Informe Algoritmos y programación II

Trabajo Practico N°2

Integrantes:

- Altieri Lamas, Franco Stefano Daniel
- Arispe, Mijail Braian.
- Balderrama, Carlos
- Cano, Ezequiel Martin
- De Bortoli, Rodrigo Gastón
- Saldaña Nigro, Franco

Docentes:

- Calvo, Patricia
- Schmidt, Gustavo Adolfo

Introducción:

El presente proyecto está dirigido a estudiar y comprender el uso de listas, pilas y colas, así como también el diseño e implementación de tipos de datos abstractos. Para ello se diseñó un juego denominado Tatetí V2.0 en donde se explican algunos ejemplos.

Tatetí es un juego de mesa para varios jugadores en el que se introducen X fichas en un tablero de N por M por Z con el fin de alinear las fichas en una línea recta, ya sea horizontal, vertical o diagonal. Por cada turno cada jugador ingresa una ficha y luego de tener todas las fichas en el tablero se pueden mover en el sentido horizontal o vertical o profundo.

OBJETIVO:

Generar una pieza de software que simule el funcionamiento del juego Tatetí en su versión multijugador, Así como también comprender el uso de las listas, pilas y colas.

MARCO TEORICO:

1) ¿Qué es un svn?

2) ¿Que es una “Ruta absoluta” o una “Ruta relativa”?

1) SVN es un sistema de control de versiones usado para que varios desarrolladores puedan trabajar en un mismo proyecto en forma más o menos ordenada. Tiene una arquitectura cliente servidor con controles de concurrencia para cuando varios desarrolladores están trabajando en el mismo archivo.
2) Una ruta o path es donde se localiza un fichero o archivo dentro de nuestro sistema de ficheros que es como si fuera su dirección. Todos los comandos que hagamos sin especificar una ruta lo hará donde esté situado y existen dos tipos de ruta que debemos diferenciar:
    - **Ruta absoluta:** se indica toda la ruta del archivo incluyendo el directorio raíz. Por ejemplo, C:\carpeta1\carpeta2\archivo1.doc
    - **Ruta relativa:** se indica la ruta a partir de donde este en ese momento situado. No se incluye el directorio raíz. Si estamos en la ruta C:\carpeta1 y queremos acceder al archivo1 que está dentro de la carpeta2, seria carpeta2\archivo1.

Lista:

Una lista es una estructura dinámica de datos que contiene una colección de elementos homogéneos (del mismo tipo) de manera que se establece entre ellos un orden.

Nodos:

Una lista se compone de nodos, que son estructuras de datos que nos permiten registrar datos de interés. Para que estos nodos se conviertan en una lista, debe existir un enlace entre ellos, que en términos más propios se los conoce como punteros. Los punteros, son la parte de los nodos que nos permiten recorrer la lista y acceder también a las direcciones de memoria donde se almacenan los elementos de la lista.

Pila:

Una pila es un contenedor de objetos que se insertan y se eliminan siguiendo el principio 'Ultimo en entrar, primero en salir' (L.I.F.O.= 'Last In, First Out'). La característica más importante de las pilas es su forma de acceso. En ese sentido puede decirse que una pila es una clase especial de lista en la cual todas las inserciones (alta, apilar o push) y borrados (baja, desapilar o pop) tienen lugar en un extremo denominado cabeza o tope. El Tope de la pila corresponde al elemento que entró en último lugar, es decir que saldrá en la próxima baja.

Cola:

Una cola es un contenedor de objetos que se insertan y se eliminan siguiendo el principio 'Primero en entrar, primero en salir' (F.I.F.O.='First In, First Out'). En ese sentido puede decirse que una cola es una lista con restricciones en el alta (encolar o enqueue) y baja (desencolar, dequeue). El Frente (FRONT) de la cola corresponde al elemento que está en primer lugar, es decir que es el que estuvo más tiempo en espera. El Fondo (END) de la cola corresponde al último elemento ingresado a la misma.

MARCO PRACTICO:

Para comprender mejor las listas, lo que se hizo fue desarrollar un tablero tridimensional, en la que se encuentran los distintos métodos que se utilizaron para recrear el juego. Se consideraron las siguientes hipótesis.

- Cada jugador tendrá 3 cartas como **MAXIMO.**
- Si el jugador ingresa una ficha fuera del rango del tablero pierde su turno.

A continuación, se detallarán los métodos utilizados para desarrollar el juego.

TDA TATETI:

Pre: -

Post:crea una partida tatetí.

Tateti();

Pre: -

Post: Libera la memoria asociada al tatetí

~ **Tateti();**

Pre: -

Post: inicializa la lista de jugadores del tatetí

con sus nombres y fichas.

void **crearJugadores**();

Pre: -

Post: Muestra por pantalla la lista de jugadores y sus respectivas fichas.

void **imprimirJugadores**();

Pre: -

Post: inicializa la cola de turnos con los jugadores presentes en el orden que se ingresaron por interfaz.

void **inicializarTurnosJugadores**();

Pre: -

Post: libera la memoria de la lista de los jugadores y su contenido.

void **destruirJugadores**();

Pre: -

Post: de realizar la configuración inicial de los parámetros del tatetí mediante la interacción con el usuario.

void **iniciarJuego**();

Pre: -

Post: comienza el juego

void **jugarJuego**();

Pre: -

Post: devuelve TRUE si existe un ganador o FALSE en caso contrario

bool **hayGanador**();

Pre: -

Post: crea el mazo principal.

void **crearMazoPrincipal**();

Pre: -

Post: libera la memoria del mazo principal utilizada

void **destruirMazoPrincipal**();

Pre: -

Post: avanza el turno actual al siguiente jugador en la cola de turnos.

void **avanzarTurno**();

Pre: -

Post: crea el tablero del juego

void **crearTablero**();

TDA TABLERO:

Pre: Los argumentos deben ser >0

Post: crea un tablero de dimensiones x, y, z

tablero(int cantFilas, int cantColumnas, int cantEnProfundidad);

Pre: -

Post: libera la memoria utilizada en el tablero

~ tablero();

Pre: Recibe los valores de filas columnas y profundidad elegidos

Post: Devuelve la ficha que se encuentra en la casilla elegida

Casillero * **getCasilla**(int filaIntroducida, int columnaIntroducida, int profundidadIntroducida);

Pre:

Post: Coloca una nueva ficha en la casilla seleccionada

void **setCasilla** (int cantFilas, int cantColumnas, int cantEnProfundidad, char ficha);

TDA JUGADOR:

Pre:

Post: crea un jugador con un nombre y una ficha asignada.

Jugador(string nombre,char simboloFicha**);**

Pre:

Post: libera la memoria utilizada para el jugador

~Jugador();

Pre:

Post: asigna un nombre al jugador

void **setNombreJugador(**string nombre);

Pre:

Post: devuelve el nombre del jugador

string **obtenerNombreJugador();**

Pre:

Post: devuelve el valor de la ficha del jugador

char **obtenerSimboloFichaJugador();**

Pre:

Post: asigna una ficha al jugador

void **setFichaJugador(**Ficha * nuevaFicha);

Pre:

Post: libera la memoria de la ficha del jugador

void **destruirFichaJugador();**

Pre:

Post: devuelve el ID del jugador

size_t **obtenerIdJugador();**

Pre:

Post: asigna un ID al jugador

void **setearIdJugador(**size_t id**);**

TDA FICHA:

Pre:

Post: crea una ficha vacia.

Ficha();

Pre:

Post: crea una ficha con el valor ingresado por el jugador

Ficha(char carácter**);**

Pre:

Post: libera la memoria de la ficha.

~Ficha();

Pre:

Post: devuelve el valor de la ficha

char **getSimboloFicha();**

Pre:

Post: devuelve TRUE si la ficha está bloqueada o FALSE en caso contrario

bool **estaBloqueadaFicha();**

Pre:

Post: bloquea una ficha

void **bloquearFicha();**

Pre:

Post: desbloquea una ficha

void **desbloquearFicha();**

TDA CASILLERO:

Pre:

Post: Inicializa un casillero

Casillero();

Pre:

Post: libera la memoria del casillero

~Casillero();

Pre:

Post: devuelve TRUE si el casiilero está vacio o FALSE en caso contrario

bool **estaCasilleroVacio();**

Pre:

Post: devuelte TRUE si el casillero esta anulado o FALSE en caso contrario

bool **estaCasilleroAnulado();**

Pre:

Post: copia el contenido de un casillero a otro casillero

void **copiarCasillero(**Casillero * dest**);**

Pre:

Post: inserta la cantidad de turnos restantes por los cuales el casillero está bajo el efecto de una carta

void **setTurnosRestantesDesbloqueo(**size_t cantidadTurnos**);**

Pre:

Post: devuelve la cantidad de turnos restantes por los cuales el casillero está bloqueado

size_t **getTurnosRestantesDesbloqueo();**

Pre:

Post: decrementa en uno la cantidad de **‘turnosRestantesDesbloqueo’** de un casillero.

void **decrementarTurnosRestantesDesbloqueo();**

Pre:

Post: bloquea la ficha del casillero

void **bloquearFichaDelCasillero();**

Pre:

Post: asigna una ficha al casillero

void **setFicha(**char ficha**);**

Pre:

Post: anula un casillero

void **anularCasillero();**

Pre:

Post: devuelve la ficha que tiene el casillero

char **obtenerFicha();**

TDA MAZO:

Pre:

Post: Crea un mazo con una cantidad de cartas sin efecto

Mazo(int cantidadCartas**);**

Pre:

Post: libera la memoria del mazo

~Mazo();

Pre:

Post: agrega al mazo la carta recibida al final del mismo

void **agregarCarta(**Carta * nuevaCarta**);**

Pre:

Post: devuelve la carta superior del mazo

Carta * **obtenerCartaSuperior();**

Pre:

Post: asigna de manera aleatoria efectos a las cartas de un mazo

void **barajarMazo();**

Pre:

Post: libera la memoria dinámica del mazo

void **destruirMazo();**

TDA CARTA:

Pre:

Post: crea una carta sin ninguna habilidad

Carta();

Pre:

Post: crea una carta con la habilidad "**efectoCarta"**

Carta(habilidadCarta_t efectoCarta**);**

Pre:

Post: libera la memoria de la carta

~Carta();

Pre:

Post: asigna la habilidad **"numero"** a una carta

void **setHabilidad(**habilidadCarta_t numero**);**

Pre:

Post: devuelve la habilidad de la carta

habilidadCarta_t **getHabilidad();**

Pre:

Post: devuelve un efecto aleatorio de los disponibles:

CARTA_SIN_EFECTO, // carta sin efecto

CARTA_PERDER_TURNO, //Hace perder el turno a un jugador

CARTA_BLOQUEAR_FICHA, //Bloquea la ficha de un jugador

CARTA_ANULAR_CASILLERO, //Anula un casillero del tablero

CARTA_VOLVER_JUGADA, //Vuelve atrás una jugada del turno.

CARTA_ELIMINAR_CARTA, //Elimina una carta de un jugador.

CARTA_ROBAR_CARTAS, //Roba todas las cartas de otro jugador.

habilidadCarta_t **generarEfectoAleatorio();**

CONCLUSION:

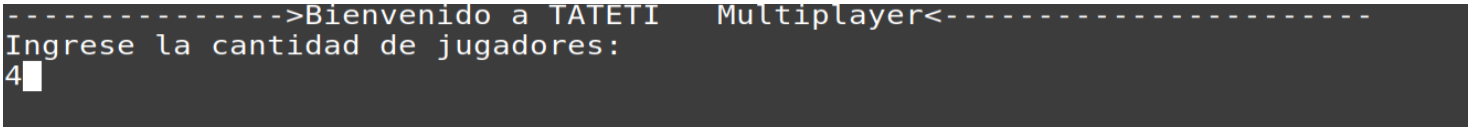
De acuerdo al proyecto realizado podemos concluir:

- Las listas son posiblemente las estructuras de datos más fáciles, rápidas y sencillas de estudiar, aprender y entender.
- La resolución de una función relacionada con listas es fácil y rápida, en especial porque no se requieren demasiadas líneas de código, por ende, la solución es inmediata
- Las listas, así como también otro tipo de estructuras similares, son útiles a la hora de trabajar problemas como PILAS y COLAS, ya que se maneja la misma lógica de agregar, borrar o buscar elementos.

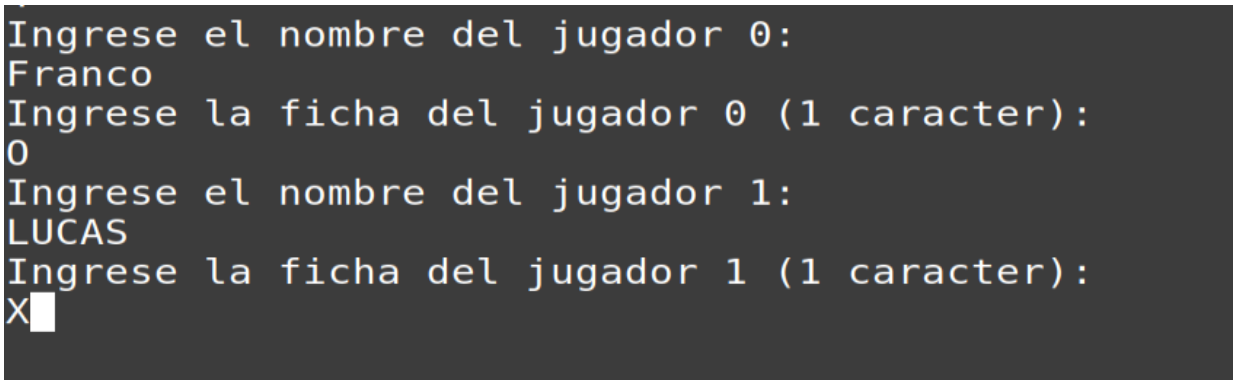
MANUAL DE USUARIO

Esta guía contiene la información que usted necesita para utilizar correctamente el producto.

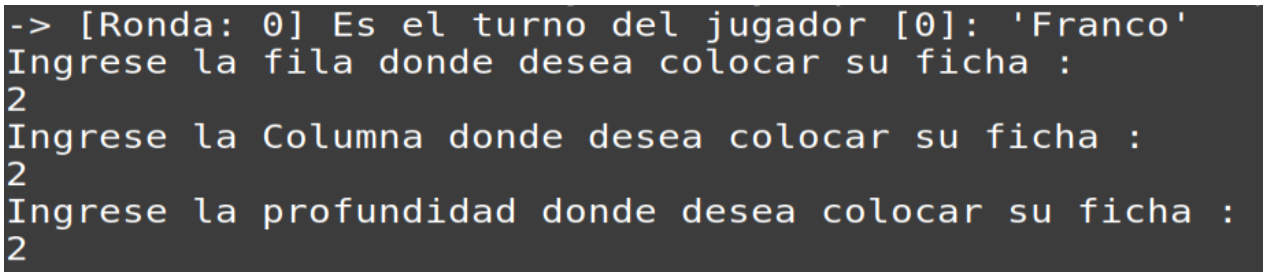
1. Inicie el juego.
2. Una vez iniciado se le pedirá que ingrese la cantidad de jugadores.



3. A continuación, se le pedirá la información de cada jugador y la ficha correspondiente.



4. Luego deberá colocar sus fichas indicando fila columna y profundidad



MANUAL DE PROGRAMADOR

El siguiente manual tiene como objetivo servir de referencia a futuros programadores que trabajen en el proyecto.

Entorno de desarrollo

Para trabajar con el proyecto se necesita tener instalados los siguientes programas y dependencias:

- Eclipse
- C++

Eclipse:

Eclipse es una plataforma de desarrollo, diseñada para ser extendida de forma indefinida a través de plug-ins. Proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones.

Se puede obtener desde https://www.eclipse.org/downloads.

Lenguaje C++:

C++ es un lenguaje de programación orientado a objetos muy potente que evolucionó de la extensión de lenguaje informático “C” y que hoy en día sigue usándose para realizar programación estructurada de alto nivel y rendimiento, como sistemas operativos, videojuegos y aplicaciones en la nube.

Importar proyecto en eclipse:

Para importar el proyecto en eclipse debemos seguir los siguientes pasos:

1. Abrir Eclipse.
2. File->Import->general->Existing projects into Workspace
3. Seleccionamos el directorio del proyecto y finalizamos.

Añadir nuevas características al proyecto:

Tras importar el proyecto en Eclipse, ya estamos en disposición de realizar modificaciones al juego.

TATETI:

Este TDA contendrá la información de todo el juego (tablero, casillero, carta, mazo, jugador).

Tateti: esta función se encargará de crear una partida tatetí lista para cargarla con información.

~Tateti: esta función se encargará de liberar toda la memoria dinámica asociada al tatetí.

crearJugadores: esta función se encargará de crear la lista de jugadores con sus nombres y fichas correspondientes.

imprimirJugadores: esta función se encargará de mostrar por pantallas la lista de jugadores con sus fichas.

InicializarTurnosJugadores: esta función se encargará de inicializar la cola de turnos con los jugadores presentes en el orden que se ingresaron por interfaz.

destruirJugadores: esta función se encargará de liberar la memoria de la lista de los jugadores y su contenido.

iniciarJuego: esta función se encargará de realizar la configuración inicial de los parámetros del tatetí mediante la interacción con el usuario.

jugarJuego: esta función se encargará de comenzar el juego.

hayGanador: esta función se encargará de devolver si existe un ganador del juego.

crearMazoPrincipal: esta función se encargará de crear el mazo principal.

destruirMazoPrincipal: esta función se encargará de liberar la memoria de la lista del mazo principal

avanzarTurno: esta función se encargará de avanzar el turno actual al siguiente jugador en la cola de turnos.

TABLERO:

Este TDA contendrá las dimensiones del tablero junto con los casilleros correspondientes en función de la dimensión.

Tablero: Esta función se encargará de crear un tablero de dimensiones X Y Z (los valores deben ser mayores a cero.

getCasilla: Recibe tres valores X, Y, Z (mayores a cero) y devuelve la ficha que se encuentra en la casilla elegida.

setCasilla: Recibe tres valores X, Y, Z y una ficha y la coloca en la casilla seleccionada.

CASILLERO:

Este TDA almacenará la información de una ficha y los posibles estados que pueda tener un casillero.

casillero: Esta función se encargará de crear un casillero y una ficha sin asignar.

estaCasilleroVacio: Esta función se encargará de informar si un casillero está vacío.

estaCasilleroAnulado: Esta función se encargará de informar si un casillero esta anulado.

copiarCasillero: esta función se encargará de copiar el contenido de un casillero a otro (previamente le casillero debe existir).

setTurnosRestantesDesbloqueo: esta función se encargará de asignar una cantidad de turnos en los cuales un casillero está bajo los efectos de una carta.

getTurnosRestantesDesbloqueo: esta función se encargará de informar la cantidad de turnos restantes por los cuales el casillero está bloqueado.

decrementarTurnosRestantesDesbloqueo: esta función se encargará de decrementar en uno la cantidad de **'turnosRestantesDesbloqueo'** de un casillero.

bloquearFichaDelCasillero: esta función se encargará de bloquear la ficha de un casillero.

setFicha: esta función se encargará de asignar una ficha al casillero.

anularCasillero: esta función se encargará de anular un casillero.

FICHA:

Este TDA contendrá la información del valor de una ficha y el posible estado que puede tener.

Ficha: esta función se encargará de crear una ficha vacía.

Ficha (char carácter): esta función se encargará de crear una ficha con el valor ingresado por el jugador.

getSimboloFicha: esta función se encargará de devolver el valor de la ficha.

estaBloqueadaFicha: esta función se encargará de informar si una ficha está bloqueada.

bloquearFicha: esta función se encargará de bloquear una ficha.

desbloquear Ficha: esta función se encargará de desbloquear una ficha.

CARTA:

Este TDA contendrá la información del valor de la habilidad de una carta.

Carta: esta función se encargará de crear una carta sin habilidad.

Carta (habilidadCarta_t efectoCarta): esta función se encargará de crear una carta con una habilidad obtenida aleatoriamente.

setHabilidad: esta función se encargará de asignar una habilidad a la carta.

getHabilidad: esta función se encargará de devolver la habilidad de la carta.

generarEfectoAleatorio: esta función se encargará de generar y devolver una habilidad aleatoria.

~carta: esta función se encargará de destruir una carta.

MAZO:

Este TDA contendrá una lista de cartas.

Mazo: esta función se encargará de crear un mazo con una cantidad de cartas sin efecto.

agregarCarta: esta función se encargará de agregar al mazo una carta al final del mismo.

obtenerCartaSuperior: esta función se encargará de devolver la carta superior del mazo.

barajarMazo: esta función se encargará de asignar de manera aleatoria efectos a las cartas un mazo.

~destruirMazo: esta función se encargará de liberar la memoria de un mazo.

JUGADOR:

Este TDA contendrá información del nombre del jugador, ficha, cartas y un id que lo identifique.

Jugador: esta función se encargará de crear un jugador con un nombre y una ficha asignada.

setNombreJugador: esta función se encargará de asignar un nombre al jugador.

obtenerNombreJugador: esta función se encargará de devolver el nombre del jugador.

obtenerSimboloFichaJugador: esta función se encargará de devolver el valor de la ficha del jugador.

setFichaJugador: esta función se encargará de asignar una ficha al jugador.

destruirFichaJugador: esta función se encargará de liberar la memoria de la ficha del jugador.

obtenerIdJugador: esta función se encargará de devolver el id del jugador.

setearIdJugador: esta función se encargará de asignar el id a un jugador.