archivo *tp1.exe* para ser ejecutado.

2. Arquitectura del sistema

2.1. Descripción jerárquica

La arquitectura del sistema cuenta con un programa principal llamado **main** el cual se encarga de usar el *TDA PARTIDA* para la creación y administración de la misma, también en el **main** usará el *TDA IA* en casos de ser necesario, es decir, si el usuario desea iniciar uno de los dos modos donde juega la computadora.

El *TDA IA* utiliza el algoritmo de **busqueda adversaria MIN-MAX**, también usara al *TDA PARTIDA* y *TDA ARBOL*.

Por ultimo el *TDA ARBOL* usará al *TDA LISTA* para guardar una lista de sus hijos.

2.2. Diagrama de módulos

2.3. Descripción general de los módulos

2.3.1. Descripción general y propósito

- **TDA LISTA**: Implementa nodos enlazados los cuales apuntan al suguiente nodo y almacenan un elemento genérico. La posición es indirecta y tiene un nodo centinela.
- **TDA ARBOL**: Implementa *tnodos* los cuales apuntan a su *tnodo* padre (en caso de no ser la raiz), almacenan un elemento generico y una lista de hijos.
- **TDA PARTIDA**: Implementa una partida con información actual de la misma. La partida guarda tanto los nombres de los jugadores, el turno del jugador al que le toca jugar, el estado altual de la partida (si alguien ganó, etc) y el modo de la partida que se está jugando. El tablero apunta a una grilla de 3x3 la cual guarda el estado del mismo teniendo en cada posición los valores 0, 1 o 2 para el caso de una casilla vacía, una ficha del jugador 1 o una ficha del jugador 2, respectivamente.

[2]Con *Code::Blocks* no se logró utilizar el archivo .dll, la única opción que había era linkear contra el archivo .a, de todas formas el archivo .dll se genera y está disponible.

- **TDA IA**: Implementa un estado y una busqueda adversaria. El estado guarda una grilla de 3x3 con la que fue llamada y la utilidad el cual representa el resultado de la grilla (si se sigue jugando, alguien ganó, etc). Busqueda adversaria apunta a un arbol y a dos enteros, el jugador max y el jugador min. Con esta información el algoritmo buscará la mejor jugada que puede realizar el jugador.

- **main**: Implementa un menú en la consola el cual administra el *TDA PARTIDA* y el *TDA IA* en caso de ser necesario.

2.3.2. Responsabilidad y restricciones

- **TDA LISTA**: La lista espera que cuando se le ingrese un elemento ya tenga un espacio en memoria asignado. La lista, al eliminar o destrir espera una función con la cual eleminará al elemento y liberará el espacio asignado a memoria del mismo. Al insertar un elemento lo asignará como siguiente a la posición recibida.

- **TDA ARBOL**: El arbol espera después de ser creado que se le asigne una raiz como primer elemento, y en caso de que se le intente asignar un nodo que no sea raíz se informará de un error. También cuando se desee insertar un elemento se le tiene que asignar un espacio en memoria al mismo previamente. Al eliminar o destruir, de igual manera que la lista se espera que se le pase una función por parametro la cual se encargara del eleminar y borrar de la memoria al elemento. Al insertar un elemento se esperan dos tnodos, el primero de ellos debe ser el padre al cual se le desea agregar un hijo y el segundo es el hermano derecho al nodo a insertar, en caso de que el segundo nose sea NULL el elemento se asigna como el ultimo hijo del tnodo padre.

- **TDA PARTIDA**: La partida lo que espera como modo y turno son enteros, asociados a contantes ya definidas en el TDA, en el caso del nombre se espera como maximo 49 caracteres. Si dado al caso se intenta hacer un movimiento en una casilla ya ocupada, retornara un error.

- **TDA IA**: La ia al momento de crear la busqueda adversaria lo que espera es que se le haya pasado correctamente la partida en su estado actual, para así poder empezar con el algoritmo. En el caso de resultado esperado lo que espera la ia es que el resultado buscado siempre sea el mejor, es decir, gana max, ya que la ia nunca pierde, sólo gana o empata. Tambien espera dos punteros a enteros los cuales modificara con la posición en la cual esta la mejor jugada posible.

- **main**: El main tiene limitaciones al ejecutarse por consola, las cuales son que los nombres de los jugadores no podran exceder los 49 caracteres, y la segunda es que espara que cada vez que da una opción a elejir espera un número entero.

2.3.3. Dependencias

- **TDA LISTA**: Este TDA no requiere de ningun paquete externo o librería ademas de las librerias esenciales proporcionadas por C.
- **TDA ARBOL**: Este TDA de la librería del *TDA LISTA* ya mencionado.
- **TDA PARTIDA**: Este TDA además de las librerías esenciales de C requiere de las librerías *time.h* y *string.h*.
- **TDA IA**: Este TDA requiere de los *TDA PARTIDA* y *TDA ARBOL* ya mencionados.
- **main**: El main solo requiere del *TDA IA* debido a que este ya tiene contenido todo el resto de los TDA y librerías.

2.3.4. Implementación

- **TDA LISTA**: Este TDA se encuentra implementado en la librería dinámina llamada **libliar**.
- **TDA ARBOL**: Este TDA se encuentra implementado en la librería dinámica llamada **libliar**.
- **TDA PARTIDA**: Este TDA se encontrara definido en un .h denominado *partida.h* e implementado en un .c llamado *partida.c*. Estos archivos se encontraran en el **.zip**.
- **TDA IA**: Este TDA al igual que el anterior se encuentra definido en *ia.h* e implementado en *ia.c*. Estos archivos se encontrarán en el **.zip**.
- **main**: Este se encuentra implementado en *main.c*. Este archivo se encontrará en el **.zip**.
- **libliar**: Librería dinámica con los *TDA LISTA* y *TDA ARBOL*, se encontrará en el **.zip**.

2.4. Dependencias externas

- **stdio.h**: Es la librería que contiene las definiciones de las macros, las constantes, las declaraciones de funciones de la biblioteca estándar del lenguaje de programación C para hacer operaciones de entrada y salida, así como la definición de tipos necesarias para dichas operaciones.
- **stdlib.h**: Es la librerá cabecera de la biblioteca estándar de propósito general del lenguaje de programación C. Contiene los prototipos de funciones de C para gestión de memoria dinámica, control de procesos y otros.
- **time.h**: Es la librería relacionada con formato de hora y fecha, es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C que contiene funciones para manipular y formatear la fecha y hora del sistema.
- **string.h**: Es na libreria de la biblioteca estándar del lenguaje de programación C que contiene la definición de macros, constantes, funciones y tipos y algunas operaciones de manipulación de memoria relacionadas a cadena de caracteres.

3. Descripcion de procesos y servicios ofrecidos por el sistema