**RETO 2: TETRIS**

**Elena Abreu Fernández**

**Antonio Cantillo Molina**

**Leandro Jorge Fernández Vega**

**Mario Martín Rodríguez**

**Doble Grado en Ingeniería Informática y Matemáticas UGR**

**Marcador.h**

#include <iostream>

#include <string>

using namespace std;

/\*\*

@brief T.D.A. Marcador

Una instancia del tipo de dato abstracto Marcador que nos permite almacenar datos acerca de una partida tales como los puntos, las piezas colocadas, el estado del juego…

El TDA Marcador proporciona además distintas herramientas para la manipulación de marcadores: constructores, setTitulo, setNivel, setLineas, setPiezas, setEstado, getTitulo, getNivel, getLineas, getPiezas, getEstado, AumentaLinea, AumentaNivel, sobrecarga de operador de salida.

Para poder usar el TDA ColaDePiezas se debe incluir el fichero

\#include <Marcador.h>

\*\*/

**class Marcador {**

**private:**

/\*\*

\*@brief titulo cadena que muestra el nombre del juego

\*/

string titulo;

/\*\*

\*@brief nivel Muestra el número de niveles que hemos completado

\*/

int nivel;

/\*\*

\*@brief lineas Muestra el número de líneas que hemos completado

\*/

int lineas;

/\*\*

\*@brief piezas Muestra el número de piezas que hemos pasado

\*/

int piezas;

/\*\*

\*@brief estado Muestra el estado de nuestra partida, es decir, si se está \*jugando o si la partida ha terminado

\*/

string estado;

**public:**

**//Constructores.**

/\*\*

\* @brief Crea un marcador con todo

\*/

Marcador ();

/\*\*

\* @brief Crea un marcador con el título, nivel, piezas y estado que recibimos

\* @param titulo Cadena con el título para el marcador

\* @param nivel Representa el número de niveles completados

\* @param lineas Número de líneas completadas

\* @param estado Cadena con el estado de nuestra partida

\*/

Marcador (string titulo, int nivel=0, int lineas=0,int piezas=0,string estado=”JUGANDO…”);

**//Setters**

**/\*\***

**\***@brief Modifica el título del juego

\*@param titulo nombre que se le otorga al modo de juego en el que estamos

*\*/*

void setTitulo(string titulo);

/\*\*

\*@brief Indica el nivel de juego en cada instante

\*@param nivel nuevo valor que toma el nivel del juego

\*/

void setNivel(int nivel);

/\*\*

\*@brief Indica el número de líneas completadas hasta el momento

\*@param nlineas nuevo número de líneas completadas

\*/

void setLineas(int nlineas);

/\*\*

\*@brief Modifica el número de piezas colocadas en el tablero

\*@param npiezas número de piezas utilizadas

*\*/*

void setPiezas(int npiezas);

/\*\*

\*@brief Indica si se está jugando, o si la partida ya ha terminado

\*@param estado nueva cadena que otorgamos al estado de nuestro marcador

\*/

void setEstado(string estado);

**//Getters**

/\*\*

\*@briefDevuelve el título del juego

\*/

string getTitulo();

/\*\*

\*@brief Devuelve el nivel actual

\*/

int getNivel();

/\*\*

\*@brief Devuelve el número de líneas utilizadas

\*/

int getLineas();

/\*\*

\*@brief Devuelve el número de piezas utilizadas

\*/

int getPiezas();

/\*\*

\*@brief Devuelve el estado de nuestro marcador

\*/

string getEstado();

**// Operaciones**

/\*\*

\*@brief Aumenta el valor de líneas completadas

\*@param n número de líneas completadas que aumentamos. Por defecto \*vale 1.

\*/

void AumentaLinea(int n=1);

/\*\*

\*@brief Aumenta el valor de niveles completados.

\*@param n número de niveles completados que aumentamos. Por defecto \*vale 1.

\*/

void AumentaNivel(int n=1);

**};**

/\*\*

\*@brief Muestra por pantalla el marcador entero.

\*@param flujo Variable de flujo.

\*@param marcador Objeto tipo marcador a mostrar.

\*/

ostream& operator<<(ostream& flujo, const Marcador& marcador);

**pieza.h**

#include <iostream>

#include <string>

#include <imagen.h>

using namespace std;

/\*\*

@brief T.D.A. Pieza

Una instancia del tipo de dato abstracto Pieza que nos permite modelizar el objeto con el que jugamos al Tetris. Serán de distintos colores, y cada color tiene asociado una forma.

El TDA Pieza proporciona además distintas herramientas para la manipulación de piezas: constructores, setColor, getRows, getCols, getColor, rotar, sobrecarga de operador de salida.

Para poder usar el TDA Imagen se debe incluir el fichero

\#include <Pieza.h>

\*\*/

**class Pieza{**

**private:**

/\*\*

\*@brief Dato tipo struct que caracterizará cada pieza

\* Dependiendo del color (amarillo, rojo, naranja, azul, verde, violeta y celeste)

\* se asignará una forma a la pieza, que se implementará como una matriz

\* con determinadas casillas vacías y rellenas.

*\*/*

string color;

/\*\*

\*@brief Dato tipo Image, en el que por tanto cada pieza será una matriz de \*pixeles

\*/

Image pieza;

/\*\*

\*@brief Función que crea un tipo de matriz según el color.

\*@param colour De entre un rango de siete colores (amarillo, rojo,

\* naranja, azul, verde, violeta y celeste) se pasa uno como parámetro

\* que asociará una forma a la pieza.\*@post Ambos campos se modifican.

\*/

void CreaMatriz (string colour);

**public:**

**//Constructores**

/\*\*

\*@brief Se crea una pieza según el color que tenga asociado

\*@param colour Color de la pieza que determinará su forma.

\*/

Pieza(string colour);

**//Setters**

/\*\*

\*@brief Establece el color de nuestro pieza

\*@param color Nuevo color que establecemos en nuestro pieza

\*/

void setColor(string color);

**// Getters**

/\*\*

\* @brief devuelve las filas de nuestra pieza.

\*/

int getRows();

/\*\*

\*@brief devuelve las columnas de nuestra pieza.

\*/

int getCols();

/\*\*

\*@brief devuelve el color de nuestra pieza.

\*/

string getColor();

**//Operaciones**

/\*\*

\* @brief Rota la pieza a la derecha o a la izquierda según indique la t \*tecla introducida.

\* @param Letra que indica el sentido de rotación.

\* @pre La tecla introducida sólo puede ser q (izq) o w(derecha).

\*/

void rotar(char tecla);

**};**

/\*\*

\* @brief Operador de salida de la clase pieza.

\* @param pieza Indica la pieza que vamos a mostrar.

\* @param variable de flujo.

\*/

ostream& operator<< (ostream& flujo, const Pieza& pieza);

**ColaDePiezas.h**

#include <iostream>

#include <string>

#include <queue>

#include <pieza.h>

using namespace std;

/\*\*

@brief T.D.A. ColaDePiezas

Una instancia del tipo de dato abstracto ColaDePiezas que nos permite establecer una cola, utilizando el TDA Lineal de mismo nombre, donde podremos ver qué piezas son las que próximamente tendremos que colocar siguiendo la filosofía del juego.

El TDA ColaDePiezas proporciona además distintas herramientas para la manipulación de colas de piezas: constructores, getN\_piezas, front, push, pop, empty, getPieza, sobrecarga de operador de indexación.

Para poder usar el TDA ColaDePiezas se debe incluir el fichero

\#include <ColadePiezas.h>

\*\*/

**class ColaDePiezas {**

**private:**

*/\*\**

*\* @brief Dato TDA que usaremos para almacenar las piezas por salir a*

*\* tablero*

*\*/*

queue <Pieza> cola;

**public:**

**// Constructores**

/\*\*

\*@brief Crea una cola de piezas vacía.

\*/

ColaDePiezas();

**// Getters**

/\*\*

\*@brief devuelve el número de piezas de nuestra cola.

\*/

int getN\_piezas();

/\*\*

\*@brief Devuelve la pieza que se encuentra en el frente de la cola.

\*/

Pieza front() const;

/\*\*

\*@brief Devuelve la pieza que se encuentra en la posición n.

\*@param n Posición en la cual se encuentra la pieza que queremos obtener.

\*@pre n > 0

\*/

const Pieza getPieza(int n) const;

/\*\*

\* @brief Sobrecarga del operador de indexación que devuelve la pieza de la \* posición n. Para ello, vuelca la cola de piezas en un auxiliar, borra todos las \* piezas previas a la de la posición n y devuelve la pieza que queríamos que \* quedaría en el front.

\* @param n Posición de la pieza en la cola de piezas.

\* @pre n > 0

\*/

const Pieza& operator[] (int n) const;

**// Operaciones**

/\*\*

\*@brief Añade una nueva pieza a la cola.

\*@param p Pieza que vamos a añadir a la cola.

\*/

void push(const Pieza &p);

/\*\*

\*@brief Elimina el frente de la cola.

\*/

void pop();

/\*\*

\*@brief Determina si la cola está vacía.

\*/

bool empty();

**};**

/\*\*

\*@brief Operador de salida que muestra todos los datos de una cola de \*piezas.

\*@param flujo Flujo sobre el que hacer la salida.

\*@param otra Estado de la cola de piezas que mostrar por pantalla.

\*/

ostream& operator<<(ostream& flujo, const ColadePiezas& otra);

**Tablero.h**

#include <string>

#include <imagen.h>

#include <iostream>

/\*\*

@brief T.D.A. Tablero

Una instancia del tipo de dato abstracto Tablero que nos permite establecer el tablero de juego sobre el que jugaremos al Tetris y donde colocaremos las distintas piezas.

El TDA Tablero proporciona además distintas herramientas para la manipulación de tableros: constructores, getRows, getCols, moverPieza, insertarPieza, EncajaPieza, LineaCompleta, EliminaFila, BajaFilas, sePuedeRotar, lim\_sup\_alcanzado, numLinCompletas, AniadirFilaAleatoria.

Para poder usar el TDA Tablero se debe incluir el fichero

\#include <Tablero.h>

\*\*/

**class Tablero {**

**private:**

/\*\*

\*@brief Tenemos un tablero que es una imagen, por tanto una matriz de \*píxeles en el cual tendremos todas nuestras piezas y será el lugar donde se \*desarrollará nuestro juego.

\*/

Image tablero;

/\*\*

\*@brief Baja todas las filas superiores a una recién eliminada.

\*@pre 0<=fila<getCols()

\*@post tablero se modifica.

\*/

void BajaFilas(int fila);

/\*\*

\*@brief Comprueba si una pieza puede bajar más.

\*/

bool EncajaPieza();

/\*\*

\*@brief Comprueba si una pieza se puede rotar.

\*/

bool sePuedeRotar();

**public:**

**// Constructores**

/\*\*

\*@brief Crea un tablero vacío.

\*/

Tablero();

/\*\*

\*@brief Crea un tablero a partir de @rows y @cols.

\*@param rows Número de filas del tablero.

\*@param rows Número de columnas del tablero.

\*/

Tablero(int rows, int cols);

**// Getters**

/\*\*

\*@brief Devuelve el número de filas de nuestro tablero.

\*/

int getRows();

/\*\*

\*@brief Devuelve el número de columnas de nuestro tablero.

\*/

int getCols();

**// Operaciones**

/\*\*

\* @brief Rota la pieza a la derecha o a la izquierda según indique la tecla

\* introducida

\* @param Letra que indica la dirección.

\* @pre La tecla introducida sólo puede ser la flecha de abajo y las flechas de \* izquierda y derecha.

\*@post tablero se modifica.

\*/

void moverPieza(char flecha);

/\*\*

\*@brief Inserta una Pieza

\*@param pieza Representa la pieza a insertar.

\*@pre La columna de inserción debe tener espacio.

\*@post tablero se modifica.

\*/

void insertarPieza(const Pieza& pieza);

/\*\*

\*@brief Comprueba si una línea está completa.

\*@pre 0<=fila<getCols()

\*/

bool LineaCompleta(int fila);

/\*\*

\* @brief Devuelve el número total de líneas completas.

\*/

int numLinCompletas();

/\*\*

\*@brief Elimina una línea completa y baja las demás, actualizando el nivel del \*marcador.

\*@pre 0<=fila<getCols()

\*@post tablero && campo nivel se modifica.

\*/

void EliminaFila(int fila);

/\*\*

\*@brief Comprueba si la columna de inserción está completa. Es decir, \*comprueba si has alcanzado el límite superior del tablero en la columna \*central, lo que indica que se ha perdido la partida.

\*/

bool lim\_sup\_alcanzado();

/\*\*

\* @brief Añade una fila aleatoria

\*/

void AniadirFilaAleatoria();

**};**

/\*\*

\*@brief Operador de salida que muestra el tablero.

\*@param flujo Flujo sobre el que hacer la salida.

\*@param otro Estado del tablero que mostrar por pantalla.

\*/

ostream& operator<<(ostream& flujo, const Tablero& otro);

**Imagen.h**

#include <cstdlib>

#include <imageIO.h>

typedef unsigned char byte;

enum LoadResult: unsigned char {

SUCCESS,

NOT\_PGM,

READING\_ERROR

};

/\*\*

@brief T.D.A. Imagen

Una instancia del tipo de dato abstracto Imagen nos permite almacenar imágenes de intensidades.

El TDA Imagen proporciona además distintas herramientas para la manipulación de dichas imágenes.

Para poder usar el TDA Imagen se debe incluir el fichero

\#include <Image.h>

\*\*/

**class Image{**

/\*\*

@page page\_repImagen Representación del TDA Imagen

@section sec\_Image\_A Título A

Contenido de la sección A.

@section sec\_Image\_B Título B

Contenido de la sección B.

Referencia a la \ref sec\_Image\_A

\*\*/

**private :**

/\*\*

@brief Puntero a la imagen almacenada

img apunta a un array-2D dinámico de bytes que contiene la imagen en sí. Almacena tantos bytes como pixeles tenga la imagen.

\*\*/

byte \*\*img;

/\*\*

@brief Número de filas de la imagen.

\*\*/

int rows;

/\*\*

@brief Número de columnas de la imagen.

\*\*/

int cols;

/\*\*

@brief Initialize una imagen.

@param nrows Número de filas que tendrá la imagen. Por defecto, 0

@param ncols Número de colwnnas que tendrá la imagen. Por defecto, 0

@param buffer Puntero a un buffer de datos con los que rellenar los píxeles de la imagen. Por defecto, 0.

@pre filas >= 0 y columnas >= 0

@post Reserva memoria para almacenar la imagen y la prepara para usarse.

\*\*/

void Initialize (int nrows= 0, int ncols= 0, byte \*buffer= 0);

/\*\*

@brief Lee una imagen PGM desde un archivo.

@param file\_path Ruta del archivo a leer

@return LoadResult

\*\*/

LoadResult LoadFromPGM(const char \* file\_path);

/\*\*

@brief Copy una imagen .

@param orig Referencia a la imagen original que vamos a copiar

@pre Asume que no hay memoria reservada o se ha llamado antes a Destroy()

@pre Asume this != &orig

\*\*/

void Copy(const Image &orig);

/\*\*

@brief Reserva o copia en memoria una imagen.

@param nrows Número de filas que tendrá la imagen.

@param ncols Número de colwnnas que tendrá la imagen.

@param buffer Puntero a un buffer de datos con los que rellenar los píxeles de la imagen. Por defecto, 0.

@pre nrows >= 0 y ncols >= 0

@post Reserva memoria para almacenar la imagen y la prepara para usarse.

\*\*/

void Allocate(int nrows, int ncols, byte \* buffer = 0);

/\*\*

\* @brief Destroy una imagen

\*

\* Libera la memoria reservada en la que se almacenaba la imagen que llama a la función.

\* Si la imagen estaba vacía no hace nada .

\*/

void Destroy();

**public :**

/\*\*

\* @brief Constructor por defecto .

\* @post Genera una instancia de la clase Imagen con 0 filas y 0 colunmas.

\* @return Imagen, el objeto imagen creado.

\*/

Image();

/\*\*

\* @brief Constructor con parámetros.

\* @param nrows Número de filas de la imagen.

\* @param ncols Número de columnas de la imagen.

\* @param value defecto Valor con el que inicializar los píxeles de la imagen. Por defecto 0.

\* @pre n fils > 0 Y n\_cols > 0

\* @post La imagen creada es de n\_fils y n\_cols columnas. Estará inicializada al valor por defecto.

\* @return Imagen, el objeto imagen creado.

\*/

Image(int nrows, int ncols, byte value=0);

/\*\*

\* @brief Constructor de copias.

\* @param orig Referencia a la imagen original que se quiere copiar.

\* @return Imagen, el objeto imagen creado.

\*/

Image (const Image & orig);

/\*\*

\* @brief Operador de tipo destructor.

\* @return void

\* @post El objeto Imagen destruido no puede usarse salvo que se haga sobre él una operacion Imagen().

\*/

~Image() ;

/\*\*

\* @brief Operador de asignación .

\* @param orig Referencia a la imagen original que desea copiarse.

\* @return Una referencia al objeto imagen modificado.

\* @post Destroy cualquier información que contuviera previamente la imagen que llama al operador de asignación.

\*/

Image & operator= (const Image & orig);

/\*\*

\* @brief Funcion para conocer si una imagen está vacía.

\* @return Si la imagene está vacía

\* @post la imagen no se modifica.

\*/

bool Empty() const;

/\*\*

\* @brief Filas de la imagen .

\* @return El número de filas de la imagen.

\* @post la imagen no se modifica.

\*/

int get\_rows() const;

/\*\*

\* @brief Columnas de la imagen.

\* @return El número de columnas de la imagen.

\* @post la imagen no se modifica.

\*/

int get\_cols() const;

/\*\*

\* @brief Devuelve el número de píxeles de la imagen.

\* @return número de píxeles de la imagen.

\* @post la imagen no se modifica.

\*/

int size() const;

/\*\*

\* @brief Asigna el valor valor al píxel (@p i, @p j) de la imagen.

\* @param i Fila de la imagen en la que se encuentra el píxel a escribir .

\* @param j Columna de la imagen en la que se encuentra el píxel a escribir.

\* @param value Valor que se escribirá en el píxel (@p i, @p j) .

\* @pre 0 <= @p i < get\_rows()

\* @pre 0 <= @p j < get\_cols()

\* @pre 0 <= @p value <= 255

\* @post El píxel (@p i, @p j) de la imagen se modificará y contendrá valor @p value.

\* Los demás píxeles permanecerán iguales.

\*/

void set\_pixel (int i, int j, byte value);

/\*\*

\* @brief Consulta el valor del píxel (fil, col) de la imagen.

\* @param i Fila de la imagen en la que se encuentra el píxel a consultar.

\* @param j Columna de la imagen en la que se encuentra el píxel a consultar.

\* @pre 0 <= fil < I.Num\_ Filas() II 0 <= col < I.Num Columnas()

\* @return el valor del píxel contenido en (fil,col)

\* @post La imagen no se modifica.

\*/

byte get\_pixel (int i, int j) const;

/\*\*

\* @brief Consulta el valor del píxel k de la imagen desenrrollada.

\* @param k Índice del píxel

\* @pre 0 <= k < filas\*columnas

\* @return el valor del píxel contenido en (k/filas,k%filas)

\* @post La imagen no se modifica.

\*/

byte get\_pixel (int k) const;

/\*\*

\* @brief Asigna el valor valor al píxel k de la imagen desenrollada.

\* @param k Índice del píxel a escribir .

\* @param value Valor que se escribirá en el píxel k.

\* @pre 0 <= k < filas\*columnas && 0 <= valor <= 255

\* @post El píxel k se modificará con el valor de value.

\*/

void set\_pixel (int k, byte value);

/\*\*

\* @brief Almacena imágenes en disco.

\* @param file\_path Ruta donde se almacenará la imagen.

\* @pre file path debe ser una ruta válida donde almacenar el fichero de salida.

\* @return Devuelve true si la imagen se almacenó con éxito y false en caso contrario.

\* @post La imagen no se modifica.

\*/

bool Save (const char \* file\_path) const;

/\*\*

\* @brief Carga en memoria una imagen de disco .

\* @param file\_path Ruta donde se encuentra el archivo desde el que cargar la imagen.

\* @pre @p file\_path debe ser una ruta válida que contenga un fichero . pgm

\* @return Devuelve @b true si la imagen se carga con éxito y @b false en caso contrario.

\* @post La imagen previamente almacenada en el objeto que llama a la función se destruye.

\*/

bool Load (const char \* file\_path);

#endif // IMAGEN\_H

**imageIO.h**

/\*\*

\* @file imageIO.h

\* @brief Fichero cabecera para la E/S de imágenes

\*

\* Permite la E/S de archivos de tipo PGM,PPM

\*

\*/

#ifndef \_IMAGEN\_ES\_H\_

#define \_IMAGEN\_ES\_H\_

/\*\*

\* @brief Tipo de imagen

\*

\* Declara una serie de constantes para representar los distintos tipos

\* de imágenes que se pueden manejar.

\*

\* @see ReadImageKind

\*/

enum ImageKind {IMG\_UNKNOWN, IMG\_PGM, IMG\_PPM};

/\*\*

\* @brief Devuelve el tipo de imagen del archivo

\*

\* @param path indica el archivo de disco que consultar

\* @return Devuelve el tipo de la imagen en el archivo

\*

\* @see TipoImagen

\*/

ImageKind ReadImageKind (const char \*path);

/\*\*

\* @brief Lee una imagen de tipo PGM

\*

\* @param path archivo a leer

\* @param rows Parámetro de salida con las filas de la imagen.

\* @param cols Parámetro de salida con las columnas de la imagen.

\* @return puntero a una nueva zona de memoria que contiene @a filas x @a columnas

\* bytes que corresponden a los grises de todos los píxeles

\* (desde la esquina superior izqda a la inferior drcha). En caso de que no

\* no se pueda leer, se devuelve cero. (0).

\* @post En caso de éxito, el puntero apunta a una zona de memoria reservada en

\* memoria dinámica. Será el usuario el responsable de liberarla.

\*/

unsigned char \*ReadPGMImage (const char \*path, int& rows, int& cols);

/\*\*

\* @brief Escribe una imagen de tipo PGM

\*

\* @param path archivo a escribir

\* @param datos punteros a los @a f x @a c bytes que corresponden a los valores

\* de los píxeles de la imagen de grises.

\* @param rows filas de la imagen

\* @param cols columnas de la imagen

\* @return si ha tenido éxito en la escritura.

\*/

bool WritePGMImage (const char \*path, const unsigned char \*datos,

const int rows, const int cols);

#endif

/\* Fin Fichero: imagenES.h \*/

**Partida.h**

#include <Tablero.h>

#include <ColadePiezas.h>

#include <Marcador.h>

/\*\*

@brief T.D.A. Partida

Una instancia del tipo de dato abstracto Partida que nos permite crear partidas de Tetris, cada una con su propio marcador, tablero, cola de piezas. Además, nos dará permitirá terminar una partida, pausarla, reiniciarla o saber si hemos perdido.

El TDA Partida proporciona además distintas herramientas para la manipulación de partidas: constructores, Perder, Pausar, Reiniciar, Terminar, sobrecarga del operador de salida.

Para poder usar el TDA Partida se debe incluir el fichero

\#include <Partida.h>

\*\*/

**class Partida {**

**private:**

Tablero MiTablero;

ColaDePiezas MiCola;

Marcador MiMarca;

**public:**

**// Constructores**

/\*\*

\* @brief Constructor con argumentos

\* @param titulo Modo de juego a iniciar. Por defecto sería una partida clásica.

\* @pre titulo debe estar entre las opciones de partida programadas.

\*/

Partida(string titulo=TETRIS);

**// Operaciones**

/\*\*

\* @brief Comprueba si se ha perdido la partida.

\* @post El estado del marcador se modifica si de verdad se ha alcanzado el \*límite superior de la fila de inserción de piezas en el tablero.

\*/

bool Perder();

/\*\*

\* @brief Pausa la partida.

\*/

void Pausar();

/\*\*

\*@brief Reinicia la partida.

\*@post Cada campo vuelve a su estado inicial.

\*/

void Reiniciar();

/\*\*

\*@brief Termina la partida, ya sea voluntariamente o porque se ha perdido.

\*/

void Terminar();

**};**

/\*\*

\*@brief Operador de salida que muestra todos los datos de una partida.

\*@param flujo Flujo sobre el que hacer la salida.

\*@param otra Estado de la partida que mostrar por pantalla.

\*/

ostream& operator<<(ostream& flujo, const Partida& otra);

**CONCLUSIÓN**

Las clases para jugar una partida estándar se han implementado. Si se admitieran varios modos de juego habría que realizar modificaciones específicas como añadir al marcador una cuenta atrás si se selecciona un modo contrarreloj, añadir un método *void ganar()* si se debe alcanzar una puntuación determinada, etc.