CPU

// jewelsLegendC.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "Diamante.h"

#include <string.h>

#include <time.h>

#include <string>

#include <stdlib.h>

#include <fstream>

#include <iostream>

# include <stdio.h>

# include <conio.h>

using namespace std;

int nColores = 8;

enum posicion { todos, arriba, abajo, derecha, izquierda };

FILE \*doc;

FILE \*leer;

void printDiamante(Diamante \*tablero, HANDLE hConsole, int filas, int columnas);

int comprobarIgualesPos(Diamante \*tablero, int posX, int posY, posicion pos, int filas, int columnas);

int comprobarIgualesDer(Diamante \*tablero, int posX, int posY, int filas, int columnas);

int comprobarIgualesAbajo(Diamante \*tablero, int posX, int posY, int filas);

int comprobarIgualesIzquierda(Diamante \*tablero, int posX, int posY, int filas);

int comprobarIgualesArriba(Diamante \*tablero, int posX, int posY, int filas);

Diamante \*explotarIguales(Diamante \*tablero, int posX, int posY, posicion pos, int filas, int columnas);

Diamante \*moverAbajo(Diamante \*tablero, int filas, int columnas);

Diamante \*menuBomba(Diamante \*tablero, int tipBomba, int filas, int columnas);

bool explotan(Diamante \*tablero, int f1, int c1, int f2, int c2, bool ady, int filas, int columnas);

void guardarPartida(Diamante \*tablero, int filas, int columnas, string dificultad);

bool hasMoreMovements(Diamante \*tablero, int filas, int columnas) {

bool expl = false;

int posX = 0;

while (posX < filas && !expl) {

for (int posY = 0; posY < columnas; posY++) {

if (posX + 1 < filas && explotan(tablero, posX, posY, posX + 1, posY,true, filas, columnas)) { // Abajo

expl = true;

}

else if (posY + 1 < columnas && explotan(tablero, posX, posY, posX, posY + 1, true, filas, columnas)) { //Derecha

expl = true;

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, true, filas, columnas) ) { //Izquierda

expl = true;

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX-1, posY, true, filas, columnas)) {//Arriba

expl = true;

}

}

posX++;

}

return expl;

}

Diamante \*comprobarCadena(Diamante \*tablero, int posX, int posY, int filas, int columnas) {

int comFilas = comprobarIgualesDer(tablero, posX, posY, filas,columnas) + comprobarIgualesIzquierda(tablero, posX, posY, filas);

int comColumnas = comprobarIgualesAbajo(tablero, posX, posY,filas) + comprobarIgualesArriba(tablero, posX, posY, filas);

//Comprobamos cual es la cadena mas larga para pasar a explotar la posicion del primer elemento;

if (comFilas > comColumnas) {

int i = posY;

while (tablero[posX\*filas + posY].color == tablero[posX\*filas + (i - 1)].color && i>0) {

i--;

}

tablero = explotarIguales(tablero, posX, i,derecha, filas, columnas);

}

else {

int i = posX;

while (tablero[posX\*filas + posY].color == tablero[(i - 1)\*filas + posY].color && i>0) {

i--;

}

tablero = explotarIguales(tablero, i, posY,abajo, filas, columnas);

}

return tablero;

}

Diamante \*explotarIguales(Diamante \*tablero, int posX, int posY, posicion pos, int filas, int columnas) {

if (comprobarIgualesDer(tablero, posX, posY, filas, columnas) >= 1 && pos == derecha)

tablero = explotarIguales(tablero, posX, posY + 1, derecha, filas, columnas);

if (comprobarIgualesAbajo(tablero, posX, posY, filas) >= 1 && pos == abajo)

tablero = explotarIguales(tablero, posX + 1, posY, abajo, filas, columnas);

tablero[posX\*filas + posY].color = 0;

return tablero;

}

/\*Diamante \*comprobarIguales(Diamante \*diam) {

for (int i = 0; i < filas; i++) { //Recorremos el array en busca de cadenas de tres numeros iguales

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

if (diam[(i\*filas) + j].color != 0) {

int contFilas = comprobarIgualesDer(diam, i, j);

int contColumnas = comprobarIgualesAbajo(diam, i, j);

if (contFilas > 1)

diam = explotarIguales(diam, i, j);

if (contColumnas > 1)

diam = explotarIguales(diam, i, j);

}

}

}

diam = moverAbajo(diam);

return diam;

}\*/

int comprobarIgualesArriba(Diamante \*tablero, int posX, int posY, int filas) {

int cont = 0;

if (posX - 1 >= 0 && tablero[(posX\*filas) + posY].color == tablero[((posX-1) \* filas) + posY].color) {

cont = 1 + comprobarIgualesArriba(tablero, posX-1, posY, filas);

}

return cont;

}

int comprobarIgualesIzquierda(Diamante \*tablero, int posX, int posY, int filas) {

int cont = 0;

if (posY - 1 >= 0 && tablero[(posX\*filas) + posY].color == tablero[(posX \* filas) + posY - 1].color ) {

cont = 1 + comprobarIgualesIzquierda(tablero, posX, posY - 1, filas);

}

return cont;

}

int comprobarIgualesDer(Diamante \*tablero, int posX, int posY, int filas, int columnas) {

int cont = 0;

if (posY + 1 < columnas && tablero[(posX\*filas) + posY].color == tablero[(posX \* filas) + posY + 1].color) {

cont = 1 + comprobarIgualesDer(tablero, posX, posY + 1, filas, columnas);

}

return cont;

}

int comprobarIgualesAbajo(Diamante \*tablero, int posX, int posY, int filas) {

int cont = 0;

//if(posX >= filas - 1 && posY >= columnas - 1)

if (posX + 1 < filas && tablero[(posX\*filas) + posY].color == tablero[((posX + 1) \* filas) + posY].color) {

cont = 1 + comprobarIgualesAbajo(tablero, posX + 1, posY , filas);

}

return cont;

}

int comprobarIgualesPos(Diamante \*tablero, int posX, int posY,posicion pos, int filas, int columnas) {

int cont = 0;

switch (pos)

{

case todos:

cont += comprobarIgualesPos(tablero, posX, posY, derecha, filas,columnas);

cont += comprobarIgualesPos(tablero, posX, posY, izquierda, filas, columnas);

cont += comprobarIgualesPos(tablero, posX, posY, arriba, filas, columnas);

cont += comprobarIgualesPos(tablero, posX, posY, abajo, filas, columnas);

break;

case derecha:

if (posY + 1 < columnas && tablero[(posX\*filas) + posY].color == tablero[(posX \* filas) + posY + 1 ].color) { // comprobamos derecha

cont += 1 + comprobarIgualesPos(tablero, posX, posY + 1,derecha, filas, columnas);

}

break;

case izquierda:

if (posY - 1 >= 0 && tablero[(posX\*filas) + posY].color == tablero[(posX \* filas) + posY - 1].color) { //comprobamos izquierda

cont += 1 + comprobarIgualesPos(tablero, posX, posY - 1,izquierda, filas, columnas);

}

break;

case abajo:

if (posX + 1 < filas && tablero[(posX\*filas) + posY].color == tablero[((posX + 1) \* filas) + posY].color) { //comprobamos abajo

cont += 1 + comprobarIgualesPos(tablero, posX + 1, posY,abajo, filas, columnas);

}

break;

case arriba:

if (posX - 1 >= 0 && tablero[(posX\*filas) + posY].color == tablero[((posX - 1) \* filas) + posY].color) { //comprobamos arriba

cont += 1 + comprobarIgualesPos(tablero, posX - 1, posY,arriba, filas, columnas);

}

break;

default:

break;

}

return cont;

}

void inicicializarArray(Diamante \*tablero, int filas, int columnas, int nColores){

int numAleatorio = 0;

cout << "\n\n";

//srand(time(0));

for (int i = 0; i < filas; i++) { //llenamos el array de nums aleatorios

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

numAleatorio = rand() % nColores + 1;

tablero[(i\*filas) + j] = Diamante(i, j);

tablero[(i\*filas) + j].color = numAleatorio;

}

}

}

Diamante \*rellenarCeros(Diamante \*tablero, int nColores, int filas, int columnas) {

int numAleatorio = 0;

srand(time(NULL));//Cambiamos semillas

for (int i = 0; i < filas; i++) { //recorremos array

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

if (tablero[(i\*filas) + j].color == 0) {

numAleatorio = rand() % nColores + 1;

tablero[(i\*filas) + j] = Diamante(i, j);

tablero[(i\*filas) + j].color = numAleatorio;

}

}

}

return tablero;

}

//funcion para mover los diamantes hacia abajo

//Recorremos la matriz de abajo hacia arriba para no tener que repetir movimientos

Diamante \*moverAbajo(Diamante \*tablero, int filas, int columnas) {

int pos = 0;

int FilaAux = 0;

//Recorremos hacia atras el array

for (int i = columnas - 1; i >= 0; i--) {

for (int j = filas - 1; j >= 0; j--) {

FilaAux = j;

pos = (j\*filas)+i;

if (tablero[(j\*filas) + i].color == 0) {

//Buscamos la primera posicion de la columna con color !=0

while (tablero[((FilaAux)\*filas) + i].color == 0 && FilaAux > 0) {

FilaAux -= 1;

}

//Intercambiamos colores

int colorAux = tablero[(FilaAux\*filas) + i].color;

tablero[(FilaAux\*filas) + i].color = 0;

tablero[pos].color = colorAux;

}

}

}

return tablero;

}

////MUEVE CEROS HACIA IZQUIERDA

int moverIzquierda(Diamante \*tablero, int filas, int columnas) {

int columnaAux = 0;

int columna1 = 0;

for (int i = filas - 1; i >= 0; i--) {

for (int j = columnas - 1; j >= 0; j--) {

columnaAux = j;

columna1 = j;

if (tablero[(i\*filas)+j].color == 0) {

//Buscamos la siguiente columna !=0

while (tablero[(i\*filas) + columnaAux].color == 0 && columnaAux > 0) {

columnaAux -= 1;

}

//Intercambiamos las columnas

while (columna1>=0 && columnaAux>=0) { //control de desbordo (de columna)

tablero[(i\*filas) + columna1].color = tablero[(i\*filas) + columnaAux].color;

tablero[(i\*filas) + columnaAux].color = 0;

columna1 = columna1 - columnas;

columnaAux = columnaAux - columnas;

}

}

}

}

return 0;

}

//Funcion que comprueba si dos diamantes son adyacentes

bool adyacentes(Diamante tablero, int fila1, int columna1, int fila2, int columna2) {

bool ady = false;

if (fila1 == fila2 + 1 || fila1 == fila2 - 1 || (fila1 == fila2 && columna1 != columna2)) {

if (columna1 == columna2 + 1 || columna1 == columna2 - 1 || (columna1 == columna2 && fila1 != fila2)) {

if ((fila1 == fila2 - 1 && columna1 == columna2 - 1) || (fila1 == fila2 + 1 && columna1 == columna2 - 1) || (fila1 == fila2 - 1 && columna1 == columna2 + 1) || (fila1 == fila2 + 1 && columna1 == columna2 + 1)) {

ady = false; //Si el movimiento es en diagonal no sera valido

}

else { ady = true; }

}

}

if (!ady) cout << "\nMOVIMIENTO ERRONEO, debe ser entre diamantes adyacentes y que no esten en diagonal";

return ady;

}

//Funcion que comprueba si los diamantes explotan

bool explotan(Diamante \*tablero, int f1, int c1, int f2, int c2, bool ady, int filas, int columnas) {

bool expl = false;

int contIgualesArriba1=0, contIgualesAbajo1=0, contIgualesDer1=0, contIgualesIzq1=0, contIgualesVert1=0 , contIgualesHoriz1=0;

int contIgualesArriba2=0, contIgualesAbajo2=0, contIgualesDer2=0, contIgualesIzq2=0, contIgualesVert2=0, contIgualesHoriz2=0;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = tablero[(f1\*filas) + c1].color;

int colorAux2 = tablero[(f2\*filas) + c2].color;

tablero[(f1\*filas) + c1].color = colorAux2;

tablero[(f2\*filas) + c2].color = colorAux1;

//Comprobamos todas las direcciones posibles

contIgualesArriba1 = comprobarIgualesPos(tablero, f1, c1, arriba, filas, columnas);

contIgualesAbajo1 = comprobarIgualesPos(tablero, f1, c1, abajo, filas, columnas);

contIgualesVert1 = comprobarIgualesPos(tablero, f1, c1, arriba, filas, columnas) + comprobarIgualesPos(tablero, f1, c1, abajo, filas, columnas);

contIgualesIzq1 = comprobarIgualesPos(tablero, f1, c1, izquierda, filas, columnas);

contIgualesDer1 = comprobarIgualesPos(tablero, f1, c1, derecha, filas, columnas);

contIgualesHoriz1 = comprobarIgualesPos(tablero, f1, c1, izquierda, filas, columnas) + comprobarIgualesPos(tablero, f1, c1, derecha, filas, columnas);

contIgualesArriba2 = comprobarIgualesPos(tablero, f2, c2, arriba, filas, columnas);

contIgualesAbajo2 = comprobarIgualesPos(tablero, f2, c2, abajo, filas, columnas);

contIgualesVert2 = comprobarIgualesPos(tablero, f2, c2, arriba, filas, columnas) + comprobarIgualesPos(tablero, f2, c2, abajo, filas, columnas);

contIgualesIzq2 = comprobarIgualesPos(tablero, f2, c2, izquierda, filas, columnas);

contIgualesDer2 = comprobarIgualesPos(tablero, f2, c2, derecha, filas, columnas);

contIgualesHoriz2 = comprobarIgualesPos(tablero, f2, c2, izquierda, filas, columnas) + comprobarIgualesPos(tablero, f2, c2, derecha, filas, columnas);

//En caso de que alguno de estos parametros sea 2 o más, al menos uno explota

if (contIgualesArriba1>=2 || contIgualesAbajo1>=2 || contIgualesVert1>=2 || contIgualesIzq1>=2 || contIgualesDer1>=2 || contIgualesHoriz1>=2) {

expl = true;

}

else if (contIgualesArriba2>=2 || contIgualesAbajo2>=2 || contIgualesVert2>=2 || contIgualesIzq2>=2 || contIgualesDer2>=2 || contIgualesHoriz2>=2) {

expl = true;

}

// deshacemos los cambios en la matriz

tablero[(f1\*filas) + c1].color = colorAux1;

tablero[(f2\*filas) + c2].color = colorAux2;

return expl;

}

//Funcion que pide movimiento hasta que sea correcto

bool movPosibleManual(Diamante \*tablero, int filas, int columnas, string dificultad) {

int f1 = 0, c1 = 0, f2 = 0, c2 = 0;

bool expl = false, ady = false;

string tipBomb = "";

string guardar = "";

cout << "\n--INTRODUCE TU JUGADA--";

cout << "\nIntroduce la fila del primer diamante: ";

cin >> f1;

if (f1 == 99) {

cout << "\nQuieres guardar la partida? (si/no): ";

cin >> guardar;

if (guardar=="si") guardarPartida(tablero, filas, columnas, dificultad);

exit(0);

}

if (f1 == 91) {

cout << "Activada bomba 1 ";

//tipBomb = (char)f1;

menuBomba(tablero, 91, filas, columnas);

}

else if (f1 == 92) {

cout << "Activada bomba 2 ";

//tipBomb = (char)f1;

menuBomba(tablero, 92, filas, columnas);

}

else if (f1 == 93) {

cout << "Activada bomba 3 ";

//tipBomb = (char)f1;

menuBomba(tablero, 93, filas, columnas);

}

else {

cout << "Introduce la columna del primer diamante: ";

cin >> c1;

cout << "Introduce la fila del segundo diamante: ";

cin >> f2;

cout << "Introduce la columna del segundo diamante: ";

cin >> c2;

ady = adyacentes(\*tablero, f1, c1, f2, c2);

expl = explotan(tablero, f1, c1, f2, c2, ady, filas, columnas);

if (!expl) cout << "\nMOVIMIENTO ERRONEO, los diamantes seleccionados no explotan";

if (ady && expl) {

int colorAux1 = tablero[(f1\*filas) + c1].color;

int colorAux2 = tablero[(f2\*filas) + c2].color;

tablero[(f1\*filas) + c1].color = colorAux2;

tablero[(f2\*filas) + c2].color = colorAux1;

if (comprobarIgualesDer(tablero, f1, c1, filas, columnas) + comprobarIgualesIzquierda(tablero, f1, c1, filas) >= 2 || comprobarIgualesAbajo(tablero, f1, c1, filas) + comprobarIgualesArriba(tablero, f1, c1, filas) >= 2) {

tablero = comprobarCadena(tablero, f1, c1, filas, columnas);

}

else {

tablero = comprobarCadena(tablero, f2, c2, filas, columnas);

}

}

moverAbajo(tablero, filas, columnas);

}

return true;

}

Diamante \*bomba1(Diamante \*tablero, int filas, int columnas) {

int fila = 0;

cout << "Selecciona la fila que quieras explotar: ";

cin >> fila;

for (int i = fila; i < fila + 1; i++) { //recorremos la fila que queremos explotar

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

tablero[(i\*filas) + j].color = 0;

}

}

moverAbajo(tablero, filas, columnas);

return tablero;

}

Diamante \*bomba2(Diamante \*tablero, int filas, int columnas) {

int columna = 0;

cout << "Selecciona la columna que quieras explotar: ";

cin >> columna;

for (int i = 0; i < filas; i++) { //ponemos a 0 las posiciones que son de esa columna

tablero[(i\*filas) + columna].color = 0;

}

moverIzquierda(tablero, filas, columnas);

return tablero;

}

Diamante \*bomba3(Diamante \*tablero, int filas, int columnas) {

int colorAux = 0;

for (int i = 1; i < filas; i++) {

for (int j = 1; j < columnas; j++) {

//Intercambiamos esquinas de cuadrdo

if (j + 1 < columnas && i + 1 < filas) {

colorAux = tablero[(i\*filas) + (j - 1)].color;

tablero[(i\*filas) + (j - 1)].color = tablero[((i + 1)\*filas) + j].color;

tablero[((i + 1)\*filas) + j].color = tablero[(i\*filas) + (j + 1)].color;

tablero[(i\*filas) + (j + 1)].color = tablero[((i - 1)\*filas) + j].color;

tablero[((i - 1)\*filas) + j].color = colorAux;

}

//Intercambiamos flor del cuadrado

if (i + 1 < filas && j + 1 < columnas) {

colorAux = tablero[((i - 1)\*filas) + (j - 1)].color;

tablero[((i - 1)\*filas) + (j - 1)].color = tablero[((i + 1)\*filas) + (j - 1)].color;

tablero[((i + 1)\*filas) + (j - 1)].color = tablero[((i + 1)\*filas) + (j + 1)].color;

tablero[((i + 1)\*filas) + (j + 1)].color = tablero[((i - 1)\*filas) + (j + 1)].color;

tablero[((i - 1)\*filas) + (j + 1)].color = colorAux;

}

j = j +2;

}

i = i +2;

}

return tablero;

}

Diamante \*menuBomba(Diamante \*tablero, int tipBomba, int filas, int columnas) {

int numTipoBomba = 0;

if (tipBomba == 91) numTipoBomba = 1;

else if (tipBomba == 92) numTipoBomba = 2;

else numTipoBomba = 3;

switch (numTipoBomba) {

case 1: bomba1(tablero, filas, columnas);

break;

case 2: bomba2(tablero, filas, columnas);

break;

case 3: bomba3(tablero, filas, columnas);

break;

}

return tablero;

}

Diamante \*autoMov(Diamante \*tablero, int movOptimoFila1, int movOptimoColumna1, int movOptimoFila2, int movOptimoColumna2, int filas, int columnas) {

int colorAux1 = tablero[(movOptimoFila1\*filas) + movOptimoColumna1].color;

int colorAux2 = tablero[(movOptimoFila2\*filas) + movOptimoColumna2].color;

tablero[(movOptimoFila1\*filas) + movOptimoColumna1].color = colorAux2;

tablero[(movOptimoFila2\*filas) + movOptimoColumna2].color = colorAux1;

if (comprobarIgualesDer(tablero, movOptimoFila1, movOptimoColumna1, filas, columnas) + comprobarIgualesIzquierda(tablero, movOptimoFila1, movOptimoColumna1, filas) >= 2

|| comprobarIgualesAbajo(tablero, movOptimoFila1, movOptimoColumna1, filas) + comprobarIgualesArriba(tablero, movOptimoFila1, movOptimoColumna1, filas) >= 2) {

tablero = comprobarCadena(tablero, movOptimoFila1, movOptimoColumna1, filas, columnas);

}

else {

tablero = comprobarCadena(tablero, movOptimoFila2, movOptimoColumna2, filas, columnas);

}

moverAbajo(tablero, filas, columnas);

return tablero;

}

int autoContMov(Diamante \*tablero, int fila1, int columna1, int fila2, int columna2, int filas, int columnas) {

int comFilas1, comColum1,comFilas2,comColum2 = 0;

int colorAux1 = tablero[(fila1\*filas) + columna1].color;

int colorAux2 = tablero[(fila2\*filas) + columna2].color;

tablero[(fila1\*filas) + columna1].color = colorAux2;

tablero[(fila2\*filas) + columna2].color = colorAux1;

comFilas1 = comprobarIgualesDer(tablero, fila1, columna1, filas, columnas) + comprobarIgualesIzquierda(tablero, fila1, columna1, filas);

comColum1 = comprobarIgualesAbajo(tablero, fila1, columna1, filas) + comprobarIgualesArriba(tablero, fila1, columna1, filas);

comFilas2 = comprobarIgualesDer(tablero, fila2, columna2, filas, columnas) + comprobarIgualesIzquierda(tablero, fila2, columna2, filas);

comColum2 = comprobarIgualesAbajo(tablero, fila2, columna2, filas) + comprobarIgualesArriba(tablero, fila2, columna2, filas);

tablero[(fila1\*filas) + columna1].color = colorAux1;

tablero[(fila2\*filas) + columna2].color = colorAux2;

if (comFilas1 > comFilas2 || comColum1 > comColum2) {

if (comFilas1 > comColum1) {

return comFilas1;

}

else return comColum1;

}

else {

if (comFilas2 > comColum2) {

return comFilas2;

}

else return comColum2;

}

}

Diamante \*movManualMode(Diamante \*tablero, string opcionBomba, HANDLE hConsole, int filas, int columnas, string dificultad) {

while (true) {

bool mov = false;

cout << "\n\n";

SetConsoleTextAttribute(hConsole, 15);

cout << "Introduce 99 para salir ";

if (!hasMoreMovements(tablero, filas, columnas)) {

cout << "\nNo hay movimiento disponibles\nQuieres usar una bomba? ";

cin >> opcionBomba;

}

if (opcionBomba == "si") {

cout << "Selecciona el tipo de bomba: ";

int tipoBomba;

cin >> tipoBomba;

menuBomba(tablero, tipoBomba, filas, columnas);

opcionBomba = "no";

}

else if (opcionBomba == "no") {

while (!mov) {

bool canMove = hasMoreMovements(tablero, filas, columnas);

mov = movPosibleManual(tablero,filas,columnas,dificultad);

}

}

else if (opcionBomba == "99") {

exit(0);

}

if (dificultad == "F") {

nColores = 4;

}

else if (dificultad == "M") {

nColores = 6;

}

tablero = rellenarCeros(tablero,nColores, filas, columnas);

cout << "\n\n";

printDiamante(tablero, hConsole, filas, columnas);

//getchar();

}

}

Diamante \*movAutomaticMode(Diamante \*tablero, int filas, int columnas) {

int posX = 0;

int movOptimoFila1, movOptimoColumna1, movOptimoFila2, movOptimoColumna2, contMovOptimo = 0;

int contDiamantesExplot = 0;

HANDLE h;

h = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (!hasMoreMovements(tablero, filas, columnas)) {

SetConsoleTextAttribute(h, 15);

cout << "\nNo hay movimiento disponibles\n ";

cout << "Selecciona el tipo de bomba: ";

int tipoBomba;

cin >> tipoBomba;

menuBomba(tablero, tipoBomba, filas, columnas);

}

else {

while (posX < filas) {

for (int posY = 0; posY < columnas; posY++) {

if (posX + 1 < filas && explotan(tablero, posX, posY, posX + 1, posY, true, filas, columnas)) { // Abajo

contDiamantesExplot = autoContMov(tablero, posX, posY, posX + 1, posY, filas, filas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX + 1;

movOptimoColumna2 = posY;

}

}

else if (posY + 1 < columnas && explotan(tablero, posX, posY, posX, posY + 1, true, filas, columnas)) { //Derecha

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY+1, filas, filas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY+1;

}

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, true, filas, columnas)) { //Izquierda

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY-1, filas, filas);

if (contDiamantesExplot > contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY-1;

}

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, true, filas, columnas)) {//Arriba

contDiamantesExplot = autoContMov(tablero, posX, posY, posX-1, posY, filas, filas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX - 1;

movOptimoColumna2 = posY;

}

}

}

posX++;

}

SetConsoleTextAttribute(h, 15);

cout << "\nLa posicion mas optima es cambiar la fila y columna " << movOptimoFila1 << " " << movOptimoColumna1

<< " por " << movOptimoFila2 << " " << movOptimoColumna2 << " Este movimiento explotara : " << contMovOptimo << endl;

getchar();

tablero = autoMov(tablero, movOptimoFila1, movOptimoColumna1, movOptimoFila2, movOptimoColumna2, filas, columnas);

}

return tablero;

}

//Metodo para guardar la partida en txt

void guardarPartida(Diamante \*tablero, int filas, int columnas, string dificultad) {

ofstream doc;

doc.open("guardado.txt");

doc << filas << "\n";

doc << columnas << "\n";

doc << dificultad << "\n";

for (int i = 0; i < (filas\*columnas); i++) {

doc << tablero[i].color;

}

doc.close();

cout << "Partida guardada";

Sleep(1000);

system("exit");

}

//Metodo para cargar la partida guardada en el txt

void cargarPartida() {

HANDLE hConsole;

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

const string fichero = "guardado.txt";

ifstream leer;

leer.open(fichero.c\_str());

string dificultad;

Diamante \*tablero;

int i = 0, n = 48, filas = 0, columnas = 0;

char fila[80];

if (!leer.fail()) {

leer.getline(fila, 80, '\n');

while (n > 47 && n < 58) {

n = (int)fila[i];

i++;

if (n > 47 && n < 58) {

filas = filas \* 10 + (n - 48);

}

}

}

n = 48;

i = 0;

if (!leer.fail()) {

leer.getline(fila, 80, '\n');

while (n > 47 && n < 58) {

n = (int)fila[i];

i++;

if (n > 47 && n < 58) {

columnas = columnas \* 10 + (n - 48);

}

}

}

if (!leer.fail()) {

leer.getline(fila,80, '\n');

dificultad = fila[0];

}

int lectura = (filas\*columnas)-1;

tablero = new Diamante[filas\*columnas];

inicicializarArray(tablero, filas, columnas, nColores);

for (int i = 0; i < filas\*columnas; i++) {

leer.getline(fila, 80, '\n');

n = (int)fila[i];

if (n > 47 && n < 58) {

tablero[i].color = (int)fila[i] - 48;

}

}

leer.close();

printDiamante(tablero, hConsole, filas, columnas);

movManualMode(tablero, "no", hConsole, filas, columnas, dificultad);

}

int main() {

string modoJuego = "";

string dificultad = "";

string recuperar = "";

int filas = 0;

int columnas = 0;

cout << "Que modo de juego desea iniciar? Automatico (A), Manual (M)" << endl;

cin >> modoJuego;

if (modoJuego == "m" || modoJuego == "M") {

cout << "Quieres cargar la partida? (si/no) :";

cin >> recuperar;

if (recuperar == "si") { cargarPartida(); }

}

cout << "Introduzca las filas y columnas que desea para la matriz?\nFilas: ";

cin >> filas;

cout << "Columnas: ";

cin >> columnas;

cout << "Que dificultad desea para el juego? Facil(F), Medio(M), Dificil(D)" << endl;

cin >> dificultad;

if (dificultad == "F") {

nColores = 4;

}

else if (dificultad == "M") {

nColores = 6;

} //Se pone por defecto en 8 colores

string opcionBomba = "no";

Diamante \*tablero = new Diamante[filas\*columnas];

HANDLE hConsole;

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

inicicializarArray(tablero,filas,columnas,nColores);

//diam = comprobarIguales(diam);

printDiamante(tablero, hConsole, filas, columnas);

//guardarPartida();

while (true) {

if (modoJuego == "a" || modoJuego == "A" || modoJuego == "automatico") {

movAutomaticMode(tablero,filas,columnas);

}

else {

movManualMode(tablero, opcionBomba, hConsole, filas, columnas, dificultad);

}

int size = filas\*columnas \* sizeof(tablero);

tablero = rellenarCeros(tablero,nColores, filas, columnas);

cout << "\n\n";

printDiamante(tablero, hConsole,filas,columnas);

getchar();

}

free(tablero);

getchar();

return 0;

}

void printDiamante(Diamante \*tablero, HANDLE hConsole, int filas, int columnas) { //modulo para imprimir el tablero

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

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

tablero[i\*filas + j].printDiamante(hConsole);

cout << endl;

}

}

<<<<<<<<<<<<<<<<<DIAMANTE CPP<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#include "stdafx.h"

#include "Diamante.h"

Diamante::Diamante()

{

}

//Constructor 2

Diamante::Diamante(int x, int y) {

X = x; //Posicion x del diamante

Y = y;

xplode = false;

//Posicion y del diamante

}

void Diamante::explotarDiamante() { //Explotar un diamante

color = 0;

}

void Diamante::printDiamante(HANDLE hConsole) {

switch (color) {

case 0: SetConsoleTextAttribute(hConsole, 0);

break;

case 1: SetConsoleTextAttribute(hConsole, 1);

break;

case 2: SetConsoleTextAttribute(hConsole, 2);

break;

case 3: SetConsoleTextAttribute(hConsole, 12);

break;

case 4: SetConsoleTextAttribute(hConsole, 13);

break;

case 5: SetConsoleTextAttribute(hConsole, 14);

break;

case 6: SetConsoleTextAttribute(hConsole, 15);

break;

case 7: SetConsoleTextAttribute(hConsole, 3);

break;

case 8: SetConsoleTextAttribute(hConsole, 10);

break;

case 9: SetConsoleTextAttribute(hConsole, 11);

break;

default:;

}

printf("%d ", color);

}

void Diamante::comprobarColor() { //Imprime el color de un diamante

printf("%d", color);

}

Diamante::~Diamante()

{

}

<<<<<<<<<<<<<<<<<<<<<<<DIAMANTE .H<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#include <Windows.h>

class Diamante

{

public:

int X, Y;

int color;

bool xplode;

Diamante();

Diamante(int x, int y);

void explotarDiamante();

void printDiamante(HANDLE hConsole);

void comprobarColor();

~Diamante();

};

-----------------------------GPU 1 BLOCK----------------------------------

#include <stdio.h>

#include <stdlib.h>

#include <cuda.h>

#include <cuda\_runtime.h>

#include <cuda\_runtime\_api.h>

#include <curand.h>

#include <curand\_kernel.h>

#include "device\_launch\_parameters.h"

#include <assert.h>

#include <cmath>

#include <Windows.h>

#include <string.h>

#include <time.h>

#include <string>

#include <stdlib.h>

#include <fstream>

#include <iostream>

# include <stdio.h>

# include <conio.h>

//Poniendo este DEFINE evitamos un error en el que falta la definici�n de HANDLE\_ERROR

#define HANDLE\_ERROR

int\* generaTablero(int filas, int columnas, int bombas);

void imprimeTablero(int\* tablero, int filas, int columnas);

char pedirModoEjecucion();

int pedirFilasTablero();

int pedirColumnasTablero();

char pedirDificultad();

void prop();

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores);

enum posicion {arriba, abajo, derecha, izquierda };

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad);

FILE \*doc;

FILE \*leer;

//Funci�n que devuelve un error si las dimensiones de la martiz son demasiado grandes para la gr�fica

cudaError\_t comprobarPropiedades(int filas, int columnas) {

cudaDeviceProp prop;

cudaError\_t cudaStatus = cudaSuccess;

int count;

long globalMem;

int sharedMem;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

globalMem = prop.totalGlobalMem;

sharedMem = prop.sharedMemPerBlock;

printf("La matriz solicitada ocupa %lu\n", filas\*columnas \* sizeof(int));

printf("La memoria global es de %lu\n", globalMem);

printf("La memoria compartida es de %d\n", sharedMem);

if ((filas\*columnas \* sizeof(int)) >= globalMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria global de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), globalMem);

goto Error;

}

if ((filas\*columnas \* sizeof(int)) >= sharedMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria compartida de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), sharedMem);

goto Error;

}

}

Error:

return cudaStatus;

}

int comprobarIgualesArriba(int \*tablero, int posX, int posY,int tamColumnas) {

int cont = 0;

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesArriba(tablero, posX - 1, posY, tamColumnas);

}

return cont;

}

int comprobarIgualesIzquierda(int \*tablero, int posX, int posY, int tamColumnas) {

int cont = 0;

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) {

cont = 1 + comprobarIgualesIzquierda(tablero, posX, posY - 1, tamColumnas);

}

return cont;

}

int comprobarIgualesDer(int\*tablero, int posX, int posY,int tamColumnas) {

int cont = 0;

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) {

cont = 1 + comprobarIgualesDer(tablero, posX, posY + 1, tamColumnas);

}

return cont;

}

int comprobarIgualesAbajo(int \*tablero, int posX, int posY, int tamColumnas, int tamFilas) {

int cont = 0;

//if(posX >= filas - 1 && posY >= columnas - 1)

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesAbajo(tablero, posX + 1, posY,tamFilas, tamColumnas);

}

return cont;

}

bool explotan(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameHorizon1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

int sameHorizon2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

// deshacemos los cambios en la matriz

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

return explotan;

}

bool hasMoreMovements(int \*tablero, int filas, int columnas) {

bool expl = false;

int posX = 0;

while (posX < filas && !expl) {

for (int posY = 0; posY < columnas; posY++) {

if (posX + 1 < filas && explotan(tablero, posX, posY, posX + 1, posY, filas, columnas)) { // Abajo

expl = true;

}

else if (posY + 1 < columnas && explotan(tablero, posX, posY, posX, posY + 1, filas, columnas)) { //Derecha

expl = true;

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, filas, columnas)) { //Izquierda

expl = true;

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, filas, columnas)) {//Arriba

expl = true;

}

}

posX++;

}

return expl;

}

\_\_device\_\_ int comprobarIgualesPos(int \*tablero, int posX, int posY, posicion pos, int tamFilas, int tamColumnas) {

int cont = 0;

switch (pos)

{

case derecha:

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) { // comprobamos derecha

cont += 1 + comprobarIgualesPos(tablero, posX, posY + 1, derecha, tamFilas, tamColumnas);

}

break;

case izquierda:

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) { //comprobamos izquierda

cont += 1 + comprobarIgualesPos(tablero, posX, posY - 1, izquierda, tamFilas, tamColumnas);

}

break;

case abajo:

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) { //comprobamos abajo

cont += 1 + comprobarIgualesPos(tablero, posX + 1, posY, abajo, tamFilas, tamColumnas);

}

break;

case arriba:

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) { //comprobamos arriba

cont += 1 + comprobarIgualesPos(tablero, posX - 1, posY, arriba, tamFilas, tamColumnas);

}

break;

default:

break;

}

return cont;

}

bool proveBig(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

// Funcion que determina si alineacion 1 es mayoor que 2 o viceversa

\_\_device\_\_ bool comprobarMayor(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

//Funcion que comprueba los estaDentro de la matriz, ya que tratamos con un array

\_\_device\_\_ bool estaDentro(int x, int y, int filas, int columnas) {

return !(x >= filas || x < 0 || y >= columnas || y < 0);

}

//Funcion que devuelve el numero mas alto de explosiones

int autoContMov(int \*dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

int comFilas1, comColum1, comFilas2, comColum2 = 0;

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

comFilas1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1,tamColumnas);

comColum1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

comFilas2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

comColum2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (proveBig(comFilas1, comColum1, comFilas2, comColum2)) {

if (comFilas1 > comColum1) {

return comFilas1;

}

else return comColum1;

}

else {

if (comFilas2 > comColum2) {

return comFilas2;

}

else return comColum2;

}

}

//funcion que comprueba las posiciones iguales del array,

//Funcion que elimina una celda

\_\_device\_\_ void eliminar(int\* dev\_tablero, int fila, int columna, int tamColumnas, int\* dev\_contadorEliminados) {

// Si el valor examinado es distinto de 0, se suma uno al contador de eliminador

if (dev\_tablero[fila \* tamColumnas + columna] != 0)

dev\_contadorEliminados[0] = dev\_contadorEliminados[0] + 1;

//El valor se pone a 0

dev\_tablero[fila \* tamColumnas + columna] = 0;

}

// Metodo que sube los 0 que se encuentran en el tablero hacia la parte mas alta del mismo

\_\_device\_\_ void reestructuracionArribaAbajo(int\* dev\_tablero, int filas, int columnas) {

int celdax = blockIdx.y\* blockDim.y + threadIdx.y; //Indice de la x

int celday = blockIdx.x\* blockDim.x + threadIdx.x; //Indice de la y

int nombre = celdax \* columnas + celday; //Valor del elemento en el array

int size = (filas\*columnas); //Tamaño de la matriz

int actual = nombre;

int count = 0;

int comprobador = 0;

// Se comprueba que esa celda no es 0 para compararla con los elementos que tiene por debajo

if (dev\_tablero[actual] != 0) {

actual += columnas;

// Se comprueban cuantos 0 hay por debajo de la celda. Este numero se guardara en la variable count

while (actual < size) {

if (dev\_tablero[actual] == 0) {

count++;

}

actual += columnas;

}

//Cambio de valor de la celda que se comprueba con la celda de las posiciones que tiene que descender

if (count > 0) {

dev\_tablero[nombre + (count \* columnas)] = dev\_tablero[nombre];

}

actual = nombre - filas;

//Comprobacion de cuantos 0 por encima tiene la celda que se comprueba

while (actual > 0) {

if (dev\_tablero[actual] == 0) {

comprobador++;

}

actual -= filas;

}

//Poner a 0 el valor de la celda que se cambia si su fila menos el numero de 0 que tiene por encima

// es menor o igual que el numero de ceros que tiene por debajo

if (celdax - comprobador < count) {

dev\_tablero[nombre] = 0;

}

}

}

\_\_device\_\_ void reestructuracionIzquierdaDerecha(int\* dev\_tablero, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if (dev\_tablero[i\*columnas + j] == 0) {

while (j>0) {

dev\_tablero[i\*columnas + j] = dev\_tablero[i\*columnas + (j-1)];

dev\_tablero[i\*columnas + (j-1)] = 0;

j--;

}

}

}

//Elimina una fila completa

\_\_device\_\_ void bomba1(int\* dev\_tablero, int fila, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su fila

if (i == fila) dev\_tablero[i\*columnas + j] = 0;

}

//Elimina un columna completa

\_\_device\_\_ void bomba2(int\* dev\_tablero, int columna, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su columna

if (j == columna) dev\_tablero[i\*columnas + j] = 0;

}

//Mueve la matriz en fomra de cuadrados

\_\_device\_\_ void bomba3(int\* dev\_tablero, int filas , int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

int colorAux = 0;

/\*float fila = i%4;

float columna = j % 4;\*/

//printf("\nEntro");

if ((i == 1 && j == 1) || ((i-1) % 3 == 0 && (j-1) % 3 == 0) || ((i-1) % 3 == 0 && j==1) || (i==1 && (j-1) % 3 == 0)){

if (j + 1 < columnas && i + 1 < filas) {

//Intercambio de las puntas del cuadrado

colorAux = dev\_tablero[(i\*columnas) + (j - 1)];

dev\_tablero[(i\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + j];

dev\_tablero[((i + 1)\*columnas) + j] = dev\_tablero[(i\*columnas) + (j + 1)];

dev\_tablero[(i\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + j];

dev\_tablero[((i - 1)\*columnas) + j] = colorAux;

//Intercambiamos flor del cuadrado

colorAux = dev\_tablero[((i - 1)\*columnas) + (j - 1)];

dev\_tablero[((i - 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j - 1)];

dev\_tablero[((i + 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j + 1)];

dev\_tablero[((i + 1)\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + (j + 1)];

dev\_tablero[((i - 1)\*columnas) + (j + 1)] = colorAux;

}

}

}

//Menu de bombas

\_\_global\_\_ void menuBombas(int \*dev\_tablero, int filas, int columnas, int explota, int bomba) {

switch (bomba) {

case 91: bomba1(dev\_tablero, explota, columnas);

reestructuracionArribaAbajo(dev\_tablero, filas, columnas);

break;

case 92: bomba2(dev\_tablero, explota, columnas);

reestructuracionIzquierdaDerecha(dev\_tablero, filas, columnas);

break;

case 93: bomba3(dev\_tablero, filas, columnas);

break;

}

}

// Funcion que elimina con un unico bloque

\_\_device\_\_ void comprobarCadena(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2,int tamFilas, int tamColumnas, int\* dev\_contadorEliminados) {

//Valores de los indices

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//eliminar(dev\_tablero, i, j, tamFilas, dev\_contadorEliminados);

int sameVertical1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (comprobarMayor(sameVertical1,sameHorizon1,sameVertical2,sameHorizon2)) { //comprobamos cual de las posiciones cambiadas explota mas

//Ahora comprobamos que el hilo sea el de la posicion que queremos

if (i == fila1 && j == columna1) {

if (sameHorizon1 > sameVertical1) {

int jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux + 1)] && jAux + 1 < tamColumnas) { //eliminamos igual por derecha

jAux++;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux - 1)] && jAux - 1 >= 0) {//eliminamos igual por izquierda

jAux--;

eliminar(dev\_tablero, i, jAux , tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

else { //if(mayor1){

if (comprobarMayor(sameVertical1, sameHorizon1, sameVertical2, sameHorizon2)) {

int iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux + 1)\*tamColumnas + j] && iAux + 1 < tamFilas) { //eliminamos igual por arriba

iAux++;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux - 1)\*tamColumnas + j] && iAux - 1 >= 0) {//eliminamos igual por abajo

iAux--;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

}

}

}

else {

if (i == fila2 && j == columna2) {

if (sameHorizon2 > sameVertical2) {

int jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux + 1)] && jAux + 1 < tamColumnas) { //eliminamos igual por derecha

jAux++;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux - 1)] && jAux - 1 >= 0) {//eliminamos igual por izquierda

jAux--;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion

}

else{// if(mayor2) {

if (!comprobarMayor(sameVertical1, sameHorizon1, sameVertical2, sameHorizon2)) {

int iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux + 1) \* tamColumnas + j] && iAux + 1 < tamFilas) { //eliminamos igual por abajo

iAux++;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

iAux = i;

while (dev\_tablero[i \* tamColumnas + j] == dev\_tablero[(iAux - 1) \* tamColumnas + j] && iAux - 1 >= 0) {//eliminamos igual por arriba

iAux--;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

}

}

}

}

\_\_device\_\_ void rellenarMatriz(int\* dev\_tablero, int tamFilas, int tamColumnas, int nColores) {

//Valores de los indices

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

curandState\_t state;

/\* we have to initialize the state \*/

curand\_init(0, /\* the seed controls the sequence of random values that are produced \*/

0, /\* the sequence number is only important with multiple cores \*/

0, /\* the offset is how much extra we advance in the sequence for each call, can be 0 \*/

&state);

/\* curand works like rand - except that it takes a state as a parameter \*/

if (dev\_tablero[i\*tamColumnas + j] == 0) dev\_tablero[i\*tamColumnas + j] = curand(&state) % nColores;

}

\_\_global\_\_ void jugarKernel(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2,int tamFila, int tamColumnas, int\* dev\_contadorEliminados) {

comprobarCadena(dev\_tablero, fila1, columna1, fila2, columna2,tamFila,tamColumnas, dev\_contadorEliminados);

\_\_syncthreads();

reestructuracionArribaAbajo(dev\_tablero, tamFila, tamColumnas);

\_\_syncthreads();

}

\_\_device\_\_ bool adyacentes(int fila1, int columna1, int fila2, int columna2) {

bool ady = false;

if (fila1 == fila2 + 1 || fila1 == fila2 - 1 || (fila1 == fila2 && columna1 != columna2)) {

if (columna1 == columna2 + 1 || columna1 == columna2 - 1 || (columna1 == columna2 && fila1 != fila2)) {

if ((fila1 == fila2 - 1 && columna1 == columna2 - 1) || (fila1 == fila2 + 1 && columna1 == columna2 - 1) || (fila1 == fila2 - 1 && columna1 == columna2 + 1) || (fila1 == fila2 + 1 && columna1 == columna2 + 1)) {

ady = false; //Si el movimiento es en diagonal no sera valido

}

else { ady = true; }

}

}

return ady;

}

\_\_device\_\_ bool explotChange(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2,int tamFilas,int tamColumnas) {

int x = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int y = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if ((x == filas1 && y == columnas1) || (x==fila2 && y == columna2)) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameVertical1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

// deshacemos los cambios en la matriz

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

return explotan;

}

}

\_\_global\_\_ void probeMovPosi(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2,int tamFilas,int tamColumnas,char modoJuego,bool\* dev\_mov, int\* dev\_contadorEliminados) {

int fil = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int col = blockIdx.x \* blockDim.x + threadIdx.x;

bool TrueMov;

bool explot;

if ((col == columnas1 && fil == filas1)) {//comprobamos que el hilo que accede a la funcion sea el que queremos cambiar(se comprueban en las funciones los dos numeros)

TrueMov = adyacentes(filas1, columnas1, fila2, columna2);

explot = explotChange(dev\_tablero, filas1, columnas1, fila2, columna2, tamFilas, tamColumnas);

if (TrueMov && explot) {

//Si ambos son true realizamos el cmbio

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

\*dev\_mov = true;

}

else { printf("MOVIMIENTO ERRONEO, Las posiciones no son adyacentes o no explotan\n"); }

}

}

cudaError\_t jugar(int\* tablero, int tamFilas, int tamColumnas, int\* contadorEliminados, char m, int nColores) {

cudaError\_t cudaStatus;

int fila1 = 0, columna1 = 0, fila2 = 0, columna2 = 0;

bool\* mov = false, \*dev\_mov;

int\* dev\_tablero;

int \*dev\_contadorEliminados;

int bomba = 0;

int explota = 0;

bool hayBomba = false;

char guardar=NULL;

dim3 blocks(1);

dim3 threads(tamFilas, tamColumnas);

//Asignamos objeto a memoria global con cudamalloc

cudaStatus = cudaMalloc((void\*\*)&dev\_tablero, tamFilas\*tamColumnas \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_contadorEliminados, sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_mov, sizeof(bool));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

//Pasamos parametros a la parte del device

cudaStatus = cudaMemcpy(dev\_tablero, tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_contadorEliminados, contadorEliminados, sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_contadorEliminados failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_mov, &mov, sizeof(bool), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_mov failed!");

goto Error;

}

bool activada = false;

while (!hasMoreMovements(tablero, tamFilas, tamColumnas) && !activada) {

printf("NO HAY MOVIMIENTO\n\nIntroduce el numero de la bomba: ");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 93) {

explota = 0;

activada = true;

}

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}

//Modo manual

if (m == 'm' && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

printf("-Para usar una bomba introduce 90 ");

printf("\n-Para terminar el juego 99 ");

printf("\nIntroduzca la fila del primer diamante: ");

scanf\_s("%d", &fila1);

if (fila1 == 99) {

printf("\nQuieres guardar la partida? (s/n): ");

fflush(stdin);

scanf\_s("%c", &guardar);

getchar();

if (guardar == 's') {

guardarPartida(tablero, tamFilas, tamColumnas, nColores);

printf("\nPartida guardada correctamente");

}

printf("\n\n-JUEGO TERMINADO-");

Sleep(1000);

exit(0);

}

if (fila1 == 90) {

printf("\nIntroduce el numero de la bomba:");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 93) explota = 0;

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}else {

printf("Introduzca la columna del primer diamante: ");

scanf\_s("%d", &columna1);

printf("Introduzca la fila del segundo diamante: ");

scanf\_s("%d", &fila2);

printf("Introduzca la columna del segundo diamante: ");

scanf\_s("%d", &columna2);

//bool canMove = hasMoreMovements(tablero);

}

}

else if(hasMoreMovements(tablero, tamFilas, tamColumnas)){

int posX = 0;

int movOptimoFila1, movOptimoColumna1, movOptimoFila2, movOptimoColumna2, contMovOptimo = 0;

int contDiamantesExplot = 0;

while (posX < tamFilas) {

for (int posY = 0; posY < tamColumnas; posY++) {

if (posX + 1 < tamFilas && explotan(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas)) { // Abajo

contDiamantesExplot = autoContMov(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX + 1;

movOptimoColumna2 = posY;

}

}

else if (posY + 1 < tamColumnas && explotan(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas)) { //Derecha

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY + 1;

}

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas)) { //Izquierda

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas);

if (contDiamantesExplot > contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY - 1;

}

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas)) {//Arriba

contDiamantesExplot = autoContMov(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX - 1;

movOptimoColumna2 = posY;

}

}

}

posX++;

}

//Definimos el movimiento mas optimo para explotar

fila1 = movOptimoFila1; fila2 = movOptimoFila2;

columna1 = movOptimoColumna1; columna2 = movOptimoColumna2;

printf("Movimiento mas optimo: \n Fila: %d Columna: %d \n Fila: %d Columna: %d ", fila1, columna1, fila2, columna2);

}

if (!hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas))

{

probeMovPosi << <blocks, threads >> > (dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, m, dev\_mov,dev\_contadorEliminados);

cudaStatus = cudaMemcpy(&mov, dev\_mov, sizeof(bool), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy device to host dev\_mov failed!");

goto Error;

}

}

if (mov && !hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

jugarKernel << <blocks, threads >> >(dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, dev\_contadorEliminados);

}

cudaStatus = cudaGetLastError();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "jugarKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));

goto Error;

}

cudaStatus = cudaMemcpy(tablero, dev\_tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(contadorEliminados, dev\_contadorEliminados, sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy contadorEliminados failed!");

goto Error;

}

Error:

cudaFree(dev\_tablero);

cudaFree(dev\_contadorEliminados);

cudaFree(dev\_mov);

return cudaStatus;

}

//Metodo para guardar la partida en txt

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad) {

doc = fopen("guardado.txt", "w");

fprintf(doc, "%i \n", filas);

fprintf(doc, "%i \n", columnas);

fprintf(doc, "%i \n", dificultad);

for (int i = 0; i < (filas\*columnas); i++) {

fprintf(doc, "%i ", tablero[i]);

}

fclose(doc);

}

//Funcion para cargar una partida guardada en el txt guardado

void cargarPartida() {

cudaError\_t cudaStatus;

leer = fopen("guardado.txt", "r");

int filas=0;

int columnas=0;

int dificultad=0;

int contadorEliminados = 0;

// leer el variables del txt

fscanf(leer, "%d", &filas);

printf("FILAS: %d",filas);

fscanf(leer, "%d", &columnas);

printf("\nCOLUMNAS: %d", columnas);

fscanf(leer, "%d", &dificultad);

printf("\nCOLORES: %d", dificultad);

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++) {

fscanf(leer, "%d", &tablero[i]);

}

do {

imprimeTablero(tablero, filas, columnas);

cudaStatus = jugar(tablero, filas, columnas, &contadorEliminados, 'm', dificultad);

imprimeTablero(tablero, filas, dificultad);

tablero = rellenarTablero(tablero, filas, columnas, dificultad);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

}

int main() {

//Declaracion de variables para la ejecucion del programa

cudaError\_t cudaStatus;

int tamFilas; //Filas que tendra el tablero del programa

int tamColumnas; //Columnas que tendra el tablero del programa

char modo; //Modo de ejecucion del programa

int contadorEliminados = 0;

char dificultad;

int\* tablero;

int nColores;

char cargar;

printf("\nQuieres cargar una partida? (s/n): ");

scanf("%c",&cargar);

getchar();

if(cargar=='s') cargarPartida();

else {

modo = pedirModoEjecucion();

dificultad = pedirDificultad();

tamFilas = pedirFilasTablero();

tamColumnas = pedirColumnasTablero();

if (dificultad == 'F') {

nColores = 4;

}

else if (dificultad == 'M') {

nColores = 6;

}

else {

nColores = 8;

}

printf("\nLos datos introducidos por el usuario son: -%c %c %d %d\n", modo, dificultad, tamFilas, tamColumnas);

cudaStatus = comprobarPropiedades(tamFilas, tamColumnas);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

tablero = generaTablero(tamFilas, tamColumnas, nColores);

}

do {

imprimeTablero(tablero, tamFilas, tamColumnas);

if (modo == 'm') {

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'm',nColores);

}

else

{

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'a',nColores);

getchar();

}

imprimeTablero(tablero, tamFilas, tamColumnas);

tablero = rellenarTablero(tablero, tamFilas, tamColumnas, nColores);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "Algo ha fallado!");

goto Error;

}

printf(" - - - - - - JUEGO TERMINADO - - - - - - - ");

Error:

getchar();

getchar();

return cudaStatus;

}

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores) {

for (int i = 0; i < tamFilas\*tamColumnas; i++)

{

if(tablero[i] == 0) tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

//Procedimiento que imprime una matriz de filas \* columnas de enteros indicando fila y columna

void imprimeTablero(int\* tablero, int filas, int columnas) {

printf("\n ------TABLERO------\n\n ");

//Imprime el numero de columna

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

if (i > 99)printf("%d ", i);

else if (i > 9) printf("%d ", i);

else printf("%d ", i);

}

printf("\n");

for (int i = 0; i < filas\*columnas; i++) {

//Imprime el numero de fila

if (i % columnas == 0) {

printf("\n");

if ((i / columnas) > 99)printf("%d | ", i / columnas);

else if ((i / columnas) > 9) printf("%d | ", i / columnas);

else printf("%d | ", i / columnas);

}

//Según el valor en la posición i del tablero se imprime de un color u otro

switch (tablero[i]) {

case 0: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0);

break;

case 1: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 1);

break;

case 2: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 2);

break;

case 3: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 12);

break;

case 4: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 13);

break;

case 5: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

break;

case 6: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15);

break;

case 7: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 3);

break;

case 8: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 10);

break;

case 9: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11);

break;

default:;

}

printf("%d", tablero[i]);

//Se imprime de nuevo en blanco

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

printf(" | ");

}

printf("\n\n");

}

/\*

Procedimiento que genera un tablero de size x size relleno con numeros del 1 al 2

y con bombas representadas con el número 3

\*/

int\* generaTablero(int filas, int columnas, int nColores) {

//Reserva de memoria para el numero de columnas

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++)

{

tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

char pedirDificultad() {

char dificultad=' ';

getchar();

while (dificultad != 'F' && dificultad != 'M' && dificultad != 'D') {

printf("Que dificultad desea para el juego? Facil(F), Medio(M), Dificil(D)\n");

fflush(stdin);

scanf("%c", &dificultad);

if (dificultad != 'F' && dificultad != 'M' && dificultad != 'D' ) {

printf("Usted ha introducido una dificutad no existente: -%c.\n", dificultad);

printf("Por favor, introduzca uno de las dificultades que se le presentan a continuacion.\n\n");

}

};

return dificultad;

}

//Metodo que solicita al usuario el modo de ejecucion del programa

char pedirModoEjecucion() {

char modo;

do {

printf("Existen 2 modos de ejecucion para Jewels Legend:\n\n");

printf("- Automatica(a): el programa pulsara aleatoriamente las teclas del tablero\n");

printf("- Manual(m): el programa esperara a que el usuario pulse las teclas del tablero\n");

printf("Introduce el modo de ejecucion del programa: ");

fflush(stdin);

scanf("%c", &modo);

if (modo != 'a' && modo != 'm') {

printf("Usted ha introducido un modo de ejecucion no existente: %c.\n", modo);

printf("Por favor, introduzca uno de los modos que se le presentan a continuacion.\n\n");

}

} while (modo != 'a' && modo != 'm');

return modo;

}

//Metodo que solicita al usuario el numero de filas que tendra el tablero

int pedirFilasTablero() {

int filas;

do {

printf("\nIntroduzca las filas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &filas);

if (filas < 1 || filas > 2147483647) {

printf("Introduzca un numero de filas correcto\n");

}

} while (filas < 1 || filas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return filas;

}

//Metodo que solicita al usuario el numero de columnas que tendra el tablero

int pedirColumnasTablero() {

int columnas;

do {

printf("\nIntroduzca las columnas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &columnas);

if (columnas < 1 || columnas > 2147483647) {

printf("Introduzca un numero de columnas correcto\n");

}

} while (columnas < 1 || columnas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return columnas;

}

void prop() {

cudaDeviceProp prop;

int count;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

printf(" --- General Information for device %d ---\n", i);

printf("Name: %s\n", prop.name);

printf("Compute capability: %d.%d\n", prop.major, prop.minor);

printf("Clock rate: %d\n", prop.clockRate);

printf("Device copy overlap: ");

if (prop.deviceOverlap)

printf("Enabled\n");

else

printf("Disabled\n");

printf("Kernel execition timeout : ");

if (prop.kernelExecTimeoutEnabled)

printf("Enabled\n");

else

printf("Disabled\n");

printf(" --- Memory Information for device %d ---\n", i);

printf("Total global mem: %lu\n", prop.totalGlobalMem);

printf("Total constant Mem: %ld\n", prop.totalConstMem);

printf("Max mem pitch: %ld\n", prop.memPitch);

printf("Texture Alignment: %ld\n", prop.textureAlignment);

printf(" --- MP Information for device %d ---\n", i);

printf("Multiprocessor count: %d\n",

prop.multiProcessorCount);

printf("Shared mem per mp: %ld\n", prop.sharedMemPerBlock);

printf("Registers per mp: %d\n", prop.regsPerBlock);

printf("Threads in warp: %d\n", prop.warpSize);

printf("Max threads per block: %d\n",

prop.maxThreadsPerBlock);

printf("Max thread dimensions: (%d, %d, %d)\n",

prop.maxThreadsDim[0], prop.maxThreadsDim[1],

prop.maxThreadsDim[2]);

printf("Max grid dimensions: (%d, %d, %d)\n",

prop.maxGridSize[0], prop.maxGridSize[1],

prop.maxGridSize[2]);

printf("\n");

}

}

---------------------------GPU MULTIPLES BLOCKS---------------------------

#include <stdio.h>

#include <stdlib.h>

#include <cuda.h>

#include <cuda\_runtime.h>

#include <cuda\_runtime\_api.h>

#include <curand.h>

#include <curand\_kernel.h>

#include "device\_launch\_parameters.h"

#include <assert.h>

#include <cmath>

#include <Windows.h>

//Poniendo este DEFINE evitamos un error en el que falta la definici�n de HANDLE\_ERROR

#define HANDLE\_ERROR

#define TESELA 32 // Valor hallado por las propiedades de la tarjera, en el que el numero maximo de hilos por bloque son 1024 que sqrt(1024) son 32

enum posicion {arriba, abajo, derecha, izquierda };

int\* generaTablero(int filas, int columnas, int bombas);

void imprimeTablero(int\* tablero, int filas, int columnas);

char pedirModoEjecucion();

int pedirFilasTablero();

int pedirColumnasTablero();

char pedirDificultad();

void prop();

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores);

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad);

FILE \*doc;

FILE \*leer;

//Funci�n que devuelve un error si las dimensiones de la martiz son demasiado grandes para la gr�fica

cudaError\_t comprobarPropiedades(int filas, int columnas) {

cudaDeviceProp prop;

cudaError\_t cudaStatus = cudaSuccess;

int count;

long globalMem;

int sharedMem;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

globalMem = prop.totalGlobalMem;

sharedMem = prop.sharedMemPerBlock;

printf("La matriz solicitada ocupa %lu\n", filas\*columnas \* sizeof(int));

printf("La memoria global es de %lu\n", globalMem);

printf("La memoria compartida es de %d\n", sharedMem);

if ((filas\*columnas \* sizeof(int)) >= globalMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria global de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), globalMem);

goto Error;

}

if ((filas\*columnas \* sizeof(int)) >= sharedMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria compartida de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), sharedMem);

goto Error;

}

}

Error:

return cudaStatus;

}

int comprobarIgualesArriba(int \*tablero, int posX, int posY,int tamColumnas) {

int cont = 0;

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesArriba(tablero, posX - 1, posY, tamColumnas);

}

return cont;

}

int comprobarIgualesIzquierda(int \*tablero, int posX, int posY, int tamColumnas) {

int cont = 0;

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) {

cont = 1 + comprobarIgualesIzquierda(tablero, posX, posY - 1, tamColumnas);

}

return cont;

}

int comprobarIgualesDer(int\*tablero, int posX, int posY,int tamColumnas) {

int cont = 0;

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) {

cont = 1 + comprobarIgualesDer(tablero, posX, posY + 1, tamColumnas);

}

return cont;

}

int comprobarIgualesAbajo(int \*tablero, int posX, int posY, int tamColumnas, int tamFilas) {

int cont = 0;

//if(posX >= filas - 1 && posY >= columnas - 1)

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesAbajo(tablero, posX + 1, posY,tamFilas, tamColumnas);

}

return cont;

}

bool explotan(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameHorizon1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

int sameHorizon2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

// deshacemos los cambios en la matriz

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

return explotan;

}

bool hasMoreMovements(int \*tablero, int filas, int columnas) {

bool expl = false;

int posX = 0;

while (posX < filas && !expl) {

for (int posY = 0; posY < columnas; posY++) {

if (posX + 1 < filas && explotan(tablero, posX, posY, posX + 1, posY, filas, columnas)) { // Abajo

expl = true;

}

else if (posY + 1 < columnas && explotan(tablero, posX, posY, posX, posY + 1, filas, columnas)) { //Derecha

expl = true;

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, filas, columnas)) { //Izquierda

expl = true;

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, filas, columnas)) {//Arriba

expl = true;

}

}

posX++;

}

return expl;

}

\_\_device\_\_ int comprobarIgualesPos(int \*tablero, int posX, int posY, posicion pos, int tamFilas, int tamColumnas) {

int cont = 0;

switch (pos)

{

case derecha:

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) { // comprobamos derecha

cont += 1 + comprobarIgualesPos(tablero, posX, posY + 1, derecha, tamFilas, tamColumnas);

}

break;

case izquierda:

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) { //comprobamos izquierda

cont += 1 + comprobarIgualesPos(tablero, posX, posY - 1, izquierda, tamFilas, tamColumnas);

}

break;

case abajo:

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) { //comprobamos abajo

cont += 1 + comprobarIgualesPos(tablero, posX + 1, posY, abajo, tamFilas, tamColumnas);

}

break;

case arriba:

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) { //comprobamos arriba

cont += 1 + comprobarIgualesPos(tablero, posX - 1, posY, arriba, tamFilas, tamColumnas);

}

break;

default:

break;

}

return cont;

}

bool proveBig(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

// Funcion que determina si alineacion 1 es mayoor que 2 o viceversa

\_\_device\_\_ bool comprobarMayor(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

//Funcion que comprueba los estaDentro de la matriz, ya que tratamos con un array

\_\_device\_\_ bool estaDentro(int x, int y, int filas, int columnas) {

return !(x >= filas || x < 0 || y >= columnas || y < 0);

}

//Funcion que devuelve el numero mas alto de explosiones

int autoContMov(int \*dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

int comFilas1, comColum1, comFilas2, comColum2 = 0;

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

comFilas1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1,tamColumnas);

comColum1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

comFilas2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

comColum2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (proveBig(comFilas1, comColum1, comFilas2, comColum2)) {

if (comFilas1 > comColum1) {

return comFilas1;

}

else return comColum1;

}

else {

if (comFilas2 > comColum2) {

return comFilas2;

}

else return comColum2;

}

}

//funcion que comprueba las posiciones iguales del array,

//Funcion que elimina una celda

\_\_device\_\_ void eliminar(int\* dev\_tablero, int fila, int columna, int tamColumnas, int\* dev\_contadorEliminados) {

// Si el valor examinado es distinto de 0, se suma uno al contador de eliminador

if (dev\_tablero[fila \* tamColumnas + columna] != 0)

dev\_contadorEliminados[0] = dev\_contadorEliminados[0] + 1;

//El valor se pone a 0

dev\_tablero[fila \* tamColumnas + columna] = 0;

}

// Metodo que sube los 0 que se encuentran en el tablero hacia la parte mas alta del mismo

\_\_device\_\_ void reestructuracionArribaAbajo(int\* dev\_tablero, int filas, int columnas) {

int celdax = blockIdx.y\* blockDim.y + threadIdx.y; //Indice de la x

int celday = blockIdx.x\* blockDim.x + threadIdx.x; //Indice de la y

int nombre = celdax \* columnas + celday; //Valor del elemento en el array

int size = (filas\*columnas); //Tamaño de la matriz

int actual = nombre;

int count = 0;

int comprobador = 0;

// Se comprueba que esa celda no es 0 para compararla con los elementos que tiene por debajo

if (dev\_tablero[actual] != 0 && celday<columnas) {

actual += columnas;

// Se comprueban cuantos 0 hay por debajo de la celda. Este numero se guardara en la variable count

while (actual < size) {

if (dev\_tablero[actual] == 0 && celday<columnas) {

count++;

}

actual += columnas;

}

//Cambio de valor de la celda que se comprueba con la celda de las posiciones que tiene que descender

if (count > 0 && celday<columnas) {

dev\_tablero[nombre + (count \* columnas)] = dev\_tablero[nombre];

}

actual = nombre - filas;

//Comprobacion de cuantos 0 por encima tiene la celda que se comprueba

while (actual > 0 && celday<columnas) {

if (dev\_tablero[actual] == 0) {

comprobador++;

}

actual -= filas;

}

//Poner a 0 el valor de la celda que se cambia si su fila menos el numero de 0 que tiene por encima

// es menor o igual que el numero de ceros que tiene por debajo

if (celdax - comprobador < count && celday<columnas) {

dev\_tablero[nombre] = 0;

}

}

}

\_\_device\_\_ void reestructuracionIzquierdaDerecha(int\* dev\_tablero, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if (dev\_tablero[i\*columnas + j] == 0) {

while (j>0 && j<columnas) {

dev\_tablero[i\*columnas + j] = dev\_tablero[i\*columnas + (j-1)];

dev\_tablero[i\*columnas + (j-1)] = 0;

j--;

}

}

}

//Elimina una fila completa

\_\_device\_\_ void bomba1(int\* dev\_tablero, int explota,int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su fila

if (i == explota && j<columnas)

dev\_tablero[i\*columnas + j] = 0;

}

//Elimina un columna completa

\_\_device\_\_ void bomba2(int\* dev\_tablero, int columna , int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su columna

if (j == columna && i<filas) dev\_tablero[i\*columnas + j] = 0;

}

//Mueve la matriz en fomra de cuadrados

\_\_device\_\_ void bomba3(int\* dev\_tablero, int filas , int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

int colorAux = 0;

if ((i == 1 && j == 1) || ((i-1) % 3 == 0 && (j-1) % 3 == 0) || ((i-1) % 3 == 0 && j==1) || (i==1 && (j-1) % 3 == 0)){

if (j + 1 < columnas && i + 1 < filas) {

//Intercambio de las puntas del cuadrado

colorAux = dev\_tablero[(i\*columnas) + (j - 1)];

dev\_tablero[(i\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + j];

dev\_tablero[((i + 1)\*columnas) + j] = dev\_tablero[(i\*columnas) + (j + 1)];

dev\_tablero[(i\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + j];

dev\_tablero[((i - 1)\*columnas) + j] = colorAux;

//Intercambiamos flor del cuadrado

colorAux = dev\_tablero[((i - 1)\*columnas) + (j - 1)];

dev\_tablero[((i - 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j - 1)];

dev\_tablero[((i + 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j + 1)];

dev\_tablero[((i + 1)\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + (j + 1)];

dev\_tablero[((i - 1)\*columnas) + (j + 1)] = colorAux;

}

}

}

//Menu de bombas

\_\_global\_\_ void menuBombas(int \*dev\_tablero, int filas, int columnas, int explota, int bomba) {

switch (bomba) {

case 91: bomba1(dev\_tablero, explota,filas, columnas);

reestructuracionArribaAbajo(dev\_tablero, filas, columnas);

break;

case 92: bomba2(dev\_tablero, explota, filas, columnas);

reestructuracionIzquierdaDerecha(dev\_tablero, filas, columnas);

break;

case 93: bomba3(dev\_tablero, filas, columnas);

break;

}

}

// Funcion que elimina con un unico bloque

\_\_device\_\_ void comprobarCadena(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2,int tamFilas, int tamColumnas, int\* dev\_contadorEliminados) {

//Valores de los indices

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//eliminar(dev\_tablero, i, j, tamFilas, dev\_contadorEliminados);

int sameVertical1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (comprobarMayor(sameVertical1,sameHorizon1,sameVertical2,sameHorizon2)) { //comprobamos cual de las posiciones cambiadas explota mas

//Ahora comprobamos que el hilo sea el de la posicion que queremos

if (i == fila1 && j == columna1) {

if (sameHorizon1 > sameVertical1) {

int jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux + 1)] && jAux + 1 < tamColumnas) { //eliminamos igual por derecha

jAux++;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux - 1)] && jAux - 1 >= 0) {//eliminamos igual por izquierda

jAux--;

eliminar(dev\_tablero, i, jAux , tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

else { //if(mayor1){

if (comprobarMayor(sameVertical1, sameHorizon1, sameVertical2, sameHorizon2)) {

int iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux + 1)\*tamColumnas + j] && iAux + 1 < tamFilas) { //eliminamos igual por arriba

iAux++;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux - 1)\*tamColumnas + j] && iAux - 1 >= 0) {//eliminamos igual por abajo

iAux--;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

}

}

}

else {

if (i == fila2 && j == columna2) {

if (sameHorizon2 > sameVertical2) {

int jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux + 1)] && jAux + 1 < tamColumnas) { //eliminamos igual por derecha

jAux++;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

jAux = j;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[i\*tamColumnas + (jAux - 1)] && jAux - 1 >= 0) {//eliminamos igual por izquierda

jAux--;

eliminar(dev\_tablero, i, jAux, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion

}

else{// if(mayor2) {

if (!comprobarMayor(sameVertical1, sameHorizon1, sameVertical2, sameHorizon2)) {

int iAux = i;

while (dev\_tablero[i\*tamColumnas + j] == dev\_tablero[(iAux + 1) \* tamColumnas + j] && iAux + 1 < tamFilas) { //eliminamos igual por abajo

iAux++;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

iAux = i;

while (dev\_tablero[i \* tamColumnas + j] == dev\_tablero[(iAux - 1) \* tamColumnas + j] && iAux - 1 >= 0) {//eliminamos igual por arriba

iAux--;

eliminar(dev\_tablero, iAux, j, tamColumnas, dev\_contadorEliminados);

}

eliminar(dev\_tablero, i, j, tamColumnas, dev\_contadorEliminados); //eliminamos la posicion del hilo

}

}

}

}

}

\_\_device\_\_ void rellenarMatriz(int\* dev\_tablero, int tamFilas, int tamColumnas, int nColores) {

//Valores de los indices

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

curandState\_t state;

/\* we have to initialize the state \*/

curand\_init(0, /\* the seed controls the sequence of random values that are produced \*/

0, /\* the sequence number is only important with multiple cores \*/

0, /\* the offset is how much extra we advance in the sequence for each call, can be 0 \*/

&state);

/\* curand works like rand - except that it takes a state as a parameter \*/

if (dev\_tablero[i\*tamColumnas + j] == 0) dev\_tablero[i\*tamColumnas + j] = curand(&state) % nColores;

}

\_\_global\_\_ void jugarKernel(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2,int tamFila, int tamColumnas, int\* dev\_contadorEliminados) {

comprobarCadena(dev\_tablero, fila1, columna1, fila2, columna2,tamFila,tamColumnas, dev\_contadorEliminados);

\_\_syncthreads();

reestructuracionArribaAbajo(dev\_tablero, tamFila, tamColumnas);

\_\_syncthreads();

}

\_\_device\_\_ bool adyacentes(int fila1, int columna1, int fila2, int columna2) {

bool ady = false;

if (fila1 == fila2 + 1 || fila1 == fila2 - 1 || (fila1 == fila2 && columna1 != columna2)) {

if (columna1 == columna2 + 1 || columna1 == columna2 - 1 || (columna1 == columna2 && fila1 != fila2)) {

if ((fila1 == fila2 - 1 && columna1 == columna2 - 1) || (fila1 == fila2 + 1 && columna1 == columna2 - 1) || (fila1 == fila2 - 1 && columna1 == columna2 + 1) || (fila1 == fila2 + 1 && columna1 == columna2 + 1)) {

ady = false; //Si el movimiento es en diagonal no sera valido

}

else { ady = true; }

}

}

return ady;

}

\_\_device\_\_ bool explotChange(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2,int tamFilas,int tamColumnas) {

int x = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int y = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if ((x == filas1 && y == columnas1) || (x==fila2 && y == columna2)) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameVertical1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

// deshacemos los cambios en la matriz

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

return explotan;

}

}

\_\_global\_\_ void probeMovPosi(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2,int tamFilas,int tamColumnas,char modoJuego,bool\* dev\_mov, int\* dev\_contadorEliminados) {

int fil = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int col = blockIdx.x \* blockDim.x + threadIdx.x;

bool TrueMov;

bool explot;

if ((col == columnas1 && fil == filas1)) {//comprobamos que el hilo que accede a la funcion sea el que queremos cambiar(se comprueban en las funciones los dos numeros)

TrueMov = adyacentes(filas1, columnas1, fila2, columna2);

explot = explotChange(dev\_tablero, filas1, columnas1, fila2, columna2, tamFilas, tamColumnas);

if (TrueMov && explot) {

//Si ambos son true realizamos el cmbio

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

\*dev\_mov = true;

}

else { printf("MOVIMIENTO ERRONEO, Las posiciones no son adyacentes o no explotan\n"); }

}

}

cudaError\_t jugar(int\* tablero, int tamFilas, int tamColumnas, int\* contadorEliminados, char m, int nColores) {

cudaError\_t cudaStatus;

int fila1 = 0, columna1 = 0, fila2 = 0, columna2 = 0;

bool\* mov = false, \*dev\_mov;

int\* dev\_tablero;

int \*dev\_contadorEliminados;

int bomba = 0;

int explota = 0;

bool hayBomba = false;

char guardar = NULL;

int bloquex, bloquey;

bloquex = tamFilas / TESELA;

bloquey = tamColumnas / TESELA;

if (tamFilas % TESELA != 0) {

bloquex++;

}

if (tamColumnas % TESELA != 0) {

bloquey++;

}

dim3 blocks(bloquex, bloquey);

dim3 threads(TESELA, TESELA);

//Asignamos objeto a memoria global con cudamalloc

cudaStatus = cudaMalloc((void\*\*)&dev\_tablero, tamFilas\*tamColumnas \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_contadorEliminados, sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_mov, sizeof(bool));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

//Pasamos parametros a la parte del device

cudaStatus = cudaMemcpy(dev\_tablero, tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_contadorEliminados, contadorEliminados, sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_contadorEliminados failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_mov, &mov, sizeof(bool), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_mov failed!");

goto Error;

}

bool activada = false;

while (!hasMoreMovements(tablero, tamFilas, tamColumnas) && !activada) {

printf("NO HAY MOVIMIENTO\n\nIntroduce el numero de la bomba: ");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 93) {

explota = 0;

activada = true;

}

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}

//Modo manual

if (m == 'm' && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

printf("-Para usar una bomba introduce 90 ");

printf("\n-Para terminar el juego 99 ");

printf("\nIntroduzca la fila del primer diamante: ");

scanf\_s("%d", &fila1);

if (fila1 == 99) {

printf("\nQuieres guardar la partida? (s/n): ");

fflush(stdin);

scanf\_s("%c", &guardar);

getchar();

guardarPartida(tablero, tamFilas, tamColumnas, nColores);

printf("\nPartida guardada correctamente");

printf("\n\n-JUEGO TERMINADO-");

Sleep(1000);

exit(0);

}

if (fila1 == 90) {

printf("\nIntroduce el numero de la bomba:");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 93) explota = 0;

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}else {

printf("Introduzca la columna del primer diamante: ");

scanf\_s("%d", &columna1);

printf("Introduzca la fila del segundo diamante: ");

scanf\_s("%d", &fila2);

printf("Introduzca la columna del segundo diamante: ");

scanf\_s("%d", &columna2);

//bool canMove = hasMoreMovements(tablero);

}

}

else if(hasMoreMovements(tablero, tamFilas, tamColumnas)){

int posX = 0;

int movOptimoFila1, movOptimoColumna1, movOptimoFila2, movOptimoColumna2, contMovOptimo = 0;

int contDiamantesExplot = 0;

while (posX < tamFilas) {

for (int posY = 0; posY < tamColumnas; posY++) {

if (posX + 1 < tamFilas && explotan(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas)) { // Abajo

contDiamantesExplot = autoContMov(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX + 1;

movOptimoColumna2 = posY;

}

}

else if (posY + 1 < tamColumnas && explotan(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas)) { //Derecha

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY + 1;

}

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas)) { //Izquierda

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas);

if (contDiamantesExplot > contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY - 1;

}

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas)) {//Arriba

contDiamantesExplot = autoContMov(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX - 1;

movOptimoColumna2 = posY;

}

}

}

posX++;

}

//Definimos el movimiento mas optimo para explotar

fila1 = movOptimoFila1; fila2 = movOptimoFila2;

columna1 = movOptimoColumna1; columna2 = movOptimoColumna2;

printf("Movimiento mas optimo: \n Fila: %d Columna: %d \n Fila: %d Columna: %d ", fila1, columna1, fila2, columna2);

}

if (!hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas))

{

probeMovPosi << <blocks, threads >> > (dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, m, dev\_mov,dev\_contadorEliminados);

cudaStatus = cudaMemcpy(&mov, dev\_mov, sizeof(bool), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy device to host dev\_mov failed!");

goto Error;

}

}

if (mov && !hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

jugarKernel << <blocks, threads >> >(dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, dev\_contadorEliminados);

}

cudaStatus = cudaGetLastError();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "jugarKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));

goto Error;

}

cudaStatus = cudaMemcpy(tablero, dev\_tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(contadorEliminados, dev\_contadorEliminados, sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy contadorEliminados failed!");

goto Error;

}

Error:

cudaFree(dev\_tablero);

cudaFree(dev\_contadorEliminados);

cudaFree(dev\_mov);

return cudaStatus;

}

//Metodo para guardar la partida en txt

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad) {

doc = fopen("guardado.txt", "w");

fprintf(doc, "%i \n", filas);

fprintf(doc, "%i \n", columnas);

fprintf(doc, "%i \n", dificultad);

for (int i = 0; i < (filas\*columnas); i++) {

fprintf(doc, "%i ", tablero[i]);

}

fclose(doc);

}

//Funcion para cargar una partida guardada en el txt guardado

void cargarPartida() {

cudaError\_t cudaStatus;

leer = fopen("guardado.txt", "r");

int filas = 0;

int columnas = 0;

int dificultad = 0;

int contadorEliminados = 0;

// leer el variables del txt

fscanf(leer, "%d", &filas);

printf("FILAS: %d", filas);

fscanf(leer, "%d", &columnas);

printf("\nCOLUMNAS: %d", columnas);

fscanf(leer, "%d", &dificultad);

printf("\nCOLORES: %d", dificultad);

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++) {

fscanf(leer, "%d", &tablero[i]);

}

do {

imprimeTablero(tablero, filas, columnas);

cudaStatus = jugar(tablero, filas, columnas, &contadorEliminados, 'm', dificultad);

imprimeTablero(tablero, filas, dificultad);

tablero = rellenarTablero(tablero, filas, columnas, dificultad);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

}

int main() {

//Declaracion de variables para la ejecucion del programa

cudaError\_t cudaStatus;

int tamFilas; //Filas que tendra el tablero del programa

int tamColumnas; //Columnas que tendra el tablero del programa

char modo; //Modo de ejecucion del programa

int contadorEliminados = 0;

char dificultad;

int\* tablero;

int nColores;

char cargar;

printf("\nQuieres cargar una partida? (s/n): ");

scanf("%c", &cargar);

getchar();

if (cargar == 's') cargarPartida();

else {

modo = pedirModoEjecucion();

dificultad = pedirDificultad();

tamFilas = pedirFilasTablero();

tamColumnas = pedirColumnasTablero();

if (dificultad == 'F') {

nColores = 4;

}

else if (dificultad == 'M') {

nColores = 6;

}

else {

nColores = 8;

}

printf("\nLos datos introducidos por el usuario son: -%c %c %d %d\n", modo, dificultad, tamFilas, tamColumnas);

cudaStatus = comprobarPropiedades(tamFilas, tamColumnas);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

tablero = generaTablero(tamFilas, tamColumnas, nColores);

}

do {

imprimeTablero(tablero, tamFilas, tamColumnas);

if (modo == 'm') {

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'm', nColores);

}

else

{

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'a', nColores);

getchar();

}

imprimeTablero(tablero, tamFilas, tamColumnas);

tablero = rellenarTablero(tablero, tamFilas, tamColumnas, nColores);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "Algo ha fallado!");

goto Error;

}

printf(" - - - - - - JUEGO TERMINADO - - - - - - - ");

Error:

getchar();

getchar();

return cudaStatus;

}

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores) {

for (int i = 0; i < tamFilas\*tamColumnas; i++)

{

if(tablero[i] == 0) tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

//Procedimiento que imprime una matriz de filas \* columnas de enteros indicando fila y columna

void imprimeTablero(int\* tablero, int filas, int columnas) {

printf("\n ------TABLERO------\n\n ");

//Imprime el numero de columna

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

if (i > 99)printf("%d ", i);

else if (i > 9) printf("%d ", i);

else printf("%d ", i);

}

printf("\n");

for (int i = 0; i < filas\*columnas; i++) {

//Imprime el numero de fila

if (i % columnas == 0) {

printf("\n");

if ((i / columnas) > 99)printf("%d | ", i / columnas);

else if ((i / columnas) > 9) printf("%d | ", i / columnas);

else printf("%d | ", i / columnas);

}

//Según el valor en la posición i del tablero se imprime de un color u otro

switch (tablero[i]) {

case 0: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0);

break;

case 1: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 1);

break;

case 2: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 2);

break;

case 3: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 12);

break;

case 4: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 13);

break;

case 5: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

break;

case 6: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15);

break;

case 7: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 3);

break;

case 8: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 10);

break;

case 9: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11);

break;

default:;

}

printf("%d", tablero[i]);

//Se imprime de nuevo en blanco

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

printf(" | ");

}

printf("\n\n");

}

/\*

Procedimiento que genera un tablero de size x size relleno con numeros del 1 al 2

y con bombas representadas con el número 3

\*/

int\* generaTablero(int filas, int columnas, int nColores) {

//Reserva de memoria para el numero de columnas

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++)

{

tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

char pedirDificultad() {

char dificultad=' ';

getchar();

while (dificultad != 'F' && dificultad != 'M' && dificultad != 'D') {

printf("Que dificultad desea para el juego? Facil(F), Medio(M), Dificil(D)\n");

fflush(stdin);

scanf("%c", &dificultad);

if (dificultad != 'F' && dificultad != 'M' && dificultad != 'D' ) {

printf("Usted ha introducido una dificutad no existente: -%c.\n", dificultad);

printf("Por favor, introduzca uno de las dificultades que se le presentan a continuacion.\n\n");

}

};

return dificultad;

}

//Metodo que solicita al usuario el modo de ejecucion del programa

char pedirModoEjecucion() {

char modo;

do {

printf("Existen 2 modos de ejecucion para Jewels Leyend:\n\n");

printf("- Automatica(a): el programa pulsara aleatoriamente las teclas del tablero\n");

printf("- Manual(m): el programa esperara a que el usuario pulse las teclas del tablero\n");

printf("Introduce el modo de ejecucion del programa: ");

fflush(stdin);

scanf("%c", &modo);

if (modo != 'a' && modo != 'm') {

printf("Usted ha introducido un modo de ejecucion no existente: %c.\n", modo);

printf("Por favor, introduzca uno de los modos que se le presentan a continuacion.\n\n");

}

} while (modo != 'a' && modo != 'm');

return modo;

}

//Metodo que solicita al usuario el numero de filas que tendra el tablero

int pedirFilasTablero() {

int filas;

do {

printf("\nIntroduzca las filas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &filas);

if (filas < 1 || filas > 2147483647) {

printf("Introduzca un numero de filas correcto\n");

}

} while (filas < 1 || filas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return filas;

}

//Metodo que solicita al usuario el numero de columnas que tendra el tablero

int pedirColumnasTablero() {

int columnas;

do {

printf("\nIntroduzca las columnas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &columnas);

if (columnas < 1 || columnas > 2147483647) {

printf("Introduzca un numero de columnas correcto\n");

}

} while (columnas < 1 || columnas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return columnas;

}

void prop() {

cudaDeviceProp prop;

int count;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

printf(" --- General Information for device %d ---\n", i);

printf("Name: %s\n", prop.name);

printf("Compute capability: %d.%d\n", prop.major, prop.minor);

printf("Clock rate: %d\n", prop.clockRate);

printf("Device copy overlap: ");

if (prop.deviceOverlap)

printf("Enabled\n");

else

printf("Disabled\n");

printf("Kernel execition timeout : ");

if (prop.kernelExecTimeoutEnabled)

printf("Enabled\n");

else

printf("Disabled\n");

printf(" --- Memory Information for device %d ---\n", i);

printf("Total global mem: %lu\n", prop.totalGlobalMem);

printf("Total constant Mem: %ld\n", prop.totalConstMem);

printf("Max mem pitch: %ld\n", prop.memPitch);

printf("Texture Alignment: %ld\n", prop.textureAlignment);

printf(" --- MP Information for device %d ---\n", i);

printf("Multiprocessor count: %d\n",

prop.multiProcessorCount);

printf("Shared mem per mp: %ld\n", prop.sharedMemPerBlock);

printf("Registers per mp: %d\n", prop.regsPerBlock);

printf("Threads in warp: %d\n", prop.warpSize);

printf("Max threads per block: %d\n",

prop.maxThreadsPerBlock);

printf("Max thread dimensions: (%d, %d, %d)\n",

prop.maxThreadsDim[0], prop.maxThreadsDim[1],

prop.maxThreadsDim[2]);

printf("Max grid dimensions: (%d, %d, %d)\n",

prop.maxGridSize[0], prop.maxGridSize[1],

prop.maxGridSize[2]);

printf("\n");

}

}

------------------------GPU SHARED MEMORY-------------------------

#include <stdio.h>

#include <stdlib.h>

#include <cuda.h>

#include <cuda\_runtime.h>

#include <cuda\_runtime\_api.h>

#include <curand.h>

#include <curand\_kernel.h>

#include "device\_launch\_parameters.h"

#include <assert.h>

#include <cmath>

#include <Windows.h>

#include <string.h>

#include <time.h>

#include <string>

#include <stdlib.h>

#include <fstream>

#include <iostream>

# include <stdio.h>

# include <conio.h>

//Poniendo este DEFINE evitamos un error en el que falta la definici�n de HANDLE\_ERROR

#define HANDLE\_ERROR

#define TESELA 32 // Valor hallado por las propiedades de la tarjera, en el que el numero maximo de hilos por bloque son 1024 que sqrt(1024) son 32

int\* generaTablero(int filas, int columnas, int bombas);

void imprimeTablero(int\* tablero, int filas, int columnas);

char pedirModoEjecucion();

int pedirFilasTablero();

int pedirColumnasTablero();

char pedirDificultad();

void prop();

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores);

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad);

FILE \*doc;

FILE \*leer;

enum posicion { arriba, abajo, derecha, izquierda };

//Funci�n que devuelve un error si las dimensiones de la martiz son demasiado grandes para la gr�fica

cudaError\_t comprobarPropiedades(int filas, int columnas) {

cudaDeviceProp prop;

cudaError\_t cudaStatus = cudaSuccess;

int count;

long globalMem;

int sharedMem;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

globalMem = prop.totalGlobalMem;

sharedMem = prop.sharedMemPerBlock;

printf("La matriz solicitada ocupa %lu\n", filas\*columnas \* sizeof(int));

printf("La memoria global es de %lu\n", globalMem);

printf("La memoria compartida es de %d\n", sharedMem);

if ((filas\*columnas \* sizeof(int)) >= globalMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria global de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), globalMem);

goto Error;

}

if ((filas\*columnas \* sizeof(int)) >= sharedMem) {

fprintf(stderr, "La matriz solicitada ocupa %lu y excede la capacidad de memoria compartida de tu tarjeta gráfica que es %lu \n",

filas\*columnas \* sizeof(int), sharedMem);

goto Error;

}

}

Error:

return cudaStatus;

}

int comprobarIgualesArriba(int \*tablero, int posX, int posY, int tamColumnas) {

int cont = 0;

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesArriba(tablero, posX - 1, posY, tamColumnas);

}

return cont;

}

int comprobarIgualesIzquierda(int \*tablero, int posX, int posY, int tamColumnas) {

int cont = 0;

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) {

cont = 1 + comprobarIgualesIzquierda(tablero, posX, posY - 1, tamColumnas);

}

return cont;

}

int comprobarIgualesDer(int\*tablero, int posX, int posY, int tamColumnas) {

int cont = 0;

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) {

cont = 1 + comprobarIgualesDer(tablero, posX, posY + 1, tamColumnas);

}

return cont;

}

int comprobarIgualesAbajo(int \*tablero, int posX, int posY, int tamColumnas, int tamFilas) {

int cont = 0;

//if(posX >= filas - 1 && posY >= columnas - 1)

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) {

cont = 1 + comprobarIgualesAbajo(tablero, posX + 1, posY, tamFilas, tamColumnas);

}

return cont;

}

bool explotan(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameHorizon1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

int sameHorizon2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

int sameVertical2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

// deshacemos los cambios en la matriz

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

return explotan;

}

bool hasMoreMovements(int \*tablero, int filas, int columnas) {

bool expl = false;

int posX = 0;

while (posX < filas && !expl) {

for (int posY = 0; posY < columnas; posY++) {

if (posX + 1 < filas && explotan(tablero, posX, posY, posX + 1, posY, filas, columnas)) { // Abajo

expl = true;

}

else if (posY + 1 < columnas && explotan(tablero, posX, posY, posX, posY + 1, filas, columnas)) { //Derecha

expl = true;

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, filas, columnas)) { //Izquierda

expl = true;

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, filas, columnas)) {//Arriba

expl = true;

}

}

posX++;

}

return expl;

}

\_\_device\_\_ int comprobarIgualesPos(int \*tablero, int posX, int posY, posicion pos, int tamFilas, int tamColumnas) {

int cont = 0;

switch (pos)

{

case derecha:

if (posY + 1 < tamColumnas && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY + 1]) { // comprobamos derecha

cont += 1 + comprobarIgualesPos(tablero, posX, posY + 1, derecha, tamFilas, tamColumnas);

}

break;

case izquierda:

if (posY - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[(posX \* tamColumnas) + posY - 1]) { //comprobamos izquierda

cont += 1 + comprobarIgualesPos(tablero, posX, posY - 1, izquierda, tamFilas, tamColumnas);

}

break;

case abajo:

if (posX + 1 < tamFilas && tablero[(posX\*tamColumnas) + posY] == tablero[((posX + 1) \* tamColumnas) + posY]) { //comprobamos abajo

cont += 1 + comprobarIgualesPos(tablero, posX + 1, posY, abajo, tamFilas, tamColumnas);

}

break;

case arriba:

if (posX - 1 >= 0 && tablero[(posX\*tamColumnas) + posY] == tablero[((posX - 1) \* tamColumnas) + posY]) { //comprobamos arriba

cont += 1 + comprobarIgualesPos(tablero, posX - 1, posY, arriba, tamFilas, tamColumnas);

}

break;

default:

break;

}

return cont;

}

bool proveBig(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

// Funcion que determina si alineacion 1 es mayoor que 2 o viceversa

\_\_device\_\_ bool comprobarMayor(int sameVertical1, int sameHorizon1, int sameVertical2, int sameHorizon2) {

bool mayor1 = true;

if (((sameVertical1 >= sameVertical2) && (sameVertical1 >= sameHorizon2)) || ((sameHorizon1 >= sameVertical2) && (sameHorizon1 >= sameHorizon2))) mayor1 = true;

else mayor1 = false;

return mayor1;

}

//Funcion que comprueba los estaDentro de la matriz, ya que tratamos con un array

\_\_device\_\_ bool estaDentro(int x, int y, int filas, int columnas) {

return !(x >= filas || x < 0 || y >= columnas || y < 0);

}

//Funcion que devuelve el numero mas alto de explosiones

int autoContMov(int \*dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas) {

int comFilas1, comColum1, comFilas2, comColum2 = 0;

int colorAux1 = dev\_tablero[(fila1\*tamColumnas) + columna1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

comFilas1 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

comColum1 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

comFilas2 = comprobarIgualesDer(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesIzquierda(dev\_tablero, fila1, columna1, tamColumnas);

comColum2 = comprobarIgualesArriba(dev\_tablero, fila1, columna1, tamColumnas) + comprobarIgualesAbajo(dev\_tablero, fila1, columna1, tamColumnas, tamFilas);

dev\_tablero[(fila1\*tamColumnas) + columna1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

if (proveBig(comFilas1, comColum1, comFilas2, comColum2)) {

if (comFilas1 > comColum1) {

return comFilas1;

}

else return comColum1;

}

else {

if (comFilas2 > comColum2) {

return comFilas2;

}

else return comColum2;

}

}

//funcion que comprueba las posiciones iguales del array,

//Funcion que elimina una celda

\_\_device\_\_ void eliminar(int\* dev\_tablero, int fila, int columna, int tamColumnas, int\* dev\_contadorEliminados) {

// Si el valor examinado es distinto de 0, se suma uno al contador de eliminador

if (dev\_tablero[fila \* tamColumnas + columna] != 0)

dev\_contadorEliminados[0] = dev\_contadorEliminados[0] + 1;

//El valor se pone a 0

dev\_tablero[fila \* tamColumnas + columna] = 0;

}

// Metodo que sube los 0 que se encuentran en el tablero hacia la parte mas alta del mismo

\_\_device\_\_ void reestructuracionArribaAbajo(int\* dev\_tablero, int filas, int columnas) {

int celdax = blockIdx.y\* blockDim.y + threadIdx.y; //Indice de la x

int celday = blockIdx.x\* blockDim.x + threadIdx.x; //Indice de la y

int nombre = celdax \* columnas + celday; //Valor del elemento en el array

int size = (filas\*columnas); //Tamaño de la matriz

int actual = nombre;

int count = 0;

int comprobador = 0;

// Se comprueba que esa celda no es 0 para compararla con los elementos que tiene por debajo

if (dev\_tablero[actual] != 0 && celday<columnas) {

actual += columnas;

// Se comprueban cuantos 0 hay por debajo de la celda. Este numero se guardara en la variable count

while (actual < size) {

if (dev\_tablero[actual] == 0 && celday<columnas) {

count++;

}

actual += columnas;

}

//Cambio de valor de la celda que se comprueba con la celda de las posiciones que tiene que descender

if (count > 0 && celday<columnas) {

dev\_tablero[nombre + (count \* columnas)] = dev\_tablero[nombre];

}

actual = nombre - filas;

//Comprobacion de cuantos 0 por encima tiene la celda que se comprueba

while (actual > 0 && celday<columnas) {

if (dev\_tablero[actual] == 0) {

comprobador++;

}

actual -= filas;

}

//Poner a 0 el valor de la celda que se cambia si su fila menos el numero de 0 que tiene por encima

// es menor o igual que el numero de ceros que tiene por debajo

if (celdax - comprobador < count && celday<columnas) {

dev\_tablero[nombre] = 0;

}

}

}

\_\_device\_\_ void reestructuracionIzquierdaDerecha(int\* dev\_tablero, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if (dev\_tablero[i\*columnas + j] == 0) {

while (j>0) {

dev\_tablero[i\*columnas + j] = dev\_tablero[i\*columnas + (j - 1)];

dev\_tablero[i\*columnas + (j - 1)] = 0;

j--;

}

}

}

//Elimina una fila completa

\_\_device\_\_ void bomba1(int\* dev\_tablero, int explota, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su fila

if (i == explota && j<columnas)

dev\_tablero[i\*columnas + j] = 0;

}

//Elimina un columna completa

\_\_device\_\_ void bomba2(int\* dev\_tablero,int explota, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

//Identificamos los diamantes por su columna

if (j == explota&& i<filas) dev\_tablero[i\*columnas + j] = 0;

}

//Mueve la matriz en fomra de cuadrados

\_\_device\_\_ void bomba3(int\* dev\_tablero, int filas, int columnas) {

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

int colorAux = 0;

/\*float fila = i%4;

float columna = j % 4;\*/

//printf("\nEntro");

if ((i == 1 && j == 1) || ((i - 1) % 3 == 0 && (j - 1) % 3 == 0) || ((i - 1) % 3 == 0 && j == 1) || (i == 1 && (j - 1) % 3 == 0)) {

if (j + 1 < columnas && i + 1 < filas) {

//Intercambio de las puntas del cuadrado

colorAux = dev\_tablero[(i\*columnas) + (j - 1)];

dev\_tablero[(i\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + j];

dev\_tablero[((i + 1)\*columnas) + j] = dev\_tablero[(i\*columnas) + (j + 1)];

dev\_tablero[(i\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + j];

dev\_tablero[((i - 1)\*columnas) + j] = colorAux;

//Intercambiamos flor del cuadrado

colorAux = dev\_tablero[((i - 1)\*columnas) + (j - 1)];

dev\_tablero[((i - 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j - 1)];

dev\_tablero[((i + 1)\*columnas) + (j - 1)] = dev\_tablero[((i + 1)\*columnas) + (j + 1)];

dev\_tablero[((i + 1)\*columnas) + (j + 1)] = dev\_tablero[((i - 1)\*columnas) + (j + 1)];

dev\_tablero[((i - 1)\*columnas) + (j + 1)] = colorAux;

}

}

}

//Menu de bombas

\_\_global\_\_ void menuBombas(int \*dev\_tablero, int filas, int columnas, int explota, int bomba) {

switch (bomba) {

case 91: bomba1(dev\_tablero, explota, filas, columnas);

reestructuracionArribaAbajo(dev\_tablero, filas, columnas);

break;

case 92: bomba2(dev\_tablero, explota,filas, columnas);

reestructuracionIzquierdaDerecha(dev\_tablero, filas, columnas);

break;

case 93: bomba3(dev\_tablero, filas, columnas);

break;

}

}

// Funcion que elimina con un unico bloque

\_\_device\_\_ void memoriaCompartida(int\* dev\_tablero, int fila1, int columna1, int tamFilas, int tamColumnas,int valor, int\* dev\_contadorEliminados) {

int sameVertical1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, izquierda, tamFilas, tamColumnas);

\_\_shared\_\_ int matrizCompartida[TESELA][TESELA]; //Variable compartida

int x = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de las x

int y = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de las y

int abajo;

int derecha;

int arriba;

int izquierda;

//Se rellena la memoria compartida con los valores correspondientes

matrizCompartida[threadIdx.y][threadIdx.x] = dev\_tablero[(x\*tamColumnas) + y];

\_\_syncthreads();

if (sameHorizon1 > sameVertical1) {

if (x == fila1 && dev\_tablero[(x \* tamColumnas) + y] == valor && y<tamColumnas) {

matrizCompartida[threadIdx.y][threadIdx.x] = 0;

}

}

else {

if ( y == columna1 && dev\_tablero[(x \* tamColumnas) + y] == valor) {

matrizCompartida[threadIdx.y][threadIdx.x] = 0;

}

}

if (x == fila1 && y == columna1) { //Centro

matrizCompartida[threadIdx.y][threadIdx.x] = 0;

}

\_\_syncthreads();

//Tras ser los valores modificados, se pasan a la memoria global

dev\_tablero[(x \* tamColumnas) + y] = matrizCompartida[threadIdx.y][threadIdx.x];

}

\_\_device\_\_ void rellenarMatriz(int\* dev\_tablero, int tamFilas, int tamColumnas, int nColores) {

//Valores de los indices

int i = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int j = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

curandState\_t state;

/\* we have to initialize the state \*/

curand\_init(0, /\* the seed controls the sequence of random values that are produced \*/

0, /\* the sequence number is only important with multiple cores \*/

0, /\* the offset is how much extra we advance in the sequence for each call, can be 0 \*/

&state);

/\* curand works like rand - except that it takes a state as a parameter \*/

if (dev\_tablero[i\*tamColumnas + j] == 0) dev\_tablero[i\*tamColumnas + j] = curand(&state) % nColores;

}

\_\_global\_\_ void jugarKernel(int\* dev\_tablero, int fila1, int columna1, int fila2, int columna2, int tamFilas, int tamColumnas, int\* dev\_contadorEliminados) {

int sameVertical1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, fila1, columna1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila1, columna1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (comprobarMayor(sameVertical1, sameHorizon1, sameVertical2, sameHorizon2)) { //comprobamos cual de las posiciones cambiadas explota mas

//Ahora comprobamos que el hilo sea el de la posicion que queremos

int valor = dev\_tablero[(fila1 \* tamColumnas) + columna1]; // Valor que tendra la celda eliminada

memoriaCompartida(dev\_tablero, fila1, columna1, tamFilas, tamColumnas, valor, dev\_contadorEliminados);

}else {

int valor = dev\_tablero[(fila2 \* tamColumnas) + columna2]; // Valor que tendra la celda eliminada

memoriaCompartida(dev\_tablero, fila2, columna2, tamFilas, tamColumnas, valor, dev\_contadorEliminados);

}

\_\_syncthreads();

reestructuracionArribaAbajo(dev\_tablero, tamFilas, tamColumnas);

\_\_syncthreads();

}

\_\_device\_\_ bool adyacentes(int fila1, int columna1, int fila2, int columna2) {

bool ady = false;

if (fila1 == fila2 + 1 || fila1 == fila2 - 1 || (fila1 == fila2 && columna1 != columna2)) {

if (columna1 == columna2 + 1 || columna1 == columna2 - 1 || (columna1 == columna2 && fila1 != fila2)) {

if ((fila1 == fila2 - 1 && columna1 == columna2 - 1) || (fila1 == fila2 + 1 && columna1 == columna2 - 1) || (fila1 == fila2 - 1 && columna1 == columna2 + 1) || (fila1 == fila2 + 1 && columna1 == columna2 + 1)) {

ady = false; //Si el movimiento es en diagonal no sera valido

}

else { ady = true; }

}

}

return ady;

}

\_\_device\_\_ bool explotChange(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2, int tamFilas, int tamColumnas) {

int x = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int y = blockIdx.x \* blockDim.x + threadIdx.x; //Indice de la y

if ((x == filas1 && y == columnas1) || (x == fila2 && y == columna2)) {

bool explotan = false;

//HAcemos el intercambio en la matriz para comprobar si se puede explotar

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

int sameVertical1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, arriba, tamFilas, tamColumnas);

int sameHorizon1 = comprobarIgualesPos(dev\_tablero, filas1, columnas1, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, filas1, columnas1, izquierda, tamFilas, tamColumnas);

int sameVertical2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, abajo, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, arriba, tamFilas, tamColumnas);

int sameHorizon2 = comprobarIgualesPos(dev\_tablero, fila2, columna2, derecha, tamFilas, tamColumnas) + comprobarIgualesPos(dev\_tablero, fila2, columna2, izquierda, tamFilas, tamColumnas);

if (sameVertical1 >= 2 || sameHorizon1 >= 2) { //Comprobamos que en cualquiera de las posiciones haya bombas que puedan explotar

explotan = true;

}

else if (sameVertical2 >= 2 || sameHorizon2 >= 2) {

explotan = true;

}

// deshacemos los cambios en la matriz

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux1;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux2;

return explotan;

}

}

\_\_global\_\_ void probeMovPosi(int\* dev\_tablero, int filas1, int columnas1, int fila2, int columna2, int tamFilas, int tamColumnas, char modoJuego, bool\* dev\_mov, int\* dev\_contadorEliminados) {

int fil = blockIdx.y \* blockDim.y + threadIdx.y; //Indice de la x

int col = blockIdx.x \* blockDim.x + threadIdx.x;

bool TrueMov;

bool explot;

if ((col == columnas1 && fil == filas1)) {//comprobamos que el hilo que accede a la funcion sea el que queremos cambiar(se comprueban en las funciones los dos numeros)

TrueMov = adyacentes(filas1, columnas1, fila2, columna2);

explot = explotChange(dev\_tablero, filas1, columnas1, fila2, columna2, tamFilas, tamColumnas);

if (TrueMov && explot) {

//Si ambos son true realizamos el cmbio

int colorAux1 = dev\_tablero[(filas1\*tamColumnas) + columnas1];

int colorAux2 = dev\_tablero[(fila2\*tamColumnas) + columna2];

dev\_tablero[(filas1\*tamColumnas) + columnas1] = colorAux2;

dev\_tablero[(fila2\*tamColumnas) + columna2] = colorAux1;

\*dev\_mov = true;

}

else { printf("MOVIMIENTO ERRONEO, Las posiciones no son adyacentes o no explotan\n"); }

}

}

cudaError\_t jugar(int\* tablero, int tamFilas, int tamColumnas, int\* contadorEliminados, char m, int nColores) {

cudaError\_t cudaStatus;

int fila1 = 0, columna1 = 0, fila2 = 0, columna2 = 0;

bool\* mov = false, \*dev\_mov;

int\* dev\_tablero;

int \*dev\_contadorEliminados;

int bomba = 0;

int explota = 0;

bool hayBomba = false;

char guardar = NULL;

dim3 blocks(1);

dim3 threads(tamFilas, tamColumnas);

//Asignamos objeto a memoria global con cudamalloc

cudaStatus = cudaMalloc((void\*\*)&dev\_tablero, tamFilas\*tamColumnas \* sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_contadorEliminados, sizeof(int));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

cudaStatus = cudaMalloc((void\*\*)&dev\_mov, sizeof(bool));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

//Pasamos parametros a la parte del device

cudaStatus = cudaMemcpy(dev\_tablero, tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_contadorEliminados, contadorEliminados, sizeof(int), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_contadorEliminados failed!");

goto Error;

}

cudaStatus = cudaMemcpy(dev\_mov, &mov, sizeof(bool), cudaMemcpyHostToDevice);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy dev\_mov failed!");

goto Error;

}

bool activada = false;

while (!hasMoreMovements(tablero, tamFilas, tamColumnas) && !activada) {

printf("NO HAY MOVIMIENTO\n\nIntroduce el numero de la bomba: ");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

activada = true;

}

if (bomba == 93) {

explota = 0;

activada = true;

}

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}

//Modo manual

if (m == 'm' && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

printf("-Para usar una bomba introduce 90 ");

printf("\n-Para terminar el juego 99 ");

printf("\nIntroduzca la fila del primer diamante: ");

scanf\_s("%d", &fila1);

if (fila1 == 99) {

printf("\nQuieres guardar la partida? (s/n): ");

fflush(stdin);

scanf\_s("%c", &guardar);

getchar();

//if (guardar == 's') {

guardarPartida(tablero, tamFilas, tamColumnas, nColores);

printf("\nPartida guardada correctamente");

//}

printf("\n\n-JUEGO TERMINADO-");

Sleep(1000);

exit(0);

}

if (fila1 == 90) {

printf("\nIntroduce el numero de la bomba:");

scanf("%d", &bomba);

if (bomba == 91) {

printf("Introduce el numero de la fila que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 92) {

printf("Introduce el numero de la columna que deseas explotar:");

scanf("%d", &explota);

}

if (bomba == 93) explota = 0;

menuBombas << <blocks, threads >> > (dev\_tablero, tamFilas, tamColumnas, explota, bomba);

hayBomba = true;

}

else {

printf("Introduzca la columna del primer diamante: ");

scanf\_s("%d", &columna1);

printf("Introduzca la fila del segundo diamante: ");

scanf\_s("%d", &fila2);

printf("Introduzca la columna del segundo diamante: ");

scanf\_s("%d", &columna2);

//bool canMove = hasMoreMovements(tablero);

}

}

else if (hasMoreMovements(tablero, tamFilas, tamColumnas)) {

int posX = 0;

int movOptimoFila1, movOptimoColumna1, movOptimoFila2, movOptimoColumna2, contMovOptimo = 0;

int contDiamantesExplot = 0;

while (posX < tamFilas) {

for (int posY = 0; posY < tamColumnas; posY++) {

if (posX + 1 < tamFilas && explotan(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas)) { // Abajo

contDiamantesExplot = autoContMov(tablero, posX, posY, posX + 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX + 1;

movOptimoColumna2 = posY;

}

}

else if (posY + 1 < tamColumnas && explotan(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas)) { //Derecha

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY + 1, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY + 1;

}

}

else if (posY - 1 >= 0 && explotan(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas)) { //Izquierda

contDiamantesExplot = autoContMov(tablero, posX, posY, posX, posY - 1, tamFilas, tamColumnas);

if (contDiamantesExplot > contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX;

movOptimoColumna2 = posY - 1;

}

}

else if (posX - 1 >= 0 && explotan(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas)) {//Arriba

contDiamantesExplot = autoContMov(tablero, posX, posY, posX - 1, posY, tamFilas, tamColumnas);

if (contDiamantesExplot >= contMovOptimo) {

contMovOptimo = contDiamantesExplot;

movOptimoFila1 = posX;

movOptimoColumna1 = posY;

movOptimoFila2 = posX - 1;

movOptimoColumna2 = posY;

}

}

}

posX++;

}

//Definimos el movimiento mas optimo para explotar

fila1 = movOptimoFila1; fila2 = movOptimoFila2;

columna1 = movOptimoColumna1; columna2 = movOptimoColumna2;

printf("Movimiento mas optimo: \n Fila: %d Columna: %d \n Fila: %d Columna: %d ", fila1, columna1, fila2, columna2);

}

if (!hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas))

{

probeMovPosi << <blocks, threads >> > (dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, m, dev\_mov, dev\_contadorEliminados);

cudaStatus = cudaMemcpy(&mov, dev\_mov, sizeof(bool), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy device to host dev\_mov failed!");

goto Error;

}

}

if (mov && !hayBomba && hasMoreMovements(tablero, tamFilas, tamColumnas)) {

jugarKernel << <blocks, threads >> >(dev\_tablero, fila1, columna1, fila2, columna2, tamFilas, tamColumnas, dev\_contadorEliminados);

}

cudaStatus = cudaGetLastError();

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "jugarKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));

goto Error;

}

cudaStatus = cudaMemcpy(tablero, dev\_tablero, tamFilas\*tamColumnas \* sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy tablero failed!");

goto Error;

}

cudaStatus = cudaMemcpy(contadorEliminados, dev\_contadorEliminados, sizeof(int), cudaMemcpyDeviceToHost);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMemcpy contadorEliminados failed!");

goto Error;

}

Error:

cudaFree(dev\_tablero);

cudaFree(dev\_contadorEliminados);

cudaFree(dev\_mov);

return cudaStatus;

}

//Metodo para guardar la partida en txt

void guardarPartida(int\* tablero, int filas, int columnas, int dificultad) {

doc = fopen("guardado.txt", "w");

fprintf(doc, "%i \n", filas);

fprintf(doc, "%i \n", columnas);

fprintf(doc, "%i \n", dificultad);

for (int i = 0; i < (filas\*columnas); i++) {

fprintf(doc, "%i ", tablero[i]);

}

fclose(doc);

}

//Funcion para cargar una partida guardada en el txt guardado

void cargarPartida() {

cudaError\_t cudaStatus;

leer = fopen("guardado.txt", "r");

int filas = 0;

int columnas = 0;

int dificultad = 0;

int contadorEliminados = 0;

// leer el variables del txt

fscanf(leer, "%d", &filas);

printf("FILAS: %d", filas);

fscanf(leer, "%d", &columnas);

printf("\nCOLUMNAS: %d", columnas);

fscanf(leer, "%d", &dificultad);

printf("\nCOLORES: %d", dificultad);

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++) {

fscanf(leer, "%d", &tablero[i]);

}

do {

imprimeTablero(tablero, filas, columnas);

cudaStatus = jugar(tablero, filas, columnas, &contadorEliminados, 'm', dificultad);

imprimeTablero(tablero, filas, dificultad);

tablero = rellenarTablero(tablero, filas, columnas, dificultad);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

}

int main() {

//Declaracion de variables para la ejecucion del programa

cudaError\_t cudaStatus;

int tamFilas; //Filas que tendra el tablero del programa

int tamColumnas; //Columnas que tendra el tablero del programa

char modo; //Modo de ejecucion del programa

int contadorEliminados = 0;

char dificultad;

int\* tablero;

int nColores;

char cargar;

printf("\nQuieres cargar una partida? (s/n): ");

scanf("%c", &cargar);

getchar();

if (cargar == 's') cargarPartida();

else {

modo = pedirModoEjecucion();

dificultad = pedirDificultad();

tamFilas = pedirFilasTablero();

tamColumnas = pedirColumnasTablero();

if (dificultad == 'F') {

nColores = 4;

}

else if (dificultad == 'M') {

nColores = 6;

}

else {

nColores = 8;

}

printf("\nLos datos introducidos por el usuario son: -%c %c %d %d\n", modo, dificultad, tamFilas, tamColumnas);

cudaStatus = comprobarPropiedades(tamFilas, tamColumnas);

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "cudaMalloc failed!");

goto Error;

}

tablero = generaTablero(tamFilas, tamColumnas, nColores);

}

do {

imprimeTablero(tablero, tamFilas, tamColumnas);

if (modo == 'm') {

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'm', nColores);

}

else

{

cudaStatus = jugar(tablero, tamFilas, tamColumnas, &contadorEliminados, 'a', nColores);

getchar();

}

imprimeTablero(tablero, tamFilas, tamColumnas);

tablero = rellenarTablero(tablero, tamFilas, tamColumnas, nColores);

printf("Contador = %d\n ", contadorEliminados);

} while ((cudaStatus == 0) && (contadorEliminados < 100));

if (cudaStatus != cudaSuccess) {

fprintf(stderr, "Algo ha fallado!");

goto Error;

}

printf(" - - - - - - JUEGO TERMINADO - - - - - - - ");

Error:

getchar();

getchar();

return cudaStatus;

}

int\* rellenarTablero(int\* tablero, int tamFilas, int tamColumnas, int nColores) {

for (int i = 0; i < tamFilas\*tamColumnas; i++)

{

if (tablero[i] == 0) tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

//Procedimiento que imprime una matriz de filas \* columnas de enteros indicando fila y columna

void imprimeTablero(int\* tablero, int filas, int columnas) {

printf("\n ------TABLERO------\n\n ");

//Imprime el numero de columna

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

if (i > 99)printf("%d ", i);

else if (i > 9) printf("%d ", i);

else printf("%d ", i);

}

printf("\n");

for (int i = 0; i < filas\*columnas; i++) {

//Imprime el numero de fila

if (i % columnas == 0) {

printf("\n");

if ((i / columnas) > 99)printf("%d | ", i / columnas);

else if ((i / columnas) > 9) printf("%d | ", i / columnas);

else printf("%d | ", i / columnas);

}

//Según el valor en la posición i del tablero se imprime de un color u otro

switch (tablero[i]) {

case 0: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 0);

break;

case 1: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 1);

break;

case 2: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 2);

break;

case 3: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 12);

break;

case 4: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 13);

break;

case 5: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 14);

break;

case 6: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 15);

break;

case 7: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 3);

break;

case 8: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 10);

break;

case 9: SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 11);

break;

default:;

}

printf("%d", tablero[i]);

//Se imprime de nuevo en blanco

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), 7);

printf(" | ");

}

printf("\n\n");

}

/\*

Procedimiento que genera un tablero de size x size relleno con numeros del 1 al 2

y con bombas representadas con el número 3

\*/

int\* generaTablero(int filas, int columnas, int nColores) {

//Reserva de memoria para el numero de columnas

int\* tablero = (int\*)malloc(filas\*columnas \* sizeof(int));

for (int i = 0; i < filas\*columnas; i++)

{

tablero[i] = 1 + (rand() % nColores);

}

return tablero;

}

char pedirDificultad() {

char dificultad = ' ';

getchar();

while (dificultad != 'F' && dificultad != 'M' && dificultad != 'D') {

printf("Que dificultad desea para el juego? Facil(F), Medio(M), Dificil(D)\n");

fflush(stdin);

scanf("%c", &dificultad);

if (dificultad != 'F' && dificultad != 'M' && dificultad != 'D') {

printf("Usted ha introducido una dificutad no existente: -%c.\n", dificultad);

printf("Por favor, introduzca uno de las dificultades que se le presentan a continuacion.\n\n");

}

};

return dificultad;

}

//Metodo que solicita al usuario el modo de ejecucion del programa

char pedirModoEjecucion() {

char modo;

do {

printf("Existen 2 modos de ejecucion para Jewels Legend:\n\n");

printf("- Automatica(a): el programa pulsara aleatoriamente las teclas del tablero\n");

printf("- Manual(m): el programa esperara a que el usuario pulse las teclas del tablero\n");

printf("Introduce el modo de ejecucion del programa: ");

fflush(stdin);

scanf("%c", &modo);

if (modo != 'a' && modo != 'm') {

printf("Usted ha introducido un modo de ejecucion no existente: %c.\n", modo);

printf("Por favor, introduzca uno de los modos que se le presentan a continuacion.\n\n");

}

} while (modo != 'a' && modo != 'm');

return modo;

}

//Metodo que solicita al usuario el numero de filas que tendra el tablero

int pedirFilasTablero() {

int filas;

do {

printf("\nIntroduzca las filas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &filas);

if (filas < 1 || filas > 2147483647) {

printf("Introduzca un numero de filas correcto\n");

}

} while (filas < 1 || filas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return filas;

}

//Metodo que solicita al usuario el numero de columnas que tendra el tablero

int pedirColumnasTablero() {

int columnas;

do {

printf("\nIntroduzca las columnas que tendra el tablero: ");

fflush(stdin);

scanf\_s("%d", &columnas);

if (columnas < 1 || columnas > 2147483647) {

printf("Introduzca un numero de columnas correcto\n");

}

} while (columnas < 1 || columnas > 2147483647); //El numero de filas tiene que ser un numero entero positivo

return columnas;

}

void prop() {

cudaDeviceProp prop;

int count;

HANDLE\_ERROR(cudaGetDeviceCount(&count));

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

HANDLE\_ERROR(cudaGetDeviceProperties(&prop, i));

printf(" --- General Information for device %d ---\n", i);

printf("Name: %s\n", prop.name);

printf("Compute capability: %d.%d\n", prop.major, prop.minor);

printf("Clock rate: %d\n", prop.clockRate);

printf("Device copy overlap: ");

if (prop.deviceOverlap)

printf("Enabled\n");

else

printf("Disabled\n");

printf("Kernel execition timeout : ");

if (prop.kernelExecTimeoutEnabled)

printf("Enabled\n");

else

printf("Disabled\n");

printf(" --- Memory Information for device %d ---\n", i);

printf("Total global mem: %lu\n", prop.totalGlobalMem);

printf("Total constant Mem: %ld\n", prop.totalConstMem);

printf("Max mem pitch: %ld\n", prop.memPitch);

printf("Texture Alignment: %ld\n", prop.textureAlignment);

printf(" --- MP Information for device %d ---\n", i);

printf("Multiprocessor count: %d\n",

prop.multiProcessorCount);

printf("Shared mem per mp: %ld\n", prop.sharedMemPerBlock);

printf("Registers per mp: %d\n", prop.regsPerBlock);

printf("Threads in warp: %d\n", prop.warpSize);

printf("Max threads per block: %d\n",

prop.maxThreadsPerBlock);

printf("Max thread dimensions: (%d, %d, %d)\n",

prop.maxThreadsDim[0], prop.maxThreadsDim[1],

prop.maxThreadsDim[2]);

printf("Max grid dimensions: (%d, %d, %d)\n",

prop.maxGridSize[0], prop.maxGridSize[1],

prop.maxGridSize[2]);

printf("\n");

}

}