#include "include/common.h"

#include "include/libutils.h"

/\* ------------------ \*/

/\* VARIABLES GLOBALES \*/

/\* ------------------ \*/

/\*\*

\* Nivel de detalle para los mensajes (para mostrar mensajes de depuración, por ejemplo).

\*/

int nivel\_detalle = 0;

/\*\*

\* Nivel actual en el juego.

\*/

int nivel = 1;

/\*\*

\* Dimensiones de la matriz de juego.

\*

\* (por defecto, es de 4 x 5)

\*/

Dim tam\_matriz = {

.filas = 4,

.columnas = 5

};

/\*\*

\* Indica si el modo es automático (true) o manual (false).

\*/

bool modo\_auto = true;

/\*\*

\* Procesa los argumentos pasados por línea de comandos

\*

\* @param

\* Número de argumentos pasados (nº de elementos en argv).

\*

\* @param argv

\* Array de cadenas con los argumentos.

\*

\*

\* @return

\* -> ERR\_ARGS si se ha especificado alguna opción no reconocida.

\* -> SUCCESS si se han procesado los argumentos correctamente y se

\* debe proseguir con la ejecución.

\* -> SUCC\_ARGS si se ha procesado un argumento y se debe terminar la

\* ejecución (p.ej.: tras procesar '-h').

\*/

int procesar\_args (int argc, char \*argv [])

{

int res,

aux = 0;

while ((res = getopt (argc, argv, "hman:f:c:v")) != -1)

{

switch (res)

{

case 'h':

imprimir (DETALLE\_LOG, "%s\n", MSG\_AYUDA);

return SUCC\_ARGS;

case 'm':

modo\_auto = false;

break;

case 'a':

modo\_auto = true;

break;

case 'n':

aux = atoi (optarg);

/\* Si el nivel especificado es menor que o igual a 0,

(atoi devuelve 0 si se introduce algo que no es un

entero) se establece el nivel 1.

Si se indica un nivel mayor que el máximo

permitido, se establece el máximo nivel \*/

nivel = (aux <= 0)?

1

: (aux > MAX\_NV)? MAX\_NV : aux;

break;

case 'f':

aux = atoi (optarg);

tam\_matriz.filas = (aux <= 0)? 1 : aux;

break;

case 'c':

aux = atoi (optarg);

tam\_matriz.columnas = (aux <= 0)? 1 : aux;

break;

case 'v':

/\* Aumenta el nivel de detalle \*/

nivel\_detalle ++;

break;

default:

return ERR\_ARGS;

}

}

return SUCCESS;

}

/\*\*

\* Imprime toda la información de las variables globales del juego.

\*/

void imprimir\_info ()

{

imprimir (DETALLE\_DEBUG,

"------------------------------------------\n"

"Valor de las variables globales del juego:\n"

"\t--> Nivel de detalle: %i\n"

"\t--> Dimensiones del juego:\n"

"\t\tFilas: %i\n"

"\t\tColumnas: %i\n"

"\t--> Nivel: %i\n"

"\t--> Modo: %s\n"

"------------------------------------------\n",

nivel\_detalle,

tam\_matriz.filas,

tam\_matriz.columnas,

nivel,

(modo\_auto)? "automático" : "manual");

}

/\*\*

\* Cambia el valor de los parámetros del juego.

\*

\* @param nuevos\_params

\* Estructura de tipo Malla (definida en 'common.h') con los nuevos nivel

\* y dimensiones del tablero de juego.

\*/

void cambiar\_params (Malla nuevos\_params)

{

nivel = nuevos\_params.nivel;

tam\_matriz.filas = nuevos\_params.dimens.filas;

tam\_matriz.columnas = nuevos\_params.dimens.columnas;

}

/\*\*

\* Devuelve una estructura Malla con los valores especificados (nivel y dimensiones),

\* pero sin ninguna memoria reservada para la matriz.

\*

\*

\* @return

\* Una nueva instancia de tipo Malla, con los valores especificados por

\* línea de comandos.

\*/

Malla ver\_params ()

{

// Malla malla = {

//

// .dimens = tam\_matriz,

// .nivel = nivel,

// .matriz = 0

// };

/\* El compilador es una mierda, así que hay que inicializar a mano \*/

Malla malla;

malla.dimens = tam\_matriz;

malla.nivel = nivel;

malla.matriz = 0;

return malla;

}

/\*\*

\* Permite guardar la malla en el fichero especificado.

\*

\* @param malla

\* Estructura con toda la información del juego actual (nivel, dimensiones

\* de la matriz y el contenido de la matriz).

\*

\* @param nombre\_fichero

\* Nombre del fichero en el que se deben guardar los datos. Si ya existe se

\* sobrescribirá; si no, se creará.

\*

\*

\* @return

\* SUCCESS si los datos se han guardado correctamente.

\* ERR\_ARCHIVO si no se pudo abrir o cerrar correctamente el archivo.

\*/

int guardar (Malla malla, const char \*nombre\_fichero)

{

FILE \*fichero = fopen (nombre\_fichero, "w+");

int i,

j,

filas = malla.dimens.filas,

columnas = malla.dimens.columnas;

if (fichero == NULL)

{

imprimir (DETALLE\_LOG,

"Error al abrir el archivo '%s'\n",

nombre\_fichero);

return ERR\_ARCHIVO;

}

/\* Vuelca el contenido en el archivo en el siguiente orden (el mismo en el que

están declarados en 'common.h'):

Nivel

Dimensiones:

Filas

Columnas

Matriz

\*/

fprintf (fichero,

"-> Nivel: %i\n"

"-> Dimensiones:\n"

"\t--> Filas: %i\n"

"\t--> Columnas: %i\n"

"Matriz:\n",

malla.nivel,

filas,

columnas);

/\* Recorre a matriz guardando su contenido por filas \*/

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

{

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

{

fprintf (fichero, "%5i ", malla.matriz [(i \* columnas) + j].id);

}

fprintf (fichero, "\n");

}

if (fclose (fichero) != 0)

{

imprimir (DETALLE\_LOG,

"Error al cerrar el archivo '%s'\n",

nombre\_fichero);

return ERR\_ARCHIVO;

}

return SUCCESS;

}

/\*\*

\* Carga desde el fichero especificado el juego guardado.

\*

\* @param malla

\* Estructura en la que se va a cargar la información del juego.

\*

\* @param nombre\_fichero

\* Nombre del fichero que contiene la información del juego.

\*

\*

\* @return

\* SUCCESS si el archivo se cargó correctamente.

\* ERR\_ARCHIVO si hubo algún error al abrir o cerrar el fichero.

\*/

int cargar (Malla \*malla, const char \*nombre\_fichero)

{

FILE \*fichero = fopen (nombre\_fichero, "r");

int i,

j,

pos,

aux;

if (fichero == NULL)

{

printf ("Error al abrir el archivo.\n");

return ERR\_ARCHIVO;

}

/\* Obtiene el contenido del archivo en el siguiente orden (el mismo en el que

están declarados en 'common.h'):

Nivel

Dimensiones:

Filas

Columnas

Matriz

\*/

fscanf (fichero,

"-> Nivel: %i\n"

"-> Dimensiones:\n"

"\t--> Filas: %i\n"

"\t--> Columnas: %i\n"

"Matriz:\n",

&malla->nivel,

&malla->dimens.filas,

&malla->dimens.columnas);

/\* Reserva memoria para la matriz \*/

malla->matriz = (Diamante \*) malloc (malla->dimens.filas

\* malla->dimens.columnas

\* sizeof malla->matriz[0]);

if (malla->matriz == NULL)

{

imprimir (DETALLE\_LOG, "Error al reservar memoria para la matriz.\n");

return ERR\_MEM;

}

/\* Rellena la matriz con los valores del fichero \*/

for (i = 0; i < malla->dimens.filas; i++)

{

for (j = 0; j < malla->dimens.columnas; j++)

{

pos = (i \* malla->dimens.columnas) + j;

fscanf (fichero, "%5d ", &aux);

malla->matriz [pos] = crear\_diamante (aux);

}

fscanf (fichero, "\n");

}

/\* Cierra el archivo \*/

if (fclose (fichero) != 0)

{

imprimir (DETALLE\_LOG,

"Error al cerrar el archivo '%s'\n",

nombre\_fichero);

return ERR\_ARCHIVO;

}

return SUCCESS;

}

/\*\*

\* Reserva la memoria necesaria para el tablero de juego

\*

\* @param malla

\* Estructura de tipo Malla (definida en 'common.h') con las dimensiones de

\* la matriz y su contenido.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_MEM si hubo algún error al intentar reservar la memoria.

\*/

int reservar\_mem (Malla \*malla)

{

int filas = malla->dimens.filas,

columnas = malla->dimens.columnas,

i,

j,

aux;

malla->matriz = (Diamante \*) malloc (filas \* columnas \* sizeof malla->matriz[0]);

if (malla->matriz == NULL)

{

imprimir (DETALLE\_LOG,

"Error al intentar reservar la memoria para la matriz\n");

return ERR\_MEM;

}

/\* Inicializa la matriz para poner todas las casillas vacías \*/

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

{

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

{

//malla->matriz [(i \* columnas) + j].id = DIAMANTE\_VACIO;

aux = (i \* columnas) + j;

malla->matriz [aux] = crear\_diamante (DIAMANTE\_VACIO);

}

}

return SUCCESS;

}

/\*\*

\* Rellena la matriz de juego con diamantes aleatorios.

\*

\* @param malla

\* Estructura de tipo Malla (definida en 'common.h') con las dimensiones de

\* la matriz y su contenido.

\*

\*

\* @return

\* SUCCESS si todo ha salido correctamente.

\* ERR\_CUDA si alguna función CUDA ha fallado.

\*/

int rellenar (Malla \*malla)

{

/\* Llama a la función para CUDA encargada de llamar a los núcleos \*/

matriz\_aleat (malla);

return SUCCESS;

}

/\*\*

\* Imprime por pantalla el contenido de la matriz.

\*

\* @param malla

\* Estructura de tipo Malla (definida en 'common.h') con las dimensiones de

\* la matriz y su contenido.

\*/

void mostrar\_malla (Malla malla)

{

int i,

j,

filas = malla.dimens.filas,

columnas = malla.dimens.columnas;

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

{

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

{

imprimir (DETALLE\_LOG,

"%s ",

malla.matriz [(i \* columnas) + j].img);

}

imprimir (DETALLE\_LOG, "\n");

}

}

/\*\*

\* Crea un diamante del tipo especificado.

\*

\* @param num

\* Número del tipo de diamante a crear.

\*

\* @return

\* Una nueva instancia de diamante.

\*/

Diamante crear\_diamante (int num)

{

/\* Si se pasa por parametro num = 0, seria como eliminar un dimante \*/

Diamante d;

d.id = num;

switch (num)

{

case 0: d.img = NOR "#" RST; break;

case 1: d.img = AZU "1" RST; break;

case 2: d.img = ROJ "2" RST; break;

case 3: d.img = CYN "3" RST; break;

case 4: d.img = VER "4" RST; break;

case 5: d.img = ROS "5" RST; break;

case 6: d.img = AMA "6" RST; break;

case 7: d.img = NOR "7" RST; break;

case 8: d.img = BLA "8" RST; break;

}

return d;

}

/\*\*

\* Imprime por pantalla la cadena pasada como argumento sólo si el nivel de detalle es

\* el especificado (o mayor).

\*

\* @param detalle

\* Nivel de detalle mínimo para imprimir el mensaje

\*

\* @param cadena

\* Cadena con formato para imprimir

\*

\* @param ...

\* Argumentos para el formato de la cadena

\*/

void imprimir (int detalle, const char \*cadena, ...)

{

va\_list args;

va\_start (args, cadena);

if (nivel\_detalle >= detalle)

{

vprintf (cadena, args);

}

va\_end (args);

}

/\*\*

\* Obtiene el máximo diamante a generar, según el nivel especificado en la malla

\*

\* @param Malla

\* Estructura con la información del nivel actual.

\*

\*

\* @return

\* El valor máximo del diamante que se puede generar, en función del nivel.

\*/

int max\_nv (Malla malla)

{

switch (malla.nivel)

{

case 1:

return 4;

case 2:

return 6;

case 3:

return DIAMANTE\_MAX;

default:

return DIAMANTE\_MAX;

}

}

/\*\*

\* Obtiene el nivel de detalle actual.

\*/

int ver\_nv\_detalle ()

{

return nivel\_detalle;

}