#include <windows.h>

#include <iostream>

#include <string>

#include "block.cuh"

#include <cuda.h>

#include <cuda\_runtime\_api.h>

#include "device\_launch\_parameters.h"

#include "device\_functions.h"

using namespace std;

const int nColores = 9;

////Metodos GPU

\_\_device\_\_ Block obtenerBloqueSiguiente(Block bloque[], Block bloqueActual, int tx, int ty, int filas, int columnas, int colorInit, bool bloqueElegido);

\_\_device\_\_ int bloquesIgualesAdyacentes(Block bloque[], int filas, int columnas, int x, int y);

int colocarArriba(Block \*bloques, int i, int j, int filas, int columnas);

int colocarDerecha(Block \*bloques, int i, int j, int filas, int columnas);

int moveBlocks(Block \*bloques, int filas, int columnas);

Block \*explotarBomba(Block \*bloques, int posX, int posY, int filas, int columnas);

//Metodo Kernel. Llamado por la CPU y ejecutado por el Dispositivo

\_\_global\_\_ void bloquesKernel(Block \*dev\_bloques, Block \*dev\_bloquesSiguientes, int filas, int columnas, int miX, int miY) {

//Hilos en 2D

int tx = threadIdx.x;

int ty = threadIdx.y;

bool bomba = false;

if(tx == miX && ty == miY){

printf("Se explotará el bloque: %d, %d\n", miX, miY);

}

if((tx == miX && ty == miY) || (tx == miX -1 && ty == miY -1) || (tx == miX && ty == miY -1) || (tx == miX -1 && ty == miY ) || (tx == miX +1 && ty == miY +1) || (tx == miX && ty == miY +1) || (tx == miX +1 && ty == miY ))

{

Block bloqueActual = dev\_bloques[tx\*filas + ty];

Block bloqueSiguiente = obtenerBloqueSiguiente(dev\_bloques, bloqueActual, tx, ty, filas, columnas, dev\_bloques[miX\*filas + miY].color, bomba);

dev\_bloquesSiguientes[tx\*filas + ty] = bloqueSiguiente;

}

else

{

Block bloqueActual = dev\_bloques[tx\*filas + ty];

dev\_bloquesSiguientes[tx\*filas + ty] = bloqueActual;

}

}

\_\_device\_\_ int bloquesIgualesAdyacentes(Block\* bloques, int filas, int columnas, int x, int y) {

int bloquesAdyacentes = 0;

int posX = x, posY = y;

//ARRIBA-IZQU

x = (posX - 1) % filas;

y = (posY - 1) % columnas;

if (posX == 0)

x = filas - 1;

if (posY == 0)

y = columnas - 1;

if (bloques[x\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//ARRIBA

x = (posX - 1);

if (posX == 0)

x = filas - 1;

if (bloques[x\*filas + posY].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//ARRIBA-DERECHA

x = (posX - 1);

y = (posY + 1);

if (posX == 0)

x = filas - 1;

if (posY == (columnas - 1))

y = 0;

if (bloques[x\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//DERECHA % columnas

y = (posY + 1);

if (posY == (columnas - 1))

y = 0;

if (bloques[posX\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//ABAJO-DERECHA

x = (posX + 1);

y = (posY + 1);

if (posX == (filas - 1))

x = 0;

if (posY == (columnas - 1))

y = 0;

if (bloques[x\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//ABAJO % filas

x = (posX + 1);

if (posX == (filas - 1))

x = 0;

if (bloques[x\*filas + posY].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//ABAJO-IZQU

x = (posX + 1);

y = (posY - 1);

if (posX == (filas - 1))

x = 0;

if (posY == 0)

y = columnas - 1;

if (bloques[x\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

//IZQU % columnas

y = (posY - 1);

if (posY == 0)

y = columnas - 1;

if (bloques[posX\*filas + y].color == bloques[posX\*filas + posY].color){

bloquesAdyacentes = bloquesAdyacentes + 1;

}

return bloquesAdyacentes;

}

\_\_device\_\_ Block obtenerBloqueSiguiente(Block\* dev\_bloques, Block bloqueActual, int tx, int ty, int filas, int columnas, int colorInit, bool bomba) {

int bloquesProximos = 0;

if(bomba){

}

bloquesProximos = bloquesIgualesAdyacentes(dev\_bloques, filas, columnas, tx, ty);

if (bloquesProximos >= 2 && bloqueActual.color == colorInit) {

bloqueActual.color = 0;

}

return bloqueActual;

}

//MUEVE CEROS HACIA ARRIBA

int colocarArriba(Block \*bloques, int i, int filas, int columnas){

bool limite=false;

int x=i; //No queremos perder la posicion del primer bloque por eso la guardamos en x

while(limite==false && bloques[x].color==0){ //Es para saber la posicion con el que hay que cambiar

if((x-columnas)>=0) x=x-columnas; //cambiamos las filas hasta encontrar la posicion deseada

else limite=true; //Limite es control de desbordo

}

bloques[i].color=bloques[x].color;

bloques[x].color=0;

return 0;

}

////MUEVE CEROS HACIA DERECHA

int colocarDerecha(Block \*bloques, int i, int filas, int columnas){

bool limite=false;

int x=i; //No queremos perder la posicion del primer bloque por eso la guardamos en x

while(limite==false && bloques[x].color==0){ //Es para saber la posicion con el que hay que cambiar el 0

if(x==((filas\*columnas)-1)) limite=true; //Limite es control de desbordo

else x++; //subimos posicion

}

//Intercambiamos las columnas

while(i>0 && x>0){//limite

bloques[i].color=bloques[x].color;

bloques[x].color=0;

i=i-columnas;

x=x-filas;

}

return 0;

}

//MOVER BLOQUES

int moveBlocks(Block \*bloques, int filas, int columnas){

int j=0;

for(j=(((filas\*columnas)-1));j>=0;j--){

if(bloques[j].color==0){

colocarArriba(bloques,j,filas,columnas);

}

}

for(j=(((filas\*columnas)-1)-columnas);j<((filas\*columnas)-1);j++){

if(bloques[j].color==0){ //comprobamos la fila de mas abajo

colocarDerecha(bloques,j,filas,columnas);

}

}

return 0;

}

Block \*explotarBomba(Block \*bloques, int posX, int posY, int filas, int columnas){

////Explotar centro-arriba

if (posX!=0){

if(bloques[(posX-1)\*columnas+posY].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX-1)\*columnas+posY].color=0;

}

}

if ((posX-2)>=0){

if(bloques[(posX-2)\*columnas+posY].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX-2)\*columnas+posY].color=0;

}

}

//Explotar centro-abajo

if (posX!=filas-1){

if(bloques[(posX+1)\*columnas+posY].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX+1)\*columnas+posY].color=0;

}

}

if (posX+2<=filas-1){

if(bloques[(posX+2)\*columnas+posY].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX+2)\*columnas+posY].color=0;

}

}

//Explotar derecha

if (posY!=columnas-1){

if(bloques[posX\*columnas+(posY+1)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX\*columnas)+(posY+1)].color=0;

}

}

if (posY+2 < columnas-1){

if(bloques[posX\*columnas+(posY+2)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX\*columnas)+(posY+2)].color=0;

}

}

//Explotar izquierda

if (posY!=0){

if(bloques[posX\*columnas+(posY-1)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX\*columnas)+(posY-1)].color=0;

}

}

if (posY-2 > 0){

if(bloques[posX\*columnas+(posY-2)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX\*columnas)+(posY-2)].color=0;

}

}

//Explotar derecha-abajo

if (posX!=(filas-1) && posY!=(columnas-1)){

if(bloques[(posX+1)\*columnas+(posY+1)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX+1)\*columnas+(posY+1)].color=0;

}

}

//Explotar derecha-arriba

if (posX!=0 && posY!=(columnas-1)){

if(bloques[(posX-1)\*columnas+(posY+1)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX-1)\*columnas+(posY+1)].color=0;

}

}

//Explotar izquierda-arriba

if (posX!=0 && posY!=0){

if(bloques[(posX-1)\*columnas+(posY-1)].color==9){

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX-1)\*columnas+(posY-1)].color=0;

}

}

//Explotar izquierda-abajo

if (posX!=(filas-1) && posY!=0){

if(bloques[(posX+1)\*columnas+(posY-1)].color==9){

bloques[(posX+1)\*columnas+(posY-1)].color=0;

}

else{

bloques[(posX+1)\*columnas+(posY-1)].color=0;

}

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

}

return bloques;

}

int main(int argc, char \*argv[]) {

bool modoAutomatico;

HANDLE hConsole;

hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int filas;

int columnas;

//string modo = "-m";

string modo = "-m";

cout<<"Introduce el número de filas y columnas:\_";

int x=0, y=0;

cout<<"\nfilas: ";

cin>>filas;

cout<<"\ncolumnas: ";

cin>>columnas;

cout<< "Filas y columnas cargadas, pulse enter para continuar";

getchar();

/\* Establece el modo de ejecución en funcion de los datos introducidos \*/

if (modo.compare("-a") == 0){

modoAutomatico = true;

}

else if (modo.compare("-m") == 0)

modoAutomatico = false;

else {

cout << "\nEl modo de ejecucion introducido es incorrecto";

exit(1);

}

//Creamos los arrays de bloques

Block\* bloques = new Block[filas \* columnas];

Block\* bloquesSiguientes = new Block[filas \* columnas];

Block \*dev\_bloques;

Block \*dev\_bloquesSiguientes;

int numAleatorio;

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

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

numAleatorio = rand() % nColores;

bloques[(i\*filas) + j] = Block(i, j);

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

}

}

while (true) {

cout << "\n\n\n";

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

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

bloques[i\*columnas + j].printBloque(hConsole);

cout << "\n";

}

cout << "\n\n\n";

SetConsoleTextAttribute(hConsole, 15);

int x=0, y=0;

cout<<"Introduce la posicion del bloque que quieras explotar o 99 para salir";

cout<<"\nfila: ";

cin>>x;

if(x == 99)

exit(0);

cout<<"columna: ";

cin>>y;

if(bloques[x\*columnas + y].color == 9){ //se trata de una bomba, explotamos la cruz de bloques

explotarBomba(bloques, x, y, filas, columnas);

}

else{

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

//Reservamos memoria

(cudaMalloc((void\*\*)&dev\_bloques, size));

(cudaMalloc((void\*\*)&dev\_bloquesSiguientes, size));

//Transferencia de datos de la memoria CPU al Device

(cudaMemcpy(dev\_bloques, bloques, size, cudaMemcpyHostToDevice));

//Dimensiones de cada bloque

dim3 dimBlock(filas, columnas);

//Llamada del método del Kernel. Tendremos 2 bloques/grid y dentro de éste, tantos hilos como posiciones tenga el tablero

bloquesKernel << <1, dimBlock >> >(dev\_bloques, dev\_bloquesSiguientes, filas, columnas, x, y);

//Transferir la variable del dispositivo al host

(cudaMemcpy(bloquesSiguientes, dev\_bloquesSiguientes, size, cudaMemcpyDeviceToHost));

//Liberar la memoria del dipositivo

cudaFree(dev\_bloquesSiguientes);

cudaFree(dev\_bloques);

if (!modoAutomatico)

system("pause");

else

Sleep(2000);

system("pause");

bloques = bloquesSiguientes;

}

moveBlocks(bloques, filas, columnas);

}

free(bloques);

free(bloquesSiguientes);

}

#include <windows.h>

class Block

{

public:

int X,Y;

int color;

bool xplode;

Block(void);

Block(int x, int y);

void explotarBloque();

void printBloque(HANDLE hConsole);

void comprobarColor();

};

#include <iostream>

#include "block.cuh"

#include <windows.h>

using namespace std;

//Constructor 1

Block::Block(void){}

//Constructor 2

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

X = x; //Posicion x del bloque

Y = y;

xplode = false;

//Posicion y del bloque

}

void Block::explotarBloque(){ //Explotar un bloque

color = 0;

}

void Block::printBloque(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);

/\*if(xplode)

printf("X");

else

printf(" ");\*/

}

void Block::comprobarColor(){ //Imprime el color de un bloque

printf("%d", color);

}