|  |  |  |
| --- | --- | --- |
| **CARRERA:**  Ingeniería en Sistemas e informática | **GUÍA**  No. 02 | **TIEMPO ESTIMADO:** |
| **ASIGNATURA:**  Estructura de Datos  NRC: 2969 | **FECHA DE ELABORACION: 28**-01-2020  **SEMESTRE**: Octubre 2019 – Enero 2020 | |
| **TÍTULO:**  Proyecto Parcial | **DOCENTE: Fernando Solis** | |

**OBJETIVO**

p.e. Aplicar los conocimientos adquiridos en el tercer parcial en un proyecto.

**INSTRUCCIONES**

**p.e.**

1. Utilice como material principal, las principales herramientas para desarrollo en C++
2. Utilice información consultada en Internet y conocimiento adquirido en clase.

**ACTIVIDADES**

1. **Ubicación de recursos**

**p.e.**

1. Formar grupos de máximo 2 personas
2. Instalar el IDE para C++
3. **Planteamiento del problema**

**p.e. Ejercicio No 1**

Desarrollar un Tetris numérico usando listas enlazadas.

Este proyecto consiste en un juego retro que consiste en llenar piezas en un tablero, en esta aplicación realizaremos un cambio de funcionalidad al usar listas enlazadas.

**Marco Teórico**

**Tetris**

Es un videojuego de puzzle originalmente diseñado y programado por Alekséi Pázhitnov en la Unión Soviética. Fue lanzado el 6 de junio de 1984, mientras trabajaba para el Centro de Computación Dorodnitsyn de la Academia de Ciencias de la Unión Soviética en Moscú, RSFS de Rusia. ​ Su nombre deriva del prefijo numérico griego tetra- (todas las piezas del juego, conocidas como Tetrominós que contienen cuatro segmentos) y del tenis, el deporte favorito de Pázhitnov.

**Algoritmo**

es un conjunto de instrucciones o reglas definidas y no-ambiguas, ordenadas y finitas que permite, típicamente, solucionar un problema, realizar un cómputo, procesar datos y llevar a cabo otras tareas o actividades.

**Lista Enlazada**

Es una de las estructuras de datos fundamentales, y puede ser usada para implementar otras estructuras de datos. Consiste en una secuencia de nodos, en los que se guardan campos de datos arbitrarios y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento. También es un tipo de dato autor referenciado porque contienen un puntero o enlace a otro dato del mismo tipo. Las listas enlazadas permiten inserciones y eliminación de nodos en cualquier punto de la lista en tiempo constante (suponiendo que dicho punto está previamente identificado o localizado), pero no permiten un acceso aleatorio. Existen diferentes tipos de listas enlazadas: listas enlazadas simples, listas doblemente enlazadas, listas enlazadas circulares y listas enlazadas doblemente circulares.

**Conclusiones**

* La potencia que nos brindan las listas enlazadas se puede ver reflejada en muchas aplicaciones.
* Con algo de imaginación se pueden desarrollar muchas aplicaciones que pueden marcar un antes y después.
* La programación empieza desde lo más básico hasta lo mas complicado.

**Codigo**

#define \_CRT\_SECURE\_NO\_DEPRECATE

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <fstream>

void encriptar(int aumento, const char\* newNombreArchivo, const char\* newNombreArchivoEncriptado) {

char c;

const char\* nombreArchivo = newNombreArchivo;//"solucion.txt"

const char\* nombreArchivoEncriptado = newNombreArchivoEncriptado;//"solucionencriptada.txt";

FILE\* ptr;

FILE\* ptr2;

#pragma warning (disable : 4996)

ptr = fopen(nombreArchivo, "r");

ptr2 = fopen(nombreArchivoEncriptado, "w");

if (ptr == NULL || ptr2 == NULL) {

cout<<"No se pudo abrir los archivos..."<<endl;

system("pause");

return;

}

//ABRO EL ARCHIVO Y A CADA CARACTER LO MODIFICO

while (fscanf(ptr, "%c", &c) != EOF) {

putc((c + aumento), ptr2);

}

cout<<"Se escribio correctamente en el archivo..."<<endl;

fclose(ptr);

fclose(ptr2);

system("pause");

}

void desencriptar(int aumento, const char\* newNombreArchivoEncriptado, const char\* newNombreArchivoDesencriptado) {

char c;

const char\* nombreArchivoEncriptado = newNombreArchivoEncriptado;//"solucionencriptada.txt";

const char\* nombreArchivoDesencriptado = newNombreArchivoDesencriptado;//"soluciondesencriptado.txt";

FILE\* ptr;

FILE\* ptr2;

ptr = fopen(nombreArchivoEncriptado, "r");

ptr2 = fopen(nombreArchivoDesencriptado, "w");

if (ptr == NULL || ptr2 == NULL) {

printf("No se pudo abrir los archivos...\n");

system("pause");

return;

}

//ABRO EL ARCHIVO Y A CADA CARACTER LO MODIFICO

while (fscanf(ptr, "%c", &c) != EOF) {

putc((c - aumento), ptr2);

}

printf("\nSe desencripto correctamente en el archivo...\n");

fclose(ptr);

fclose(ptr2);

system("pause");

}

#include "Cubo.h"

int Cubo::getFila(void)

{

return fila;

}

void Cubo::setFila(int newFila)

{

fila = newFila;

}

int Cubo::getColumna(void)

{

return columna;

}

void Cubo::setColumna(int newColumna)

{

columna = newColumna;

}

int Cubo::getValor(void)

{

return valor;

}

void Cubo::setValor(int newValor)

{

valor = newValor;

}

#include <iostream>

#include <windows.h>

static HWND hConWnd;

HWND BCX\_Bitmap(char\*, HWND = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0, int = 0);

HWND GetConsoleWndHandle(void);

HWND BCX\_Bitmap(char\* Text, HWND hWnd, int id, int X, int Y, int W, int H, int Res, int Style, int Exstyle)

{

HWND A;

HBITMAP hBitmap;

// set default style

if (!Style) Style = WS\_CLIPSIBLINGS | WS\_CHILD | WS\_VISIBLE | SS\_BITMAP | WS\_TABSTOP;

// form for the image

A = CreateWindowEx(Exstyle, "static", NULL, Style, X, Y, 0, 0, hWnd, (HMENU)id, GetModuleHandle(0), NULL);

// Text contains filename

hBitmap = (HBITMAP)LoadImage(0, Text, IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);

// auto-adjust width and height

if (W || H) hBitmap = (HBITMAP)CopyImage(hBitmap, IMAGE\_BITMAP, W, H, LR\_COPYRETURNORG);

SendMessage(A, (UINT)STM\_SETIMAGE, (WPARAM)IMAGE\_BITMAP, (LPARAM)hBitmap);

if (W || H) SetWindowPos(A, HWND\_TOP, X, Y, W, H, SWP\_DRAWFRAME);

return A;

}

// tricking Windows just a little ...

HWND GetConsoleWndHandle(void)

{

HWND hConWnd;

OSVERSIONINFO os;

char szTempTitle[64], szClassName[128], szOriginalTitle[1024];

os.dwOSVersionInfoSize = sizeof(OSVERSIONINFO);

GetVersionEx(&os);

// may not work on WIN9x

if (os.dwPlatformId == VER\_PLATFORM\_WIN32s) return 0;

GetConsoleTitle(szOriginalTitle, sizeof(szOriginalTitle));

sprintf(szTempTitle, "%u - %u", GetTickCount(), GetCurrentProcessId());

SetConsoleTitle(szTempTitle);

Sleep(60);

// handle for NT and XP

hConWnd = FindWindow(NULL, szTempTitle);

SetConsoleTitle(szOriginalTitle);

// may not work on WIN9x

if (os.dwPlatformId == VER\_PLATFORM\_WIN32\_WINDOWS)

{

hConWnd = GetWindow(hConWnd, GW\_CHILD);

if (hConWnd == NULL) return 0;

GetClassName(hConWnd, szClassName, sizeof(szClassName));

// while ( \_stricmp( szClassName, "ttyGrab" ) != 0 )

while (strcmp(szClassName, "ttyGrab") != 0)

{

hConWnd = GetNextWindow(hConWnd, GW\_HWNDNEXT);

if (hConWnd == NULL) return 0;

GetClassName(hConWnd, szClassName, sizeof(szClassName));

}

}

return hConWnd;

}

#ifndef MENU\_H\_INCLUDED

#define MENU\_H\_INCLUDED

#include<conio.h>

#include <iostream>

#include <windows.h>

#define ARRIBA 72

#define ABAJO 80

#define ENTER 13

using namespace std;

class Menu {

private:

public:

Menu();

void gotoxy(USHORT, USHORT);

int menu(string titulo, string\* opciones, int n);

};

Menu::Menu()

{

}

int Menu::menu(string titulo, string\* opciones, int n) {

int opcionSeleccionada = 1; // Indica la opcionSeleccionada

int tecla;

bool repite = true; // controla el bucle para regresar a la rutina que lo llamo, al presionar ENTER

do {

system("cls");

gotoxy(5, 3 + opcionSeleccionada); cout << "-->" << endl;

// Imprime el título del menú

gotoxy(15, 2); cout << titulo;

// Imprime las opciones del menú

for (int i = 0; i < n; ++i) {

gotoxy(10, 4 + i); cout << opciones[i];

}

// Solo permite que se ingrese ARRIBA, ABAJO o ENTER

do {

tecla = \_getch();

// gotoxy(15, 15); cout << "tecla presionada: " << (char)tecla << " = " << tecla;

} while (tecla != ARRIBA && tecla != ABAJO && tecla != ENTER);

switch (tecla) {

case ARRIBA: // En caso que se presione ARRIBA

opcionSeleccionada--;

if (opcionSeleccionada < 1) {

opcionSeleccionada = n;

}

break;

case ABAJO:

opcionSeleccionada++;

if (opcionSeleccionada > n) {

opcionSeleccionada = 1;

}

break;

case ENTER:

repite = false;

break;

}

} while (repite);

return opcionSeleccionada;

}

void Menu::gotoxy(USHORT x, USHORT y) {

COORD cp = { x,y };

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cp);

}

#endif // MENU\_H\_INCLUDED

#include "Nodo.h"

Cubo Nodo::getCubo(void)

{

return cubo;

}

void Nodo::setCubo(Cubo newCubo)

{

cubo = newCubo;

}

Nodo\* Nodo::getSiguiente(void)

{

return siguiente;

}

void Nodo::setSiguiente(Nodo\* newSiguiente)

{

siguiente = newSiguiente;

}

Nodo\* Nodo::getAnterior(void)

{

return anterior;

}

void Nodo::setAnterior(Nodo\* newAnterior)

{

anterior = newAnterior;

}

#include "OperacionesNodo.h"

#include "Cubo.h"

#include <iostream>

#include <fstream>

using namespace std;

fstream enter;

int valorClave = -1;

int n = 1;

void OperacionesNodo::insertarCabeza(Nodo\*\* referencia, Cubo cubo)

{

if (\*referencia == NULL)

{

class Nodo\* temporal = new Nodo;

temporal->setCubo(cubo);

temporal->setAnterior(temporal);

temporal->setSiguiente(temporal);

\*referencia = temporal;

return;

}

class Nodo\* temporal = new Nodo;

class Nodo\* ultimo = (\*referencia)->getAnterior();

temporal->setCubo(cubo);

temporal->setSiguiente(\*referencia);

temporal->setAnterior(ultimo);

(\*referencia)->setAnterior(temporal);

ultimo->setSiguiente(temporal);

\*referencia = temporal;

}

void OperacionesNodo::insertarCola(Nodo\*\* referencia, Cubo cubo)

{

if (\*referencia == NULL)

{

class Nodo\* temporal = new Nodo;

temporal->setCubo(cubo);

temporal->setAnterior(temporal);

temporal->setSiguiente(temporal);

\*referencia = temporal;

return;

}

Nodo\* ultimo = (\*referencia)->getAnterior();

class Nodo\* temporal = new Nodo;

temporal->setCubo(cubo);

temporal->setSiguiente(\*referencia);

(\*referencia)->setAnterior(temporal);

temporal->setAnterior(ultimo);

ultimo->setSiguiente(temporal);

}

void OperacionesNodo::imprimir(Nodo\* referencia)

{

class Nodo\* temporal = referencia;

Cubo cubo;

int contador = 0;

while (temporal->getSiguiente() != referencia)

{

cubo = temporal->getCubo();

cout << cubo.getValor() << " ";

contador++;

if (contador > 9) {

cout << endl;

contador = 0;

}

temporal = temporal->getSiguiente();

}

cout << endl;

}

void OperacionesNodo::imprimirTxt(Nodo\* referencia)

{

class Nodo\* temporal = referencia;

Cubo cubo;

int contador = 0;

if (n == 1) {

enter.open("Soluciones/solucion.txt", fstream::out);

enter << "Tablero: " << endl << endl;

}

enter << "Intento Numero: " << n << endl;

while (temporal->getSiguiente() != referencia)

{

cubo = temporal->getCubo();

enter << cubo.getValor() << " ";

contador++;

if (contador > 9) {

enter << endl;

contador = 0;

}

temporal = temporal->getSiguiente();

}

n++;

enter << endl;

}

void OperacionesNodo::verificacionHorizontal(Nodo\* fila)

{

class Nodo\* tempInicial = fila;

class Nodo\* tempSiguiente = fila->getSiguiente();

class Nodo\* tempSiguientex2 = tempSiguiente->getSiguiente();

Cubo cuboInicial;

Cubo cuboSiguiente;

Cubo cuboSiguientex2;

int auxiliar;

bool bandera;

while (tempInicial->getSiguiente() != fila)

{

bandera = false;

cuboInicial = tempInicial->getCubo();

cuboSiguiente = tempSiguiente->getCubo();

cuboSiguientex2 = tempSiguientex2->getCubo();

auxiliar = cuboInicial.getValor();

if (cuboInicial.getValor() == cuboSiguiente.getValor()&&cuboInicial.getValor()!=0) {

cuboInicial.setValor(valorClave);

cuboSiguiente.setValor(valorClave);

bandera = true;

tempInicial->setCubo(cuboInicial);

tempSiguiente->setCubo(cuboSiguiente);

}

if (auxiliar == cuboSiguientex2.getValor() && bandera&&auxiliar != 0) {

cuboSiguientex2.setValor(valorClave);

tempSiguientex2->setCubo(cuboSiguientex2);

}

tempInicial = tempInicial->getSiguiente();

tempSiguiente = tempSiguiente->getSiguiente();

tempSiguientex2 = tempSiguientex2->getSiguiente();

}

}

void OperacionesNodo::verificacionVertical(Nodo\* fila)

{

class Nodo\* base = fila;

class Nodo\* pivot = fila;

int i = 0;

while (i < 10) {

pivot = pivot->getSiguiente();

i++;

}

Cubo cuboBase;

Cubo cuboPivot;

while (base->getSiguiente() != fila)

{

cuboBase = base->getCubo();

cuboPivot = pivot->getCubo();

if (cuboBase.getValor() == cuboPivot.getValor() && cuboBase.getValor() != 0 && cuboPivot.getValor() != 0) {

cuboBase.setValor(valorClave);

cuboPivot.setValor(valorClave);

base->setCubo(cuboBase);

pivot->setCubo(cuboPivot);

}

base = base->getSiguiente();

pivot = pivot->getSiguiente();

}

cuboBase = base->getCubo();

cuboPivot = pivot->getCubo();

//este if sirve para verifcar solo la ultima posicion de la fila

if (cuboBase.getValor() == cuboPivot.getValor() && cuboBase.getValor() != 0 && cuboPivot.getValor() != 0) {

cuboBase.setValor(valorClave);

cuboPivot.setValor(valorClave);

base->setCubo(cuboBase);

pivot->setCubo(cuboPivot);

}

}

void OperacionesNodo::insertarDatoPorPosicion(Nodo\* fila, int posicionfila,int posicionColumna,int dato)

{

class Nodo\* temporal = fila;

Cubo cubo;

while (temporal->getSiguiente() != temporal)

{

cubo = temporal->getCubo();

if (cubo.getColumna() == posicionColumna&&cubo.getFila() == posicionfila) {

cubo.setValor(dato);

temporal->setCubo(cubo);

break;

}

temporal = temporal->getSiguiente();

}

}

void OperacionesNodo::inicializarFila(Nodo\*\* lista, int casillas)

{

int posicionColumna = 0;

int posicionFila=0;

int i = 0;

Cubo cubo;

while (i<=casillas) {

cubo.setColumna(posicionColumna+1);

posicionFila = asignarFila(i);

cubo.setFila(posicionFila);

cubo.setValor(0);

insertarCola(lista, cubo);

posicionColumna++;

i++;

if (posicionColumna > 9) {

posicionColumna = 0;

}

}

}

bool OperacionesNodo::seguirEnJuego(Nodo\* fila)

{

class Nodo\* temp;

class Nodo\* filaTemp=fila;

Cubo cuboAuxiliar;

int valor;

int auxiliar = 180;

int i = 0;

while (i < auxiliar) {

filaTemp = filaTemp->getSiguiente();

i++;

}

while (filaTemp->getSiguiente()!=fila) {

cuboAuxiliar = filaTemp->getCubo();

valor = cuboAuxiliar.getValor();

if (valor != 0) {

return false;

}

filaTemp = filaTemp->getSiguiente();

}

return true;

}

int OperacionesNodo::asignarFila(int numero) {

if (numero < 10) {

return 1;

}

else if (numero < 20) {

return 2;

}

else if (numero < 30) {

return 3;

}

else if (numero < 40) {

return 4;

}

else if (numero < 50) {

return 5;

}

else if (numero < 60) {

return 6;

}

else if (numero < 70) {

return 7;

}

else if (numero < 80) {

return 8;

}

else if (numero < 90) {

return 9;

}

else if (numero < 100) {

return 10;

}

else if (numero < 110) {

return 11;

}

else if (numero < 120) {

return 12;

}

else if (numero < 130) {

return 13;

}

else if (numero < 140) {

return 14;

}

else if (numero < 150) {

return 15;

}

else if (numero < 160) {

return 16;

}

else if (numero < 170) {

return 17;

}

else if (numero < 180) {

return 18;

}

else if (numero < 190) {

return 19;

}

else {

return 20;

}

}

#include "Tetris.h"

#include <SFML/Graphics.hpp>

#include "SFML/Audio.hpp"

#include <time.h>

#include <iostream>

#include "OperacionesNodo.h"

#include "Cubo.h"

#include "Nodo.h"

using namespace sf;

using namespace std;

const int limiteAlto = 20;

const int limiteAncho = 10;

struct Point

{

int x, y;

};

Point\* encerarEstruct() {

Point\* temporal = new Point[4];

for (int i = 0; i < 4; i++) {

(temporal + i)->x = 0;

(temporal + i)->y = 0;

}

return temporal;

}

Point\* figuraBase = encerarEstruct();

Point\* figuraPivot = encerarEstruct();

sf::SoundBuffer buffer;

int\*\* encerarField() {

int\*\* temporal = new int\* [limiteAlto];

for (int i = 0; i < limiteAlto; ++i) {

\*(temporal + i) = new int[limiteAncho];

}

for (int i = 0; i < limiteAlto; i++) {

for (int j = 0; j < limiteAncho; j++) {

\*(\*(temporal + i) + j) = 0;

}

}

return temporal;

}

int\*\* field = encerarField();

bool check()

{

for (int i = 0; i < 4; i++)

if ((figuraBase + i)->x < 0 || (figuraBase + i)->x >= limiteAncho || (figuraBase + i)->y >= limiteAlto) return 0;

else if (\*(\*(field + (figuraBase + i)->y) + (figuraBase + i)->x)) return 0;

return 1;

};

int\* numeros = new int[20];

int posicionColumna;

int posicionFila;

int auxiliarColorNum;

bool revisar;

bool seguirEnJuego = true;

void Tetris::juego()

{

buffer.loadFromFile("Tetris.wav");

Sound sound;

sound.setBuffer(buffer);

sound.play();

sound.setLoop(10);

sound.setVolume(10);

sound.setAttenuation(10);

int dimFila = 200;

OperacionesNodo operaciones;

Nodo\* listaBase = NULL;

Nodo\* listaSuperior = NULL;

operaciones.inicializarFila(&listaBase, dimFila);

operaciones.inicializarFila(&listaSuperior, dimFila);

srand(time(0));

RenderWindow window(VideoMode(320, 480), "Tetris ");

Texture textCubo, textFondo, textMarco, textGameOver;

textCubo.loadFromFile("images/tiles.png");

textFondo.loadFromFile("images/background.png");

textGameOver.loadFromFile("images/gameOver.png");

Sprite s(textCubo), background(textFondo), lose(textGameOver);

int dx = 0;

int colorNum = 1;

float timer = 0;

float auxDelay = 0;

float delay = 0.2 - auxDelay;

int contadorDificultad = 0;

Clock clock;

int x = 20;

for (int i = 0; i < 20; i++) {

\*(numeros + i) = x;

x--;

}

while (window.isOpen() && seguirEnJuego)

{

revisar = false;

float time = clock.getElapsedTime().asSeconds();

clock.restart();

timer += time;

Event e;

while (window.pollEvent(e))

{

if (e.type == Event::Closed)

window.close();

if (e.type == Event::KeyPressed)

if (e.key.code == Keyboard::Left) dx = -1;

else if (e.key.code == Keyboard::Right) dx = 1;

}

if (Keyboard::isKeyPressed(Keyboard::Down)) delay = 0.05;

//// <- Move -> ///

for (int i = 0; i < 4; i++) {

\*(figuraPivot + i) = \*(figuraBase + i);

(figuraBase + i)->x += dx;

}

if (!check()) {

for (int i = 0; i < 4; i++) {

\*(figuraBase + i) = \*(figuraPivot + i);

}

}

///////Tick//////

if (timer > delay)

{

for (int i = 0; i < 4; i++) {

\*(figuraPivot + i) = \*(figuraBase + i);

(figuraBase + i)->y += 1;

}

if (!check())

{

for (int i = 0; i < 4; i++) {

\*(\*(field + (figuraPivot + i)->y) + (figuraPivot + i)->x) = colorNum;

}

colorNum = 1 + rand() % 7;

//coloNum=1;

for (int i = 0; i < 4; i++)

{

(figuraBase + i)->x = 4;

(figuraBase + i)->y = 1;

}

posicionColumna = (posicionColumna + 18) / 18;

posicionFila = (posicionFila + 18) / 18;

posicionFila = \*(numeros + posicionFila - 1);

cout << "Fila: " << posicionFila << endl;

cout << "Columna: " << posicionColumna << endl;

cout << auxiliarColorNum;

operaciones.insertarDatoPorPosicion(listaBase, posicionFila, posicionColumna, auxiliarColorNum);

revisar = true;

cout << endl;

//aqui reviso las filas

operaciones.verificacionHorizontal(listaBase);

operaciones.verificacionVertical(listaBase);

operaciones.imprimir(listaBase);

operaciones.imprimirTxt(listaBase);

contadorDificultad++;

bool banderaDificultad = false;

if (auxDelay <= 0.15) {

banderaDificultad = true;

}

if (contadorDificultad == 2 && banderaDificultad) {

auxDelay += 0.01;

contadorDificultad = 0;

}

}

timer = 0;

}

///////check lines//////////

Cubo cuboBase;

Cubo cuboPivot;

class Nodo\* temporalBase = listaBase;

class Nodo\* temporalPivot = listaBase;

int i\_posision, j\_posicion;

if (revisar) {

int i = 0;

while (i < 10) {

temporalPivot = temporalPivot->getSiguiente();

i++;

}

while (temporalBase->getSiguiente() != listaBase)

{

cuboPivot = temporalPivot->getCubo();

cuboBase = temporalBase->getCubo();

i\_posision = \*(numeros + cuboBase.getFila());

j\_posicion = cuboBase.getColumna() - 1;

if (cuboBase.getValor() != cuboPivot.getValor() && cuboBase.getValor() == -1) {

int auxFila = i\_posision;

int auxVector;

for (auxFila; auxFila > 0; auxFila--) {

\*(\*(field + auxFila) + j\_posicion) = \*(\*(field + auxFila - 1) + j\_posicion);

auxVector = \*(numeros + auxFila);

operaciones.insertarDatoPorPosicion(listaBase, auxVector, cuboBase.getColumna(), \*(\*(field + auxFila - 1) + j\_posicion));

}

}

if (cuboBase.getValor() == cuboPivot.getValor() && cuboBase.getValor() != 0) {

int auxFila = i\_posision;

int auxVector;

for (auxFila; auxFila > 2; auxFila--) {

\*(\*(field + auxFila) + j\_posicion) = \*(\*(field + auxFila - 2) + j\_posicion);

auxVector = \*(numeros + auxFila);

operaciones.insertarDatoPorPosicion(listaBase, auxVector, cuboBase.getColumna(), \*(\*(field + auxFila - 2) + j\_posicion));

}

}

temporalBase = temporalBase->getSiguiente();

temporalPivot = temporalPivot->getSiguiente();

}

}

dx = 0; delay = 0.2 - auxDelay;

/////////draw//////////

window.clear(Color::White);

window.draw(background);

for (int i = 0; i < limiteAlto; i++)

for (int j = 0; j < limiteAncho; j++)

{

if (\*(\*(field + i) + j) == 0) continue;

s.setTextureRect(IntRect(\*(\*(field + i) + j) \* 18, 0, 18, 18));

s.setPosition(j \* 18, i \* 18);

s.move(28, 31); //offset

window.draw(s);

}

for (int i = 0; i < 4; i++)

{

posicionColumna = (figuraBase + i)->x \* 18;

posicionFila = (figuraBase + i)->y \* 18;

s.setTextureRect(IntRect(colorNum \* 18, 0, 18, 18));

s.setPosition((figuraBase + i)->x \* 18, (figuraBase + i)->y \* 18);

s.move(28, 31); //offset

window.draw(s);

}

auxiliarColorNum = colorNum;

window.display();

seguirEnJuego = operaciones.seguirEnJuego(listaBase);

if (seguirEnJuego == false) {

lose.move(0, 100);

window.draw(lose);

window.display();

sound.pause();

system("pause");

}

}

if (seguirEnJuego == false) {

cout << "Perdiste Perro";

}

}

#include "Tetris.h"

#include "Menu.h"

#include "Cifrador.h"

#include "Imagen.h"

#include "pthread.h"

int main()

{

Menu\* menu = new Menu();

string titulo = "MENU DE OPCIONES";

string\* opciones = new string[8];

\*(opciones + 0) = "Juego";

\*(opciones + 1) = "Imagen";

\*(opciones + 2) = "Encriptar archivo";

\*(opciones + 3) = "Desencriptar archivo";

\*(opciones + 4) = "Generar barcode";

\*(opciones + 5) = "Generar PDF";

\*(opciones + 6) = "Ayuda";

\*(opciones + 7) = "Salir";

int numOpciones = 8;

int opcion;

do {

opcion = menu->menu(titulo, opciones, numOpciones);

system("cls");

switch (opcion) {

case 1:

Tetris tetris;

tetris.juego();

break;

case 2:

hConWnd = GetConsoleWndHandle();

if (hConWnd)

{

BCX\_Bitmap((char\*)"logo.bmp", hConWnd, 123, 1, 1, 0, 0);

//system("pause");

Sleep(10000);

}

system("pause");

break;

case 3:

encriptar(2019, "Soluciones/solucion.txt", "Soluciones/solucionencriptada.txt");

break;

case 4:

desencriptar(2019, "Soluciones/solucionencriptada.txt", "Soluciones/soluciondesencriptada.txt");

break;

case 5:

system("java -jar Generar\_barcode.jar");

break;

case 6:

system("java -jar textToPdf.jar");

break;

case 7:

system("Tetris.chm");

break;

case 8:

cout << "Hasta pronto joven" << endl;

break;

}

} while (opcion != 8);

delete[] opciones;

return 0;

}