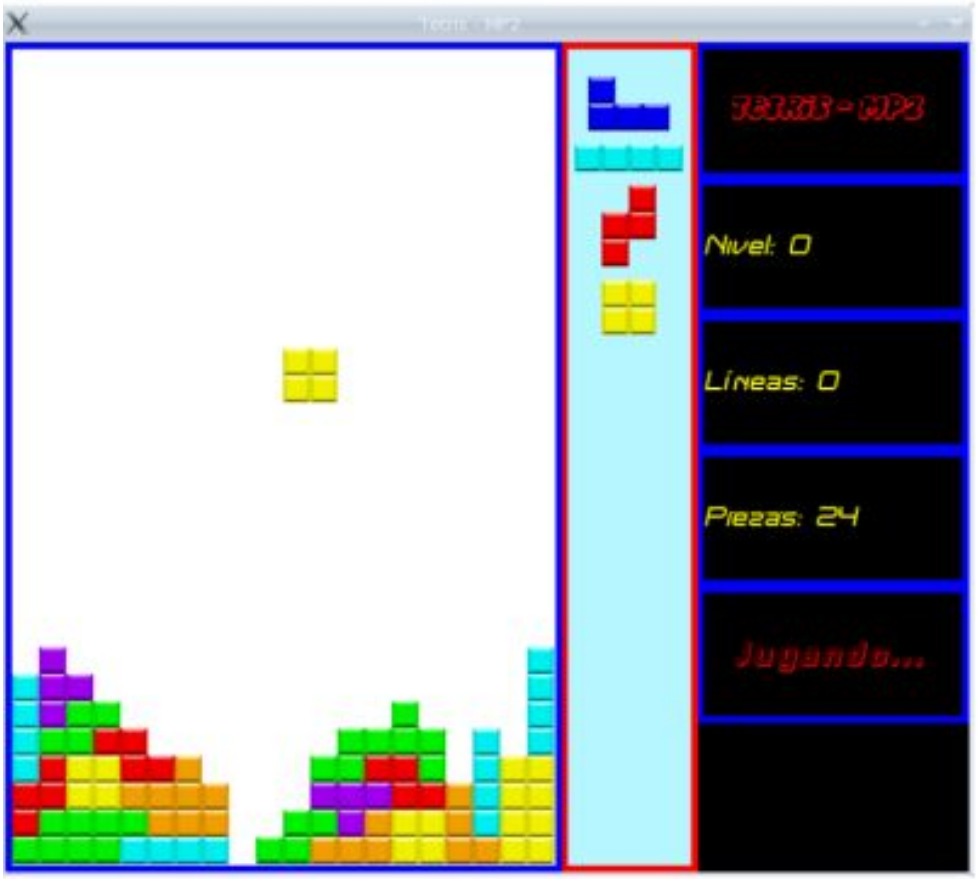
Reto 2: Estructura de Datos

David Martinez Diaz - Jose Luis Rico Ramos



- Introducción:

Para este reto, de las clases TDA, nosotros hemos pretendido implementar ciertas clases que engloban las principales partes del juego Tetris como son;

- ***RaggMatrix***
- ***Pieza***
- ***Tablero***
- ***ColaPiezas***
- ***Estadísticas***
- ***Juego***

En primer lugar, en un aspecto general, podemos comentar la clase Juego, con el cual, a través de diversos métodos esta controla todo el proyecto en sí, ya sea organizando las estadísticas o "limpiandolas", o estando pendiente de la situación del tablero y la lista de la cola de piezas, es decir, será el cerebro del trabajo.

Con las que vamos a trabajar de forma dinámica y pasaremos a explicar una a una en el encabezado de las mismas.

- Clases:

1. Clase RaggMatrix

.- Presentación:

Esta clase contiene en la parte privada, una Raggmatrix de tipo dinámico compuesta por un puntero a puntero de tipo int (el tipo int es el que determinará el color de la pieza en su defecto, ej: 1[Azul], 2[Rojo], etc...)
Dos enteros como son el número de filas y un vector que indique el número de*columnas por fila.

En la parte pública tenemos los constructores y destructores propios de la clase además de los métodos Get y los métodos at(tanto const como no) para poder acceder a la RaggMatrix.

.- Código:

```
class RaggMatrix{

        private:

                int ** matriz;
                int num_filas;
                int *num_columnas;

        public:

        /**
        * @brief constructor por defecto
        */
        RaggMatrix();

        /**
        * @brief Constructor con parametros
        * @param num_filas, el número de filas de la matriz
        * @param num_columnas, el número de columnas en cada fila
        */
        RaggMatrix(int num_filas,int *num_columnas);

        /**
```

```
        * @brief Destructor
        */
        ~RaggMatrix();

        /**
        * @brief Devuelve el número de filas
        * @return Const int
        */
        const int GetFilas();

        /**
        * @brief Devuelve el número de columnas de una fila
        * @param indice, el número de la fila
        * @return Const int
        */
        const int GetColums(int indice);


        /**
        * @brief Devuelve el valor de la posición en la matriz
        * @param f, el número de la fila
        * @param c, el número de la columna
        * @return int
        */
        int at(int f, int c);

        /**
        * @brief Devuelve el valor de la posición en la matriz
        * @param f, el número de la fila
        * @param c, el número de la columna
        * @return const int
        */
        const int at(int f, int c);

        /**
        * @brief operator de igualacion de la RaggMatrix que iguala dos RaggMatrix
        * @param la RaggMatrix a copiar constante
        * @return devuelve la misma matriz que la copiada
        */
        RaggMatrix operator=(const RaggMatrix &orig);
};
```

2. Clase Pieza

.- Presentación:

Esta clase Pieza va a ser la destinada a las distintas piezas del juego, compuesta en su parte privada por la RaggMatrix (int), que le dará el color como se ha explicado antes y el bool "estado" que nos mostrará cuando la pieza está en movimiento o no.

En la parte pública tenemos los constructores y destructores de la clase, una clase de funciones con la que poder trabajar con la Raggmatrix que representa la pieza.

.- Código:

```
class Pieza {

        private:

                RaggMatrix pieza;
                bool estado;

        public:

                /**
                * @brief constructor por defecto
                */
                Pieza();

                /**
                * @brief Constructor con parametros
                * @param orig, matriz para copiar
                */
                Pieza(RaggMatrix orig);

                /**
                * @brief Destructor
                */
                ~Pieza();

                /**
                * @brief Gira la matriz Raggmatrix segun la opcion que se elija
                * @param opcion, letra para saber si girar a la izquierda o a la derecha la
                matriz
```

```
*/
void GirarPieza(char opcion); // [*q / *e]

/**
* @brief Devuelve la matriz por referencia para que utilice los metodos para sacar las distintas dimensiones y datos
* @return Const RaggMatrix
*/
const RaggMatrix & pieza_dimensiones();

/**
* @brief Devuelve el numero, para saber su color
* @return const int
*/
const int GetColor();

/**
* @brief Establece el color de la matriz de la pieza, también lo puede sobreescribir
* @return int (el color elegido).
*/
int SetColor();

/**
* @brief Establece el Flag del booleano “estado” a false
*/
void SetStop();

/**
* @brief Establece el Flag del booleano “estado” a true
*/
void SetRun();

/**
* @brief operator de igualacion de la RaggMatrix que iguala dos RaggMatrix
* @param orig, la RaggMatrix a copiar constante
* @return devuelve la misma matriz que la copiada
*/
friend Pieza operator=(const Pieza&orig);

/**
* @brief Devuelve el booleano “estado”
* @return const bool
*/
const bool Estado();
};
```

3. Clase tablero

.- Presentación:

Esta clase es una de las más importantes, ya que será donde se verá representado el juego en sí, en cuanto a su parte privada, tenemos de variables.

En primer lugar, hallamos una RaggMatrix, la cual la utilizaremos para ir añadiendo las piezas correspondientes, con sus propios métodos para ir recorriendo para saber si una línea está completa o no entre otras. La logística que hemos seguido para determinar el tablero son los números, si una casilla es 0, quiere decir que esta vacía, sin embargo si tiene algun otro numero, esta ocupada, ya sea con un color o otro.

Y un objeto "Pieza Principal", el cual se utilizara para ir moviendo o girando esta por el tablero.Una vez que se coloca una pieza, esta pierde sus propiedades como pieza principal y se inserta una nueva.

.- Código:

```
Class Tablero {

        private:

                RaggMatrix tablero;
                Pieza principal;
        public:

                /**
                * @brief constructor por defecto
                */
                Tablero();

                /**
                * @brief Constructor con parámetros, recibe una matriz a copiar
                * @param orig, matriz para copiar
                * @param color, el valor de enteros de la pieza
                */
```

```
Tablero(RaggMatrix orig);

/**
* @brief Constructor con parámetros, recibe un número de filas y un puntero de columnas para copiar con valor predeterminado 0
* @param num_filas, número de filas.
* @param *num_columnas, puntero con número de columnas para cada fila
*/
Tablero(int num_filas,int *num_columnas);

/**
* @brief Destructor
*/
~Tablero();

/**
* @brief Devuelve la matriz por referencia para que utilice los metodos para sacar las distintas dimensiones y datos
* @return Const RaggMatrix
*/
const RaggMatrix & tablero_dimensiones();

/**
* @brief Añade una pieza al tablero, en la posicion central de este, estableciendola como la principal, para poder ir modificandola.
* @param pieza, la pieza nueva que se va a insertar y a convertir en la principal.
* @return const Pieza, la pieza en sí por referencia.
*/
const Pieza &  AñadirPieza (const Pieza & pieza);

/**
* @brief Modifica el tablero para poder ir moviendo la pieza por este, según la opción que elija, ya sea moverse o rotar
* @param opcion, la tecla que se pulse, para determinar la opcion
*/
void MoverPieza (char opccion );

/**
* @brief Detecta si la pieza principal se choca con alguna otra pieza del tablero, y se establece el flag de Estado a false de esta.
*/
void Choca();

/**
* @brief Comprueba recorriendo el tablero si hay alguna fila entera completa, si la esta ponemos la flag a true sino a false
```

```
        * @return bool
        */
        bool CheckLineaCompleta();

        /**
        * @brief Una vez comprobado si la línea está completa, dependemos de
        esta para limpiarla, y eliminar la fila completa.
        */
        void EliminarLinea();
        /**
        * @brief Devuelve el valor de la posición en la matriz
        * @param f, el número de la fila
        * @param c, el número de la columna
        * @return const int
        */
        const int at(int f, int c);
        /**
        * @brief Devuelve el valor de la posición en la matriz
        * @param f, el número de la fila
        * @param c, el número de la columna
        * @return const int
        */
        int at (int indice);
        /**
        * @brief Devuelve la Pieza principal por referencia
        * @return Const Pieza
        */
        const Pieza & Principal();
        /**
        * @brief Devuelve la Pieza principal por referencia
        * @return Pieza
        */
        Pieza & Principal();
        /**
        * @brief Limpiamos el tablero, poniendo todas las casillas al valor
        predeterminado de 0
        */
        void LimpiarTablero();

        /**
        * @brief operator de igualacion de la RaggMatrix tablero que iguala dos
        tableros (RaggMatrix).
        * @param orig, la RaggMatrix a copiar constante
        * @return devuelve  el mismo tablero que el copiado
        */
        friend Tablero operator=(const Tablero &orig);

};
```

4. Cola Piezas

.- Presentación:

Esta clase es donde se almacenarán las piezas que se ubiquen en la cola, está consta de una parte privada donde habrá un vector de piezas donde se almacenarán estas mismas.

También un int número de piezas que contará el número de piezas en cola en ese momento.

.- Código:

```
class ColaPiezas{

        private:

                Pieza *cola;
                int num_piezas;

        public:
                me empalmo conttigoaidos
                /**
                * @brief constructor por defecto de la clase ColaPiezas
                */
                ColaPiezas();

                /**
                * @brief constructor que recibe un numero de piezas y un array de
                piezas para convertirlo en una clase ColaPiezas
                *@param *cola, array de pieza
                *@param n_piezas,numero de piezas
                */
                ColaPiezas(pieza *cola, int n_piezas)

                /**
                * @brief destructor de la clase ColaPiezas
                */
                ~ColaPiezas();

                /**
                * @brief  funcion vacia que elimina una pieza
                */
                void EliminarPieza();
```

```
        /**
        * @brief  funcion vacia que genera una nueva pieza
        *@param piezaa, pieza a añadir hola bb no?
        *@return devuelve la pieza a añadir
        */
        const Pieza & AñadirNuevaPieza(const Pieza & orig);

        /**
        * @brief  función no constante para acceder a una pieza
        *@param Indice, indice para poder acceder al vector de piezas
        */
        Pierza & ConsultarPieza(int Indice);

        /**
        * @brief  función constante para acceder a una pieza
        *@param Indice, indice para poder acceder al vector de piezas
        */
        const Pierza & ConsultarPieza(int Indice);

        /**
        * @brief  operator de copia de la clase ColaPiezas
        *@param orig, ColaPiezas a copiar
        *@return devuelve la cola implícitamente copiado
        */
        ColaPiezas operator=(const ColaPiezas &orig);
};
```

5. Clase Estadisticas

.- Presentación:

En la clase estadística almacenaremos todas las variables ajenas de forma directa con el tablero como es el nivel, número de piezas, etc… en forma de variables en la parte privada de la misma.

.- Código:

```
class Estadisticas{

        private:

                int nivel;
                int lineas_completadas;
                int num_piezas;


        public:

                /**
                * @brief  constructor por defecto de la clase Estadisctica
                */
                Estadisticas();

                /**
                * @brief  constructor de la clase Estadisctica
                *@param lineas_ocupadas, número de líneas ocupadas por el jugador
                *@param num_piezas, el número de piezas que ha usado el jugador
                */
                Estadisticas(int nivel, int lineas_ocupadas, int num_piezas);

                /**
                * @brief  destructor de la clase estadistica
                */
                ~Estadisticas();

                /**
                * @brief  funcion consultora de nivel
                *@return devuelve el int que hace referencia al nivel
                */
                const int ConsultarNivel();

                /**
                * @brief  funcion consultora de num_lineas
```

```
*@return devuelve el int que hace referencia al num_lineas
*/
const int Num_Lineas();

/**
* @brief  funcion consultora de num_piezas
*@return devuelve el int que hace referencia al num_piezas
*/
const int Num_Piezas();

/**
* @brief  funcion set de num_lineas
*@param linea, numero de lineas
*/
void SetLineas(int linea);

/**
* @brief  funcion set de num_piezas
*@param numero, numero de piezas
*/
void SetNum(int numero);


/**
* @brief  funcion set de nivel
*@param nivell, numero del nivel
*/
void SetNivel(int nivell);


/**
* @brief  funcion reset de las estadisticas
*/
void Reset();


};
```

6. *Clase Juego*

.- Presentación:

Esta clase es el coordinador del proyecto, esta clase de las diversas secciones principales del juego.

En primer lugar, hallamos que este se encargará de iniciar el juego (cargar el fichero y leer las distintas opciones), con las distintas variables permitidas.

Por otro lado, también se centra en el movimiento de las piezas principales y la introducción de los valores enteros dentro de la matriz tablero. Así como, ir comprobando que todo funciona bien y comprobando que el tablero se va modificando correctamente, cuando se rellena por ejemplo una fila entera.

Por último, podemos comentar, la posibilidad de poder guardar el juego, con sus estadísticas y sus opciones configuradas.

.- Código:

```
class Juego {

        private:
                Estadisticas datos;
                ColaPiezas piezas;
                Tablero juego;
                bool estado_juego;

        public:

        /**
        * @brief  constructor por defecto de la clase Juego
        */
        Juego();
        /**
        * @brief  constructor por defecto de la clase Juego
        *@param dato, variable de tipo Estadistica
        *@param tableros, variable de tipo RaggMatrix
        *@param pieza,  variable de tipo ColaPiezas
        *@param juegos, variable de tipo Tablero
```

```
*@param estado_juego, variable que indica si el juego esta en pause o no
*/
Juego(Estadisticas dato,RaggMatrix tableros,ColaPiezas pieza,Tablero juegos, bool
estado_juego);

/**
* @brief  constructor de copia
*/
Juego(const Juego & orig);

/**
* @brief  Método inicial, para poder iniciar todos los métodos de manera
general.
*/
void IniciarJuego ();

/**
* @brief  función cargar Juego de un fichero
*@param fichero, nombre del fichero de donde se carga el juego
*@return devuelve el jeugo cargado del fichero
*/
Juego CargarJuego (const char * fichero );

/**
* @brief  función guardar Juego de un fichero
*@param fichero, nombre del fichero de donde se guarda el juego
*@return devuelve el juego guarda del fichero
*/
Juego GuardarJuego (const char * fichero );


/**
* @brief  operator de copia de la clase Juego
*@param orig, Juego a copiar
*@return Juego por referencia
*/
Juego & operator=(const Juego & orig);

/**
* @brief  funcion que indica si esta el tablero lleno o no
*@return bool
*/
const bool Esta_lleno() const ;

/**
* @brief  funcion game over si el tablero esta lleno para acabar el juego
*/
void Game_Over();
```

```
/**
* @brief  Devuelve el tablero del objeto implícito
*@return Tablero (RaggMatrix) por referencia
*/
Tablero & GetTablero();

/**
* @brief  Devuelve el tablero del objeto implícito
*@return Const Tablero (RaggMatrix) por referencia
*/

const Tablero & GetTablero() const;


/**
* @brief  función para acceder a la ColaPiezas de forma const
*@return ColaPiezas
*/
const ColaPiezas & Piezas ();



/**
* @brief  función para acceder a la ColaPiezas de forma no const
*@return ColaPiezas
*/
ColaPiezas & Piezas ();


/**
* @brief  Devuelve el estado en el que se encuentra el juego
*@return Const bool
*/
const bool Estado_Juego();


/**
* @brief  Devuelve el estado en el que se encuentra el juego
*@return bool
*/
bool Estado_Juego();


/**
* @brief  Establece el flag de estado para el booleano “estado_juego” a true
*/
void Start();
```

```
        /**
        * @brief  Establece el flag de estado para el booleano "estado_juego" a false
        */
        void Stop();


        /**
        * @brief  función para acceder a la Estadisticas de forma const
        *@return Estadisticas
        */
        const Estadisticas & GetEstadisticas();




        /**
        * @brief  función para acceder a la Estadisticas de forma no const
        *@return Estadisticas por referencia
        */
        Estadisticas & GetEstadisticas();
        /*
        *brief sobrecarga operator << para poder imprimir la clase
        *param flujo, flujo de salida
        *return flujo ostream
        */
        friend ostream & operator<<(ostream & flujo);

};
```