Universidad de Buenos Aires
Facultad De Ingenieria
Año 2021-2do Cuatrimestre

TP2 Algoritmos y programación II (75.41 / 95.12 / 95.15)

Curso: 1
Fecha de entrega: 9-12-21

INTEGRANTES:

- **Ursino, Ian Mika**
  Padrón: 104509
  Mail: iursino@fi.uba.ar
- **Aguirre, Walter Mario Antonio**
  Padrón: 106328
  Mail: waguirre@fi.uba.ar
- **Pescie, Juan Pablo**
  Padrón: 105904
  Mail: jpescie@fi.uba.ar

1. Objetivo

Generar una pieza de software que simule el funcionamiento del juego Tateti en su versión multi Jugador.

2. Cuestionario

- ¿Qué es un svn?

Un subversion (svn) es una herramienta para sistema de versionado de archivos y repositorios.
Un repositorio es un almacenador de archivos y carpetas donde.
El sistema funciona manejando los cambios en los archivos de un repositorio principal, donde cada vez que se suben nuevas versiones del código, el sistema solo analiza los archivos que tienen cambios con respecto al repositorio original, por esto, el sistema es más eficiente acortando tiempos en vez de sobreescribir todos los archivos que son subidos innecesariamente al no tener cambios.

- ¿Que es una “Ruta absoluta” o una “Ruta relativa”?

Una ruta absoluta es la forma de decir donde está ubicado un archivo, aplicación o carpeta, de manera que contiene todos los directorios por los que pasó desde el directorio raíz del sistema, o dominio web.
Una ruta relativa es la ubicación del archivo relativa desde un directorio raíz parcial, como por ejemplo el directorio raíz de la aplicacion o pagina web
Por ejemplo:
Ruta absoluta : “http://www.paginaweb.com/productos/nuevos/remeras”
Ruta relativa de remeras desde la raíz de la página: “/productos/nuevos/remeras”.
Ruta absoluta : “C://Users/usuario/documents/proyecto/carta.cpp”
Ruta relativa de carta.cpp desde proyecto: “./carta.cpp”

3. Manual del programador

El programa contiene diversos TDA que corresponden a cada elemento que compone al juego. Por ejemplo a los jugadores, el tablero, las cartas,etc. Cada uno está separado en un archivo header que contiene los métodos y atributos de cada TDA, junto con las definiciones y pre-post condiciones de los métodos. Por otro lado se encuentran los archivos source donde se encuentra la implementación de los métodos. También está el archivo main.cpp que se utiliza para correr el programa.
Además de los TDA que se describirán más adelante, para la implementación del juego se utilizan templates de lista, que dentro utiliza un template de nodo para su funcionamiento, y un template de cola. Además, para generar el tablero del juego en un archivo BMP se utiliza la biblioteca EasyBMP, con el cual se pueden ir viendo los sucesivos estados del tablero durante el juego.

Explicaremos parte de los TDAs utilizados y sus funciones más relevantes

TDA CASILLERO

este tda va a simular el comportamiento de una casilla del tablero que luego vamos a explicar también, este tda tiene como atributos la ficha, que en principio es un carácter vacío , donde va a ir el carácter que luego ponga el usuario, tambien tiene atributo ficha anterior que es la ficha que estuvo antes que la actual,
Casilleros Vecinos que es una matriz de 3 x 3 x 3, que simula lo que son los casilleros que tienen al lado en el tablero, estos representan las fichas que rodean al casillero, si tiene como vecinos en una posición no válida, tiene como valor NULL.
Y también un atributo estado que define si se puede usar o mover la ficha dentro del casillero
El TDA tiene varios getters y setters para los estados, los vecinos y las fichas
entre las funciones a describir podemos mencionar la funcion getAdyancente y tieneAdyacente

```
Casillero* Casillero::getAdyacente(int profundidad, int fila, int columna){
    return this->casillerosVecinos[profundidad][fila][columna];
}


bool Casillero::tieneAdyacente(int profundidad, int fila, int columna){
    if( (profundidad >= 0 && profundidad<=2) && (fila>= 0 && fila<=2) && (columna >= 0 && columna<=2)){
        return this->casillerosVecinos[profundidad][fila][columna] != NULL;
    }
    return false;
}
```

son las funciones que determinan si realmente el casillero tiene un vecino en la matriz de vecinos en la posición que se pasa por parámetro, estas funciones nos van a servir para determinar cuando se forma línea de tateti

TDA TABLERO

El TDA Tablero es un tda contenedor que dentro suyo tiene instancias de casillero en cada una de las coordenadas, se implementó mediante una lista de lista de listas de punteros a casillero de forma dinámica manejada con punteros, la lista de lista de listas simula la matriz n x m x k. dentro de cada uno de los nodo, en el atributo valor, tiene un puntero a casillero
como atributos tenemos la profundidad, la altura y la cantidad de columnas, para conocer las dimensiones del tablero
luego guardamos las coordenadas de la última posición para luego poder usarla como referencia al último movimiento
dentro de las funciones más importantes tenemos :
getCasilla, que le hace un get para obtener el puntero a casillero ubicado en la coordenada z, y, x que le pasamos por parametro

```
Casillero* Tablero::getCasilla(int profundidad, int fila, int columna){
    return this->casilleros->get(profundidad)->get(columna)->get(fila);
}
```

luego la función más importante, hayTateti, que nos sirve para saber si se formo en línea continua de la ficha que se le pasa por parámetro

```
bool Tablero::hayTateti(int cantidadDeFichasParaGanar){
    Casillero * casilleroAChequear = this->getCasilla(this->ultimaProfundidad, this->ultimaFila, this->ultimaColumna);
    int longitudesAdyacentes[3][3][3];
    int sumaLongitudes[13];

    for(int i=0; i<3; i++){
        for(int j=0; j<3; j++){
            for(int k=0; k<3; k++){
                if(i == 1 && j == 1 && k == 1){
                    longitudesAdyacentes[i][j][k] = 1;
                }
                else{
                    longitudesAdyacentes[i][j][k] = casilleroAChequear->getLongitud(i,j,k);
                }
            }
        }
    }
    sumaLongitudes[0] = longitudesAdyacentes[0][0][0] + longitudesAdyacentes[2][2][2];
    sumaLongitudes[1] = longitudesAdyacentes[0][0][1] + longitudesAdyacentes[2][2][1];
    sumaLongitudes[2] = longitudesAdyacentes[0][0][2] + longitudesAdyacentes[2][2][0];
    sumaLongitudes[3] = longitudesAdyacentes[0][1][0] + longitudesAdyacentes[2][1][2];
    sumaLongitudes[4] = longitudesAdyacentes[0][1][1] + longitudesAdyacentes[2][0][1];
    sumaLongitudes[5] = longitudesAdyacentes[0][1][2] + longitudesAdyacentes[2][1][0];
    sumaLongitudes[6] = longitudesAdyacentes[0][2][0] + longitudesAdyacentes[2][0][2];
    sumaLongitudes[7] = longitudesAdyacentes[0][2][1] + longitudesAdyacentes[2][0][1];
    sumaLongitudes[8] = longitudesAdyacentes[0][2][2] + longitudesAdyacentes[2][0][0];
    sumaLongitudes[9] = longitudesAdyacentes[1][0][0] + longitudesAdyacentes[1][2][2];
    sumaLongitudes[10] = longitudesAdyacentes[1][0][1] + longitudesAdyacentes[1][2][1];
    sumaLongitudes[11] = longitudesAdyacentes[1][0][2] + longitudesAdyacentes[1][2][0];
    sumaLongitudes[12] = longitudesAdyacentes[1][1][0] + longitudesAdyacentes[1][1][2];

    for(int i=0; i<13; i++){
        if(sumaLongitudes[i] + 1 == cantidadDeFichasParaGanar){
            return true;
        }
    }
    return false;
}
```

la función comienza viendo donde fue la casilla donde se hizo el último movimiento, para empezar a ver desde ahí
luego en longitudes adyacentes guarda la guarda la cantidad de fichas seguidas que tienen los casilleros i j k,
luego arma un array de enteros donde esos enteros vana a ser la cantidad de fichas consecutivas con el mismo carácter, luego con el ciclo for se fija si alguno de los enteros de sumaLongitudes junta la cantidad de casilleros necesarios para ganar y si los cumple devuelve true, y si no devuelve false
luego la función mostrarTablero por capas que lo que hace es mostrar por consola capa por capa el tablero
y la función generarBitMap escribe en un archivo bitmap con el tablero también por capas
La función reiniciar tablero setea la ficha de todos sus casilleros en carácter espacio(“ “) que simulará el vacío

TDA JUGADOR

este TDA se ocupa de instanciar a los participantes del juego, mediante un nombre y un identificador
este también guarda la cantidad de cartas que va a tener en su poder
y también tiene como atributo la cantidad de veces que ganó la partida

TDA CARTA

Este TDA se ocupa de darle funcionalidad a las cartas del juego que se van a aplicarse en el tablero

la función utilizarCarta lo que hace es hacer un switch y dependiendo del número que se le pase va a accionar una de las 6 cartas que hay en el juego,
las funcionalidades de las mismas están detalladas en el manual del usuario
todas reciben un tablero y la cola de turnos para poder manejar los turnos y una vez que se acciona por ejemplo la carta de hacer perder un turno a un jugador poder manejar la cola de turnos y tirar su turno para atrás de la cola

TDA TURNOS
este TDA asocia un jugador con una lista de cartas que podrá usar durante el juego.

TDA TATETI
Es el TDA en el que se va a situar todo el juego, el constructor va a crea un puntero a tablero nulo, que después se va a construir cuando se le pida las dimensiones al usuario
funcion iniciarJuego

```
void TATETI::iniciarJuego(){
    this->mostrarBienvenida();

    cout<<"\n";

    this->pedirDimensionesDelTablero();

    cout<<"\n";

    this->inicializarJugadores();

    cout<<"\n";

    this->crearCartas();

}
```

mediante una serie de inputs le da la bienvenida al juego
y se construye el tablero en base a las medidas del usuario y crea todas las cartas

funcion jugarPartida

```
void TATETI::jugarPartida(){

    this->pedirNombreDelArchivoBMP();

    string nombreDelArchivoFinal = this->nombreDelArchivoBMP;

    nombreDelArchivoFinal += "final.bmp";

    if(this->insertarFichas()){
        cout<<"Estado final del tablero: "<<endl<<endl;
        this->imprimirTableros();
        cout<<endl;
        this->tablero->generarBitMap(nombreDelArchivoFinal);
        this->tablero->reiniciar();
        return;
    }

    this->moverFichas();

    cout<<"Estado final del tablero: "<<endl<<endl;
    this->imprimirTableros();
    cout<<endl;

    this->tablero->generarBitMap(nombreDelArchivoFinal);
    this->tablero->reiniciar();
}
```

esta función es la principal del desarrollo del juego, ya que en esta se le pide a los usuarios que introduzcan las fichas en el tablero y que una vez que se pusieron todas las fichas en el tablero, que las empiezen a mover hasta que termine el juego

funcion insertarFichas

```
bool TATETI::insertarFichas(){
    int contador = 0;
    string cadenaColumna, cadenaFila, cadenaProfundidad;
    bool finalizoLaPartida = false;

    while(contador < this->cantidadDeFichasPorJugador * this->cantidadDeJugadores
            && !finalizoLaPartida){

        Jugador *jugadorActual = this->turnos->front()->getJugador();


        string nombreDelArchivoBMP = this->nombreDelArchivoBMP;
        char numero[3];

        intToString(contador, numero);
        nombreDelArchivoBMP += numero;
        nombreDelArchivoBMP += ".bmp";

        cout<<"\nTurno de: "<<jugadorActual->obtenerNombre()<<endl;
        cout<<"Ingresar coordenadas: "<<endl;

        cout<<"Columna: ";
        cin>>cadenaColumna;

        cout<<"Fila: ";
        cin>>cadenaFila;

        cout<<"Profundidad: ";
        cin>>cadenaProfundidad;

        if(esUnNumero1(cadenaColumna) ||
                esUnNumero1(cadenaFila) ||
                esUnNumero1(cadenaProfundidad)){

            int columna = atoi(cadenaColumna.c_str());
            int fila = atoi(cadenaFila.c_str());
            int profundidad = atoi(cadenaProfundidad.c_str());
```

```
if(validarCoordenadas(columna,fila,profundidad)){
    if(this->tablero->casilleroEstaVacio(profundidad, fila, columna)
            && this->tablero->getCasilla(profundidad, fila, columna)->estaDisponible()){

        this->tablero->getCasilla(profundidad, fila, columna)->setFicha(jugadorActual->obtenerFicha());
        this->tablero->setUltimaPosicion(profundidad, fila, columna);
        if(this->tablero->hayTateti(this->getCantidadDeFichasPorJugador())){
            cout<<jugadorActual->obtenerNombre()<<" gano la partida"<<endl;
            jugadorActual->ganoLaPartida();
            return true;
        }


        this->repartirCartas();
        cout<<endl;
        this->imprimirTableros();
        cout<<endl;
        if(!finalizoLaPartida){
            this->utilizarCarta();
            // se setea la ultima posicion en coordenadadas en utilizarCarta
            if(this->tablero->hayTateti(this->getCantidadDeFichasPorJugador())){
                cout<<jugadorActual->obtenerNombre()<<" gano la partida"<<endl;
                jugadorActual->ganoLaPartida();
                finalizoLaPartida = true;
            }
        }


        this->tablero->generarBitMap(nombreDelArchivoBMP);
        Turnos *turnoActual = this->turnos->front();
        turnos->desacolar();
        turnos->acolar(turnoActual);
        contador++;
    }

    else{
        cout<<"El casillero no esta vacio"<<endl;
    }
}
else{
    cout<<"Coordenadas invalidas"<<endl;
```

hace la lógica de pedirle al usuario las coordenadas donde colocar su ficha
una vez que son válidas, se fija que no esté ocupado el casillero
luego hace la comparación si hay tateti y si no hay, pasa al fondo de la cola de turnos al jugador actual

funcion moverFichas

tiene un comportamiento similar a insertarFichas pero con la lógica de mover para alguna dirección la ficha que se selecciona y luego tambien compara y se fija si se gano el juego

funcion imprimirTableros
esta funcion llama al metodo mostrarTableroPorCapas del tablero y lo imprime

funcion repartirCartas

```
void TATETI::repartirCartas(){
    Lista<Carta*> *cartasJugadorActual = this->turnos->front()->getCartasDelJugador();

    if(cartasJugadorActual->contarElementos() < this->getCantidadMaximaCartasPorJugador()){
        Carta *carta = this->cartas->front();
        cartasJugadorActual->add(carta);
        this->cartas->desacolar();
        this->cartas->acolar(carta);
    }
}
```

esta función se ocupa de repartir las cartas al jugador al que le corresponde el turno
si el jugador ya tiene la cantidad de cartas máxima permitida, no se le reparte más
IMPORTANTE: la cantidad máxima de cartas está determinada por el programador, en la constante CARTAS_MAXIMAS_POR_JUGADOR

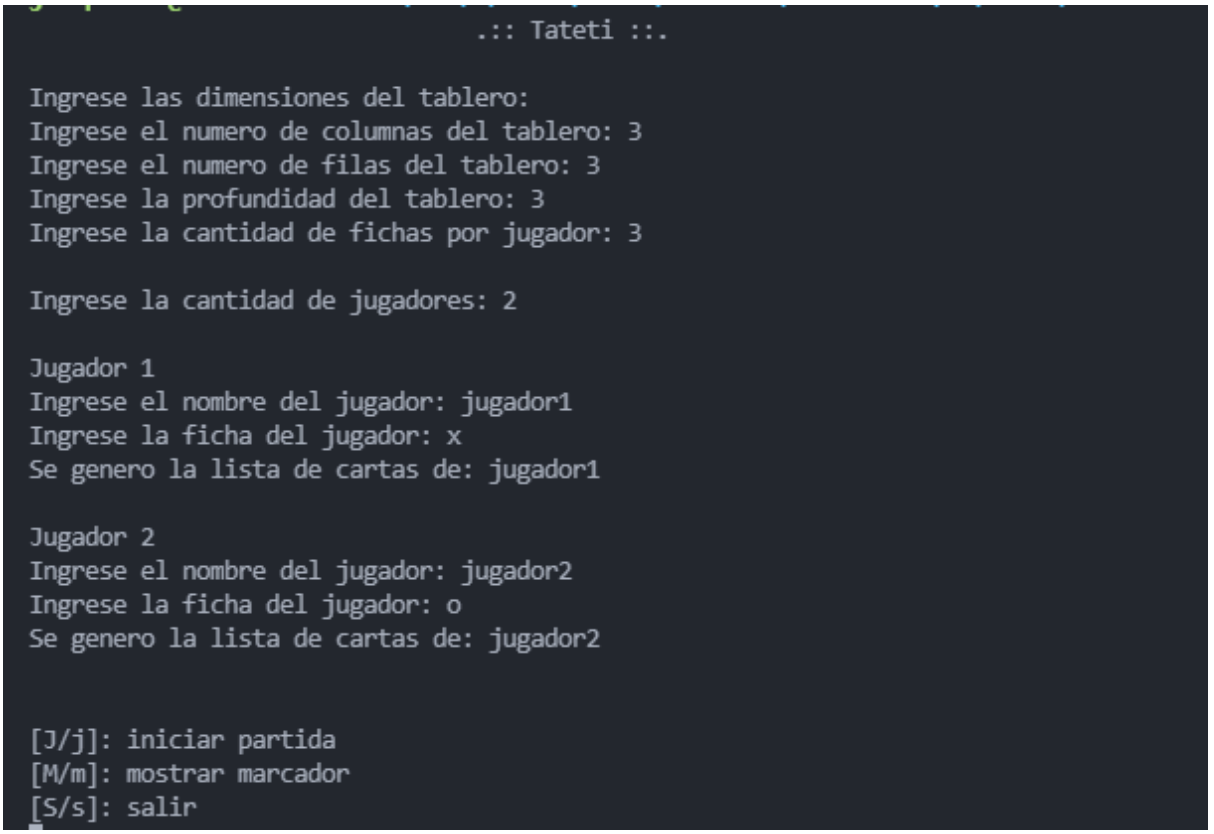
```
const int CARTAS_MAXIMAS_POR_JUGADOR = 3;
```

si se quiere cambiar la cantidad de cartas se modifica la constante al principio del archivo TATETI.cpp

4. Manual del usuario

El objetivo del juego consiste en colocar fichas de manera alineada en un tablero tridimensional.

Al comenzar, se le pide al usuario que ingrese por la consola una serie de parámetros que están relacionados con la jugabilidad. Entre ellos, se deberá ingresar las dimensiones del tablero, la cantidad de jugadores, la cantidad de fichas necesarias para ganar, y datos relacionados con los jugadores. Los sucesivos estados del tablero, que irá cambiando a medida que se juega, se guardan luego de cada turno en un archivo BMP, el cual se le pedirá al usuario que ingrese el nombre que tendrá el archivo o alternativamente su ruta.



Una vez ingresado todo,apretando la tecla J/j empezará el juego.

Además de las fichas que cada jugador debe insertar, se dispone de 6 cartas distintas que se pueden usar después de jugar una ficha:

-**Anular casillero**: inserta en la posición que indica el jugador, en caso de estar vacía, un carácter "/" que indica que el casillero está bloqueado y no se pueden insertar fichas, y si ya hay una ficha presente, el casillero no se puede bloquear.

-**Hacer perder turno al siguiente jugador**: el próximo jugador no podrá jugar su turno y será salteado y recién podrá jugar la próxima ronda

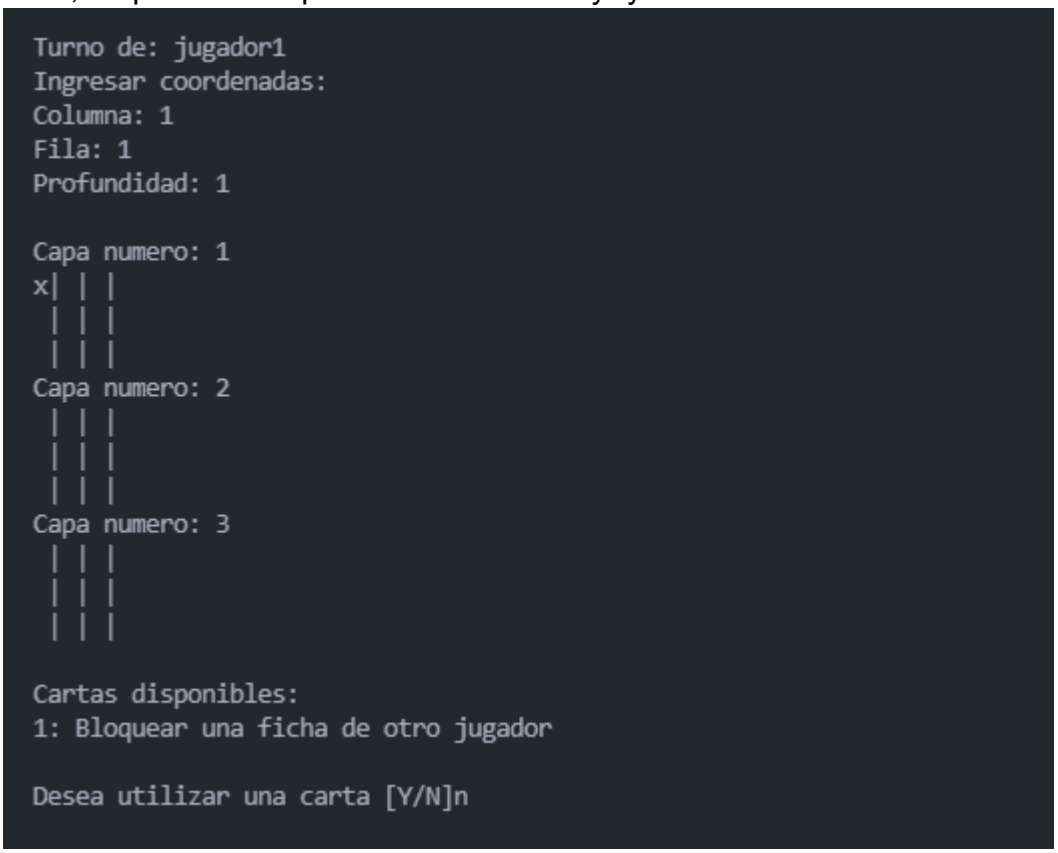
-**Bloquear Ficha**: hace que una ficha no se pueda mover en los próximos movimientos

-**Desbloquear Ficha**: deshace la acción de bloquear casillero, si este estaba bloqueado, y permite el movimiento del mismo en sus próximas jugadas

-**Habilitar casillero**: Deshace la acción de la carta “Anular casillero”. Se aplica sobre un casillero que esté bloqueado y lo desbloquea, dejándolo libre para insertar fichas.

-**Volver atrás un turno**: Esta carta no debe utilizarse al momento de insertar fichas, ya que si se hace esto, retirara la última ficha ingresada por el jugador, quedando en desventaja ya que tendrá una ficha menos que el resto de los jugadores en el tablero. Si se la utiliza al momento de mover fichas, se deshace el último movimiento de ficha realizado por el jugador.

Una vez que esté jugando sus primeros turnos para ubicar las fichas, el jugador recibirá por consola una serie de inputs para insertar sus fichas en el tablero, mostrándole a cada jugado el estado del tablero y preguntándole si desea usar una carta, respondiendo que si con la letra “Y/y” y no con la letra “n”



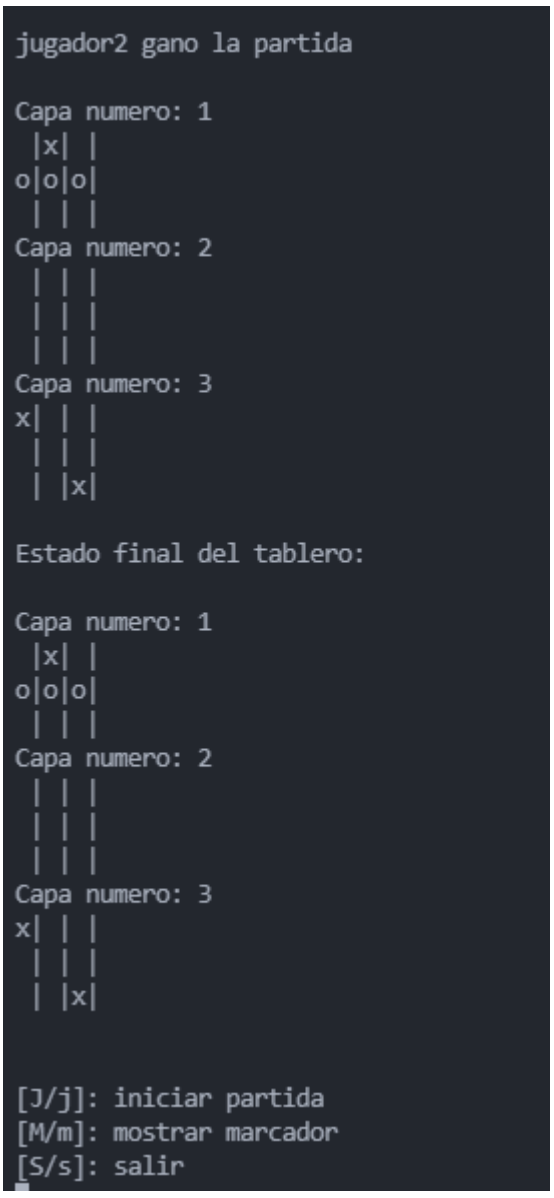
Si el jugador ya ingresó todas las fichas a disposición, se deberá elegir una del tablero y seleccionar una dirección (arriba, izquierda, derecha, abajo, para el frente o para atrás) para moverla o pasar sin mover la ficha, y así lograr el objetivo del juego. Esto se podrá hacer siempre y cuando no haya una ficha de otro jugador en el casillero del tablero donde se quiera mover, o éste esté habilitado.

Al momento de jugar un turno, se le pide al usuario que ingrese el número de columna, fila y profundidad del tablero donde se quiere insertar una ficha. Si el lugar está ocupado/deshabilitado, se deberá ingresar de nuevo. Luego de insertar, se le pregunta al jugador si quiere utilizar una carta. En caso afirmativo se muestran las cartas disponibles junto con una descripción de lo que hace cada una y un número para identificarla. Para elegir una carta basta con ingresar en la consola el número de la carta que se quiere utilizar.

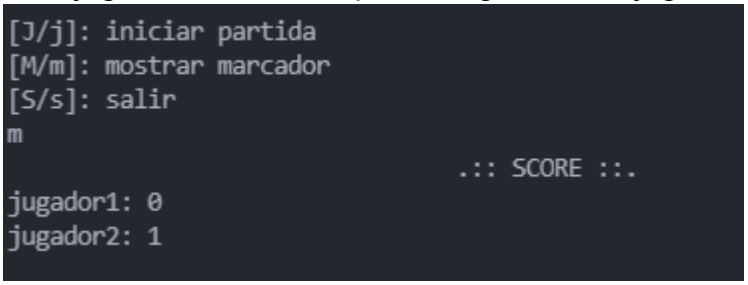
Según la funcionalidad de la carta elegida, se le podrá pedir al usuario que ingrese ciertos datos relacionados con el tablero. Con respecto a estas, el jugador comienza con una carta al empezar el juego y se le irá agregando una luego de jugar cada turno hasta que tenga 3 cartas (valor máximo de cartas por defecto)

Como se mencionó anteriormente, si el jugador ya ingresó todas sus fichas el juego continuará pidiéndole al usuario que las mueva hasta que estén alineadas y de esta forma gane el juego. En esta parte también se pueden utilizar las cartas que se disponen.

El juego finaliza cuando algún jugador logre alinear todas sus fichas en el tablero, ya sea durante el proceso de ingreso de fichas, de moverlas o utilizando una carta que permita hacerlo. Una vez terminado se le muestra por consola un mensaje indicando que el jugador que completó el tateti ganó la partida



si se selecciona la opción mostrar marcado se muestra como va el marcador entre esos jugadores, cuantas partidas gano cada jugador



si se quiere jugar de nuevo, se pulsa la tecla ”j” y se reinicia el tablero, y se le pide al usuario un nuevo nombre para el archivo donde generar el tablero y comienzan una nueva partida entre los jugadores ya declarados anteriormente
si no quiere seguir el juego, con pulsar la letra “s” ya basta para salir del juego

5. Informe

El trabajo consiste en el desarrollo de un programa para jugar al juego “Ta te ti”. A diferencia de como se lo juega normalmente, con un tablero de dos dimensiones que representan la altura y el ancho del mismo, en esta implementación el tablero es una estructura de tres dimensiones, siendo la profundidad aquella que es nueva en el sistema del juego.
Además de las fichas que se deben colocar en el tablero, los jugadores tienen cartas a disposición con diferentes funcionalidades, que permiten que la jugabilidad sea más dinámica y entretenida ya que aumentan el nivel estratègico del juego.

Comenzando con el desarrollo del trabajo, al ser un juego que contiene varios elementos como los jugadores, el tablero, etc, resultó una tarea larga tener que pensar y luego implementar cada uno de estos. Por esto es que nuestra metodología de trabajo consistió en plantear un esquema general de todo lo necesario para implementar el juego, es decir, analizar cada una de las estructuras que lo componen y cómo se relacionan entre sí al momento de escribir el código. De esta manera se nos hizo mucho más fácil el desarrollo porque teníamos claro por dónde seguir en cada momento. Obviamente surgieron complicaciones al momento de pensar como resolver algunos problemas, por ejemplo la condición para que el juego termina cuando un jugador tiene las fichas necesarias alineadas en el tablero, pero organizándonos entre los miembros del grupo y aportando diferentes ideas pudimos resolver las cuestiones más complejas.

En la implementación de las estructuras necesarias para permitir el funcionamiento del juego, predomina el uso de memoria dinámica junto con temas vistos en las clases. El concepto de TDA es fundamental ya que se utiliza para representar prácticamente todos los elementos que componen al juego. Los jugadores, el tablero, las cartas e incluso el juego en sí son tipos de datos abstractos que dentro de ellos incluyen otros TDA, como pilas y/o colas. De esta manera el código resulta mucho más prolijo y organizado ya que las funciones y métodos están contenidas dentro de cada uno de estos. Respecto al manejo de memoria dinámica, su uso resulta conveniente ya que las estructuras necesarias para hacer funcionar al juego se van creando en el mismo momento que se las necesite (y al finalizar se las destruye), y con esto nos ahorramos pedir de entrada una cantidad de memoria fija, que eventualmente podría desperdiciarse si no se utiliza completamente.