**ARCHIVOS .H DEL PROYECTO**

**Integrantes:**

* Hidalgo Mauro DNI 40074392
* Lizarraga Matias DNI 41102400
* Gonzalez Julian DNI 40760818
* Baez Pablo Ezequiel DNI 42672975

**BANDIDO.H**

#ifndef BANDIDO\_H\_\_

#define BANDIDO\_H\_\_

#include <string>

#include "Lista.h"

#include <SDL.h>

using namespace std;

/\*\*

AXIOMAS:

Fila y Columna no pueden estar fuera de rango.

vidaBandido > 0 [0..VB]

intervaloFinal > 0 [0..IB]

\*/

/\*\*

Definición del tipo de dato.

\*/

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

// ESTRUCTURAS

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

typedef struct{

int fila;

int columna;

int cantidadItem;

string codItem;

int vidaBandido;

int intervaloFinal;//Cuando desaparece

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

}Bandido;

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

// PRIMITIVAS

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

/\*

pre : el bandido no debe haber sido creado.

post: el bandido queda creado y preparado para ser usado.

bandido : estructura de datos a ser creado.

\*/

void crearBandido(Bandido &bandido,SDL\_Renderer\* renderer, int fila,int columna, int anchoCelda, int altoCelda,string codItem, int cantidadItem,int vidaBandido,int intervaloFinal);

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

/\*

pre : el bandido debe haber sido creado.

post: se obtiene la fila donde se encuentra el bandido.

return : fila

\*/

int getFila(Bandido \*bandido);

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

/\*

pre : el bandido debe haber sido creado.

post: se obtiene la columna donde se encuentra el bandido.

return : fila

\*/

int getColumna(Bandido \*bandido);

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

/\*

pre : el bandido debe haber sido creado.

post: se obtiene la cantidadstring que pide el bandido

return : cantidadstring

\*/

int getCantidadItem(Bandido \*bandido);

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

/\*

pre : el bandido debe haber sido creado.

post: se setea la cantidadstring que pide el bandido

\*/

void setCantidadItem(Bandido &bandido, int cantidadItem);

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

/\*

pre : el bandido debe haber sido creado.

post: se obtiene el string que pide el bandido

return : string

\*/

string getCodItem(Bandido \*bandido);

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

///SETTERS

/\*

pre : el bandido debe haber sido creado.

post: se dibuja el bandido

bandido: estructura de datos a ser usado.

renderer

\*/

void dibujarBandido(Bandido \*bandido,SDL\_Renderer\* renderer);

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

/\*

pre : el bandido debe haber sido creado.

post: se destruye el bandido

bandido: estructura de datos a ser usado.

\*/

void destruirBandido(Bandido \*bandido);

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

void reemplazarBandido(Bandido \*bandido,SDL\_Renderer \*renderer);

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

#endif

**CAJA.H**

#ifndef \_CAJA\_H

#define \_CAJA\_H

#include <string>

using namespace std;

/\*\*

AXIOMAS:

codItem > 0

cantidadItem: No puede contener mas de 5 items

capacidadMaxima <= 25

capacidadActual >=0

\*/

/\*\*

Definición del tipo de dato.

\*/

typedef struct{

string codItem; //

int cantidadItem; //Cantidad del item

int capacidadMaxima; //Capacidad maxima de la caja

int capacidadActual;

}Caja;

/\*

pre:Caja no debe haber sido creada

post: inicializa parametros de Caja en 0

\*/

void crearCaja(Caja & caja,int cantidadItem,int capacidadActual,int capacidadMaxima,string codItem);

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

/\*

pre: Que la caja existe

post: Devuelve codItem

\*/

string getCodItem(Caja &caja);

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

/\*

pre: La caja debe haber sido creada

post: La cantidadItem queda registrada en la caja

\*/

void setCantidadItem(Caja &caja, int cantidadItem);

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

/\*

pre: La caja debe haber sido creada

post: Devuelve cantidad de item

\*/

int getCantidadItem(Caja &caja);

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

/\*

pre: La caja debe haber sido creada

post: Devuelve capacidad maxima

\*/

int getCapacidadMaxima(Caja &caja);

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

/\*

pre: La caja debe haber sido creada

post: Devuelve capacidad actual

\*/

int getCapacidadActual(Caja &caja);

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

/\*

pre: La caja debe haber sido creada

post: El codItem queda registrado en la caja

\*/

void setCodItem(Caja &caja, string codItem);

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

/\*

pre: La caja debe haber sido creada

post: La capacidadMaxima queda registrado en la caja

\*/

void setCapacidadMaxima(Caja &caja, int capacidadActual);

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

/\*

pre: La caja debe haber sido creada

post: La capacidadActual queda registrado en la caja

\*/

void setCapacidadActual(Caja &caja, int capacidadMaxima);

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

/\*

pre:La caja debe haber sido creada

post:La caja queda inutilizable

\*/

void eliminarCaja(Caja & caja);

#endif //\_CAJA\_H

**COLA.H**

#ifndef \_\_COLA\_H\_\_

#define \_\_COLA\_H\_\_

#ifndef NULL

#define NULL 0

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definiciones de Tipos de Datos \*/

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

/\* Tipo de Informacion que esta contenida en los Nodos de la

Cola, identificada como Dato. \*/

typedef void\* PtrDato;

/\* Tipo de Estructura de los Nodos de la Cola. \*/

struct NodoCola {

PtrDato ptrDato; // dato almacenado

NodoCola\* sgte; // puntero al siguiente

};

/\* Tipo de Puntero a los Nodos de la Cola, el cual se usa para recorrer

la Cola y acceder a sus Datos. \*/

typedef NodoCola\* PtrNodoCola;

/\* Tipo de Estructura de la Cola \*/

struct Cola{

PtrNodoCola primero; // puntero al primer nodo de la pila

PtrNodoCola ultimo; // puntero al ultimo nodo de la pila

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definicion de Primitivas \*/

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

/\*

pre : la cola no debe haber sido creada.

post: cola queda creada y preparada para ser usada.

cola : estructura de datos a ser creado.

\*/

void crearCola(Cola &cola);

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

/\*

pre : cola Creada con crearCola().

post: devuelve la representacion de lo Siguiente al último Nodo de la cola,

o sea el valor Null, que en esta implementacion representa el final de

la cola.

return: representación del fin de la cola.

\*/

PtrNodoCola finCola();

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

/\*

pre : cola creada con crearCola().

post: Devuelve true si cola esta vacia, sino devuelve false.

cola: cola sobre la cual se invoca la primitiva.

\*/

bool colaVacia(Cola &cola);

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

/\*

pre : cola creada con crearCola().

post: devuelve el puntero al primer elemento de la cola, o devuelve fin() si

esta vacia

cola: cola sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoCola colaFrente(Cola &cola);

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

/\*

pre : cola creada con crearCola().

post: devuelve el puntero al ultimo elemento de la cola, o devuelve fin() si

esta vacia

cola: cola sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoCola colaFin(Cola &cola);

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

/\*

pre : cola creada con crearCola().

post: agrega un nodo nuevo al final de la cola con el dato proporcionado

y devuelve un puntero a ese elemento.

cola: cola sobre la cual se invoca la primitiva.

dato : elemento a adicionar al principio de la cola.

return puntero al nodo adicionado.

\*/

PtrNodoCola encolar(Cola &cola, PtrDato ptrDato);

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

/\*

pre : cola creada con crearCola().

post: saca un nodo al frente de la cola con el dato proporcionado

y devuelve un puntero a ese elemento.

cola: cola sobre la cual se invoca la primitiva.

dato : elemento a adicionar al principio de la cola.

return puntero al nodo adicionado.

\*/

PtrDato desencolar(Cola &cola);

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

/\*

pre : cola creada con crearCola().

post: elimina todos los Nodos de la cola quedando destruida e inhabilitada

para su uso.

cola: cola sobre la cual se invoca la primitiva.

\*/

void eliminarCola(Cola &cola);

#endif

**COMANDA.H**

#ifndef COMANDA\_H

#define COMANDA\_H

#include <fstream>

using namespace std;

/\*\*

AXIOMAS:

codItem > 0

cantidad < 100 (podria variar segun el objetivo facil/medio/dificil)

\*/

/\*\*

Definición del tipo de dato.

\*/

//enum CodItem{oro=1,plata=2,bronce=3,platino=4,roca=5,carbon=6,NADA};

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

// ESTRUCTURAS

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

typedef struct{

string codItem;

int cantidad;

}Comanda;

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

// PRIMITIVAS

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

/\*

pre: La comanda no debe haber sido creada

post: La comanda queda registrada

\*/

void crearComanda(Comanda &comanda,string codItem, int cantidad);

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

/\*

pre: Comanda tiene que existir

post: comanda queda inutilizable

\*/

void destruirComanda(Comanda &comanda);

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

/\*

pre: Comanda tiene que existir

post: El codItem queda registrado en Comanda

\*/

void setCodItem(Comanda &comanda, string codItem);

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

/\*

pre: Comanda tiene que existir

post: Devuelve codItem

\*/

string getCodItem(Comanda &comanda);

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

/\*

pre: Comanda tiene que existir

post: La cantidad queda registrada en comanda

\*/

void setCantidad(Comanda &comanda, int cant);

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

/\*

pre: Comanda tiene que existir

post: Devuelve cantidad

\*/

int getCantidad(Comanda &comanda);

#endif // COMANDA\_H

**ESTACION.H**

#ifndef \_\_ESTACION\_H\_\_

#define \_\_ESTACION\_H\_\_

#include <SDL.h>

#include <SDL\_image.h>

#define CAPACIDAD\_VAGON 5

using namespace std;

/\*\*

AXIOMAS:

Representa la posicion donde se compran vagones. F y C no pueden estar fuera de rango.

La estacion "vende" vagones en relacion a la formula Peso\_maximo = 5kg\*cantidad\_monedas.

\*/

/\*\*

Definición del tipo de dato.

\*/

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

// ESTRUCTURAS

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

typedef struct Estacion{

int f;

int c;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

}Estacion;

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

// PRIMITIVAS

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

/\*

pre: Estacion no debe existir

post: Estacion es creada y devuelve los valores iniciales

estacion:Estructura de datos a utilizar

\*/

void crearEstacion(Estacion &estacion,SDL\_Renderer\* renderer, int f,int c, int anchoCelda,

int altoCelda);

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

/\*

pre : la estacion debe haber sido creada.

post: se obtiene la fila donde se encuentra la estacion.

return : fila

\*/

int getFila(Estacion \*estacion);

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

/\*

pre : la estacion debe haber sido creada.

post: se setea la fila donde se encontrara de la estacion.

estacion: estructura de datos a ser usada.

fila: fila donde estara la estacion.

\*/

void setFila(Estacion &estacion, int fila);

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

/\*

pre : la estacion debe haber sido creada.

post: se obtiene la columna donde se encuentra la estacion.

return : columna

\*/

int getColumna(Estacion \*estacion);

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

/\*

pre : la estacion debe haber sido creada.

post: se setea la columna donde se encontrara de la estacion.

estacion: estructura de datos a ser usada.

columna: columna donde estara la estacion.

\*/

void setColumna(Estacion &estacion, int columna);

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

/\*

pre : la estacion debe haber sido creada.

post: se dibuja la estacion

estacion: estructura de datos a ser usado.

renderer

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

void dibujarEstacion(Estacion \*estacion,SDL\_Renderer\* renderer);

/\*

pre: Estacion creada

post: calcula y devuelve peso que soporta vagon

\*/

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

int venderVagon(Estacion &estacion, int cantidadMonedas);

/\*

pre: Estacion creada

post: Se libera la memoria de la estacion y queda inutilizable

\*/

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

void destruirEstacion(Estacion &estacion);

#endif

**GAME.H**

#ifndef \_\_GAME\_H\_\_

#define \_\_GAME\_H\_\_

#include <SDL.h>

#include "Celda.h"

#include "Terreno.h"

#include "Lista.h"

#include "Comanda.h"

#include "Parametros.h"

#include "Estacion.h"

#include "Vagon.h"

/\*\*

AXIOMAS:

fila y columna no pueden estar fuera de rango.

fila >= 0

columna >= 0

intervalo > 0

contadorSegundo >=0

contadorMonedas >=0

anchoCelda > 0

altoCelda > 0

direccion: ["der","izq","arr","aba"]

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

// ESTRUCTURAS

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

typedef struct{

bool gameover;

int intervalo;

int contadorSegundo;

int contadorMonedas;

int anchoCelda;

int altoCelda;

int fila;

int columna;

Terreno\*\* terreno;

char direccion[4];

Lista minas;

Lista comanda;

Parametros parametros;

}Game;

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

// PRIMITIVAS

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

/\*

pre : game no debe haber sido creado.

post: game queda creado y preparado para ser usado.

Game : estructura de datos a ser creado.

\*/

void crearGame(Game &game,int fila,int columna,int anchoCelda,int altoCelda);

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

/\*

pre : game debe haber sido creado.

post: se obtiene el ancho de la Celda.

Game : estructura de datos a ser creado.

\*/

int getAnchoCelda(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se obtiene el alto de la Celda.

Game : estructura de datos a ser creado.

\*/

int getAltoCelda(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se obtiene el intervalo de tiempo.

Game : estructura de datos a ser creado.

\*/

int getIntervalo(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se setea el intervalo de tiempo.

Game : estructura de datos a ser creado.

intervalo: dato a asignar.

\*/

void setIntervalo(Game &game, int intervalo);

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

/\*

pre : game debe haber sido creado.

post: se obtiene el contador de los segundos transcurridos.

Game : estructura de datos a ser creado.

\*/

int getContadorSegundo(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se setea el contador de los segundos.

Game : estructura de datos a ser creado.

contadorSegundo: dato a asignar.

\*/

void setContadorSegundo(Game &game,int contadorSegundo);

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

/\*

pre : game debe haber sido creado.

post: se setea el terreno a utilizar.

Game : estructura de datos a ser creado.

render

\*/

void setTerreno(Game &game, SDL\_Renderer \* renderer);

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

/\*

pre : game debe haber sido creado.

post: se obtiene el terreno generado.

Game : estructura de datos a ser creado.

\*/

Terreno\*\* getTerreno(Game &game);

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

/\*

pre : game debe haber sido creado.

post: lee los datos almacenados en minas.txt para su uso.

Game : estructura de datos a ser creado.

render

\*/

void leerMinas(Game &game, SDL\_Renderer \* renderer);

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

/\*

pre : game debe haber sido creado.

post: crea una lista de Minas y devuelve su contenido.

Game : estructura de datos a ser creado.

\*/

Lista getMinas(Game &game);

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

/\*

pre : game debe haber sido creado.

post: crea una lista de comanda y devuelve su contenido.

Game : estructura de datos a ser creado.

\*/

Lista getComanda(Game &game);

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

/\*

pre : game debe haber sido creado.

post: lee los datos que se encuentran de comanda.txt para su uso.

Game : estructura de datos a ser creado.

\*/

void leerComanda(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se obtiene los datos de parametros.

Game : estructura de datos a ser creado.

\*/

Parametros getParametros(Game &game);

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

/\*

pre : game debe haber sido creado.

post: modifica los datos clave y valor de parametros.

Game : estructura de datos a ser creado.

clave: dato a asignar

valor: dato a asignar

\*/

void setParametros(Game &game, string clave, string valor);

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

/\*

pre : game debe haber sido creado.

post: lee los datos que se encuentran de parametros.txt para su uso.

Game : estructura de datos a ser creado.

\*/

void leerParametros(Game &game);

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

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

/\*

pre : game debe haber sido creado.

post: se dibuja el terreno.

Game : estructura de datos a ser creado.

render

\*/

void dibujarTerreno(Game game,SDL\_Renderer \*renderer);

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

/\*

pre : game debe haber sido creado.

post: se dibuja las entidades sobre el terreno.

Game : estructura de datos a ser creado.

render

\*/

void dibujarEntidades(Game game,SDL\_Renderer \*renderer);

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

/\*

pre : game debe haber sido creado.

post: indica la finalizacion del programa.

Game : estructura de datos a ser creado.

\*/

bool getGameOver(Game &game);

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

/\*

pre : game debe haber sido creado.

post: setea el estado del programa e indica su finalizacion.

Game : estructura de datos a ser creado.

flag: dato a asignar

\*/

void setGameOver(Game &game, bool flag);

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

/\*

pre : game debe haber sido creado.

post: se elimina game.

Game : estructura de datos a ser creado.

render

\*/

void destruirGame(Game &game,SDL\_Renderer\* renderer,SDL\_Window \*window,Lista &locomotora,Lista &bandidos, Lista &monedas);

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

/\*

pre : game debe haber sido creado.

post: indica la posicion donde se situara el vagon en el terreno.

Game : estructura de datos a ser creado.

\*/

void ubicarVagon(Game &game,PtrNodoLista ptrNodo);

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

/\*

pre : game debe haber sido creado.

post: indica la posicion donde se situara el bandido en el terreno.

Game : estructura de datos a ser creado.

\*/

void ubicarBandido(Game &game,Bandido \*bandido);

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

/\*

pre : game debe haber sido creado.

post: indica la posicion donde se situaran las minas en el terreno.

Game : estructura de datos a ser creado.

\*/

void ubicarMina(Game &game,Mina \*mina);

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

/\*

pre : game debe haber sido creado.

post: indica la posicion donde se situara moneda en el terreno.

Game : estructura de datos a ser creado.

\*/

void ubicarMoneda(Game &game,Moneda \*moneda);

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

/\*

pre : game debe haber sido creado.

post: indica la posicion donde se situara la estacion en el terreno.

Game : estructura de datos a ser creado.

\*/

void ubicarEstacion(Game &game,Estacion \*estacion);

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

/\*

pre : game debe haber sido creado.

post: situa el movimiento del vagon.

Game : estructura de datos a ser creado.

direccion: dato a asignar

\*/

void setDireccion(Game &game,char direccion[]);

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

/\*

pre : game debe haber sido creado.

post: se obtiene la direccion del movimiento del vagon.

Game : estructura de datos a ser creado.

\*/

char\* getDireccion(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se obtiene la cantidad de columnas a utilizar.

Game : estructura de datos a ser creado.

\*/

int getColumna(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se obtiene la cantidad de filas a utilizar.

Game : estructura de datos a ser creado.

\*/

int getFila(Game &game);

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

/\*

pre : game debe haber sido creado.

post: se controla el desplazamiento del vagon y sus interacciones.

Game : estructura de datos a ser creado.

\*/

void evaluarColision(Game &game,Lista &locomotora,Lista &monedas,Lista &bandidos,SDL\_Renderer\* renderer);

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

/\*

pre : game debe haber sido creado.

post: al interactuar con mina se modifica sus datos.

Game : estructura de datos a ser creado.

\*/

void actualizarMinas(Game &game);

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

/\*

pre : game debe haber sido creado.

post: indica la cantidad de interacciones con moneda.

Game : estructura de datos a ser creado.

\*/

int contadorMonedas(Game &game);

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

/\*

pre : game debe haber sido creado.

post: situa la cantidad de interacciones de moneda.

Game : estructura de datos a ser creado.

\*/

void setContadorMonedas(Game &game, int contadorMonedas);

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

/\*

pre : game debe haber sido creado.

post: bandido es generado y ubicado en el terreno.

Game : estructura de datos a ser creado.

\*/

void generarBandidos(Game &game,Lista &bandidos, SDL\_Renderer\* renderer,int VB);

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

/\*

pre : game debe haber sido creado.

post: moneda es generado y ubicado en el terreno.

Game : estructura de datos a ser creado.

\*/

void generarMonedas(Game &game,Lista &monedas, SDL\_Renderer\* renderer,int VM);

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

/\*

pre : game debe haber sido creado.

post: bandido es eliminado al finalizar su intervalo de vida.

Game : estructura de datos a ser creado.

\*/

void actualizarBandidos(Game &game,Lista &bandidos, SDL\_Renderer\* renderer);

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

/\*

pre : game debe haber sido creado.

post: moneda es eliminada al finalizar su intervalo de vida.

Game : estructura de datos a ser creado.

\*/

void actualizarMonedas(Game &game,Lista &monedas, SDL\_Renderer\* renderer);

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

/\*

pre : game debe haber sido creado.

post: se dibuja la puntuacion.

Game : estructura de datos a ser creado.

\*/

void dibujarPuntuacion(Game &game, SDL\_Renderer \* renderer,SDL\_Window \* window, Lista &locomotora,Lista &monedas,Lista &bandidos);

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

/\*

pre : game debe haber sido creado.

post: vagon es situado en una nueva posicion del terreno.

Game : estructura de datos a ser creado.

\*/

void nuevaPosicionVagon(Game &game, Vagon \* ultimo, int &posX, int &posY, char \* direccion);

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

/\*

pre : game debe haber sido creado.

post: al interactuar con estacion y cumplir las condiciones, se crea un nuevo vagon.

Game : estructura de datos a ser creado.

\*/

void agregarVagon(Game &game, Lista &locomotora, Estacion &estacion);

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

/\*

pre : se debe haber ingresado el dato

post: devuelve el valor de la cadena como un entero.

Game : estructura de datos a ser creado.

\*/

int numeroStringRand(string numero);

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

/\*

pre : game debe haber sido creado.

post: al interactuar la locomotora con el bandido se modifican/eliminan los items almacenados o

en caso de no poseer, se elimina un vagon/locomotora.

Game : estructura de datos a ser creado.

\*/

void encuentroConBandido(Game &game, Lista &locomotora, Lista &bandidos);

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

/\*

pre : game debe haber sido creado.

post: situa la cantidad de items que seran eliminados al interactuar con bandido.

Game : estructura de datos a ser creado.

\*/

void pedirItem(Game &game,Lista &locomotora, Bandido &bandido);

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

/\*

pre : game debe haber sido creado.

post: se obtiene la direccion del vagon, en base a las teclas seleccionadas.

Game : estructura de datos a ser creado.

\*/

void controlarEventos(Game &game,SDL\_Event &event,const unsigned char \*keys);

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

/\*

pre : game debe haber sido creado.

post: se situa la direccion de la locomotora.

Game : estructura de datos a ser creado.

\*/

void cambiarDireccion(Game &game,Lista &locomotora);

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

/\*

pre : game debe haber sido creado.

post: modifica el valor de la fila en el que se encuentra la locomotora.

Game : estructura de datos a ser creado.

\*/

void cambiarFila(Game &game,PtrNodoLista ptrNodo);

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

/\*

pre : game debe haber sido creado.

post: modifica el valor de la columna en el que se encuentra la locomotora.

Game : estructura de datos a ser creado.

\*/

void cambiarColumna(Game &game,PtrNodoLista ptrNodo);

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

/\*

pre : game debe haber sido creado.

post: devuelve el estado de la locomotora y su interaccion con el entorno.

Game : estructura de datos a ser creado.

return: estado de gameover

\*/

bool evaluarColisiones(Game &game,PtrNodoLista ptrNodo);

#endif

**LISTA.H**

#ifndef \_\_LISTA\_H\_\_

#define \_\_LISTA\_H\_\_

#ifndef NULL

#define NULL 0

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definiciones de Tipos de Datos \*/

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

/\* tipo enumerado para realizar comparaciones \*/

enum ResultadoComparacion {

MAYOR,

IGUAL,

MENOR

};

/\* Tipo de Informacion que esta contenida en los Nodos de la

Lista, identificada como Dato. \*/

typedef void\* PtrDato;

/\* Tipo de Estructura de los Nodos de la Lista. \*/

struct NodoLista {

PtrDato ptrDato; // dato almacenado

NodoLista\* sgte; // puntero al siguiente

};

/\* Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer

la Lista y acceder a sus Datos. \*/

typedef NodoLista\* PtrNodoLista;

typedef ResultadoComparacion (\*PFComparacion)(PtrDato , PtrDato);

/\* Tipo de Estructura de la Lista \*/

struct Lista{

PtrNodoLista primero; // puntero al primer nodo de la lista

PFComparacion compara;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definicion de Primitivas \*/

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

lista : estructura de datos a ser creado.

\*/

void crearLista(Lista &lista, PFComparacion funcComparacion);

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

/\*

pre : lista Creada con crearLista().

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null, que en esta implementacion representa el final de

la lista.

return: representación del fin de la lista.

\*/

PtrNodoLista finLista();

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

/\*

pre : lista Creada con crearLista().

post: Devuelve true si lista esta vacia, sino devuelve false.

lista : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(Lista &lista);

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

/\*

pre : lista Creada con crearLista().

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null, que en esta implementacion representa el final de

la lista.

return representación del fin de la lista.

\*/

PtrNodoLista fin();

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

/\*

pre : lista Creada con crearLista().

post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si

esta vacia

lista : lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoLista primero(Lista &lista);

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

/\*

pre : lista Creada con crearLista().

post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si

ptrNodo apuntaba a fin() o si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

prtNodo : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoLista siguiente(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista Creada con crearLista().

ptrNodo es un puntero a un nodo de lista.

post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

prtNodo : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si

si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoLista ultimo(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo despues del apuntado por ptrNodo con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

Si la lista esta vacía agrega un nodo al principio de esta y devuelve

un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta

nada y devuelve fin().

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar.

ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarDespues(Lista &lista, PtrDato ptrDato, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarFinal(Lista &lista, PtrDato ptrDato);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo

y devuelve un puntero al nodo insertado. Si la lista esta vacia no

inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo

insertado sera el nuevo primero.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar.

ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarAntes(Lista &lista, PtrDato ptrDato, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista

esta vacia o si ptrNodo apunta a fin().

lista : lista sobre la cual se invoca la primitiva.

ptrNodo : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: si la lista no esta vacia, elimina su nodo primero, sino no realiza

accion alguna.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: si la lista no esta vacia elimina su nodo ultimo,

sino no realiza accion.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: elimina todos los Nodos de la lista quedando destruida e inhabilitada

para su uso.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(Lista &lista);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definición de Operaciones Adicionales \*/

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

/\*

pre : lista fue creada con crearLista().

post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo

que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a localizar.

return puntero al nodo localizado o fin().

\*/

PtrNodoLista localizarDato(Lista &lista , PtrDato ptrDato);

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

/\*

pre : lista fue creada con crearLista() y cargada con datos ordenados de

menor a mayor respecto del sentido progresivo.

post: agrega a la lista el dato manteniendo el orden pero con multiples

valores iguales y devuelve un puntero al nodo insertado.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a insertar.

return puntero al nodo insertado.

\*/

PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato);

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

/\*

pre : la lista fue creada con crearLista().

post : elimina el dato de la lista, si el mismo se encuentra.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a eliminar.

\*/

void eliminarDato(Lista &lista, PtrDato ptrDato);

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

/\*

pre : la lista fue creada con crearLista().

post : reordena la lista.

lista : lista sobre la cual se invoca la primitiva.

\*/

void reordenar(Lista &lista);

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

/\*

pre : la lista fue creada con crearLista().

post : devuelve la cantidad de datos que tiene la lista.

lista : lista sobre la cual se invoca la primitiva.

\*/

int longitud(Lista &lista);

#endif

#ifndef \_\_LISTA\_H\_\_

#define \_\_LISTA\_H\_\_

#ifndef NULL

#define NULL 0

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definiciones de Tipos de Datos \*/

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

/\* tipo enumerado para realizar comparaciones \*/

enum ResultadoComparacion {

MAYOR,

IGUAL,

MENOR

};

/\* Tipo de Informacion que esta contenida en los Nodos de la

Lista, identificada como Dato. \*/

typedef void\* PtrDato;

/\* Tipo de Estructura de los Nodos de la Lista. \*/

struct NodoLista {

PtrDato ptrDato; // dato almacenado

NodoLista\* sgte; // puntero al siguiente

};

/\* Tipo de Puntero a los Nodos de la Lista, el cual se usa para recorrer

la Lista y acceder a sus Datos. \*/

typedef NodoLista\* PtrNodoLista;

typedef ResultadoComparacion (\*PFComparacion)(PtrDato , PtrDato);

/\* Tipo de Estructura de la Lista \*/

struct Lista{

PtrNodoLista primero; // puntero al primer nodo de la lista

PFComparacion compara;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definicion de Primitivas \*/

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

/\*

pre : la lista no debe haber sido creada.

post: lista queda creada y preparada para ser usada.

lista : estructura de datos a ser creado.

\*/

void crearLista(Lista &lista, PFComparacion funcComparacion);

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

/\*

pre : lista Creada con crearLista().

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null, que en esta implementacion representa el final de

la lista.

return: representación del fin de la lista.

\*/

PtrNodoLista finLista();

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

/\*

pre : lista Creada con crearLista().

post: Devuelve true si lista esta vacia, sino devuelve false.

lista : lista sobre la cual se invoca la primitiva.

\*/

bool listaVacia(Lista &lista);

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

/\*

pre : lista Creada con crearLista().

post: devuelve la representacion de lo Siguiente al último Nodo de la lista,

o sea el valor Null, que en esta implementacion representa el final de

la lista.

return representación del fin de la lista.

\*/

PtrNodoLista fin();

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

/\*

pre : lista Creada con crearLista().

post: devuelve el puntero al primer elemento de la lista, o devuelve fin() si

esta vacia

lista : lista sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoLista primero(Lista &lista);

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

/\*

pre : lista Creada con crearLista().

post: devuelve el puntero al nodo proximo del apuntado, o devuelve fin() si

ptrNodo apuntaba a fin() o si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

prtNodo : puntero al nodo a partir del cual se requiere el siguiente.

return puntero al nodo siguiente.

\*/

PtrNodoLista siguiente(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista Creada con crearLista().

ptrNodo es un puntero a un nodo de lista.

post: devuelve el puntero al nodo anterior del apuntado, o devuelve fin() si

ptrNodo apuntaba al primero o si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

prtNodo : puntero al nodo a partir del cual se requiere el anterior.

return puntero al nodo anterior.

\*/

PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: devuelve el puntero al ultimo nodo de la lista, o devuelve fin() si

si lista esta vacia.

lista : lista sobre la cual se invoca la primitiva.

return puntero al último nodo.

\*/

PtrNodoLista ultimo(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo nuevo al principio de la lista con el dato proporcionado

y devuelve un puntero a ese elemento.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar al principio de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo despues del apuntado por ptrNodo con el dato

proporcionado y devuelve un puntero apuntado al elemento insertado.

Si la lista esta vacía agrega un nodo al principio de esta y devuelve

un puntero al nodo insertado. Si ptrNodo apunta a fin() no inserta

nada y devuelve fin().

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar.

ptrNodo : puntero al nodo después del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarDespues(Lista &lista, PtrDato ptrDato, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo al final de la lista con el dato proporcionado y devuelve

un puntero al nodo insertado.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar al final de la lista.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarFinal(Lista &lista, PtrDato ptrDato);

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

/\*

pre : lista creada con crearLista().

post: agrega un nodo con el dato proporcionado antes del apuntado por ptrNodo

y devuelve un puntero al nodo insertado. Si la lista esta vacia no

inserta nada y devuelve fin(). Si ptrNodo apunta al primero, el nodo

insertado sera el nuevo primero.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a adicionar.

ptrNodo : puntero al nodo antes del cual se quiere adicionar el dato.

return puntero al nodo adicionado.

\*/

PtrNodoLista adicionarAntes(Lista &lista, PtrDato ptrDato, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: elimina el nodo apuntado por ptrNodo. No realiza accion si la lista

esta vacia o si ptrNodo apunta a fin().

lista : lista sobre la cual se invoca la primitiva.

ptrNodo : puntero al nodo que se desea eliminar.

\*/

void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo);

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

/\*

pre : lista creada con crearLista().

post: si la lista no esta vacia, elimina su nodo primero, sino no realiza

accion alguna.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoPrimero(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: si la lista no esta vacia elimina su nodo ultimo,

sino no realiza accion.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarNodoUltimo(Lista &lista);

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

/\*

pre : lista creada con crearLista().

post: elimina todos los Nodos de la lista quedando destruida e inhabilitada

para su uso.

lista : lista sobre la cual se invoca la primitiva.

\*/

void eliminarLista(Lista &lista);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definición de Operaciones Adicionales \*/

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

/\*

pre : lista fue creada con crearLista().

post: si el dato se encuentra en la lista, devuelve el puntero al primer nodo

que lo contiene. Si el dato no se encuentra en la lista devuelve fin().

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a localizar.

return puntero al nodo localizado o fin().

\*/

PtrNodoLista localizarDato(Lista &lista , PtrDato ptrDato);

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

/\*

pre : lista fue creada con crearLista() y cargada con datos ordenados de

menor a mayor respecto del sentido progresivo.

post: agrega a la lista el dato manteniendo el orden pero con multiples

valores iguales y devuelve un puntero al nodo insertado.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a insertar.

return puntero al nodo insertado.

\*/

PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato);

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

/\*

pre : la lista fue creada con crearLista().

post : elimina el dato de la lista, si el mismo se encuentra.

lista : lista sobre la cual se invoca la primitiva.

dato : elemento a eliminar.

\*/

void eliminarDato(Lista &lista, PtrDato ptrDato);

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

/\*

pre : la lista fue creada con crearLista().

post : reordena la lista.

lista : lista sobre la cual se invoca la primitiva.

\*/

void reordenar(Lista &lista);

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

/\*

pre : la lista fue creada con crearLista().

post : devuelve la cantidad de datos que tiene la lista.

lista : lista sobre la cual se invoca la primitiva.

\*/

int longitud(Lista &lista);

#endif

**MINA.H**

#ifndef \_\_MINA\_H\_\_

#define \_\_MINA\_H\_\_

#include <string>

#include <SDL.h>

#include "Cola.h"

#include "Lista.h"

using namespace std;

/\*\*

AXIOMAS:

F y C no pueden estar fuera de rango.

codItem: ["oro", "plata", "bronce", "platino", "roca", "carbon"]

f >= 0

c >= 0

ip > 0

secuencia[] > 0

secuenciaActual > 0

intervaloActual > 0

\*/

/\*\*

Definición del tipo de dato.

\*/

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

// ESTRUCTURAS

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

typedef struct{

int f;

int c;

int ip;

string codItem;

int secuencia[5];

int secuenciaActual;

int intervaloActual;

Cola cajas;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

}Mina;

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

// PRIMITIVAS

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

/\*

pre : la mina no debe haber sido creada.

post: la mina queda creada y preparado para ser usado.

Mina : estructura de datos a ser creado.

\*/

void crearMina(Mina &mina,SDL\_Renderer\* renderer, int f,int c, int anchoCelda,

int altoCelda, int ip, int secuencia[5],string codItem);

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

/\*

pre : la mina debe haber sido creada.

post: se obtiene la fila donde se encuentra la mina.

return : fila

\*/

int getFila(Mina \*mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea la fila donde se encontrara de la mina.

mina: estructura de datos a ser usada.

fila: fila donde estara la mina.

\*/

void setFila(Mina &mina, int fila);

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

/\*

pre : la mina debe haber sido creada.

post: se obtiene la columna donde se encuentra la mina.

return : columna

\*/

int getColumna(Mina \*mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea la columna donde se encontrara de la mina.

mina: estructura de datos a ser usada.

columna: columna donde estara la mina.

\*/

void setColumna(Mina &mina, int columna);

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

/\*

pre : la mina debe haber sido creado.

post: se obtiene el intervalo de produccion de la mina.

return: intervalo de produccion

\*/

int getIP(Mina &mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea el intervalo de produccion de la mina.

mina: estructura de datos a ser usada.

IP: intervalo de produccion a setear

\*/

void setIP(Mina &mina, int IP);

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

/\*

pre : la mina debe haber sido creado.

post: se obtiene el la secuencia de produccion de la mina.

return: secuencia de produccion

\*/

int\* getSecuencia(Mina &mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea la secuencia de la mina.

mina: estructura de datos a ser usada.

secuencia: secuencia de la mina a setear.

\*/

void setSecuencia(Mina &mina, int secuencia[5]);

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

/\*

pre : la mina debe haber sido creada.

post: se obtiene la secuencia actual de la mina.

mina: estructura de datos a ser usada.

return: secuencia actual de la mina.

\*/

int getSecuenciaActual(Mina &mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea la secuencia actual de la mina.

mina: estructura de datos a ser usada.

intervalo: secuencia actual de la mina.

\*/

void setSecuenciaActual(Mina &mina, int secuencia);

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

/\*

pre : la mina debe haber sido creada.

post: se obtiene el intervalo actual de la mina.

mina: estructura de datos a ser usada.

return: intervalo actual de la mina.

\*/

int getIntervarloActual(Mina &mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea el intervalo actual de la mina.

mina: estructura de datos a ser usada.

intervalo: intervalo actual de la mina.

\*/

void setIntervaloActual(Mina &mina, int intervalo);

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

/\*

pre : la mina debe haber sido creado.

post: se obtiene el codItem que produce la mina.

return: item

\*/

string getCodItem(Mina \*mina);

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

/\*

pre : la mina debe haber sido creada.

post: se setea el item de la mina.

mina: estructura de datos a ser usada.

codItem: item que producira la mina.

\*/

void setCodItem(Mina &mina, string codItem);

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

/\*

pre : la mina debe haber sido creado.

post: se dibuja la mina

mina: estructura de datos a ser usado.

renderer

\*/

void dibujarMina(Mina \*mina,SDL\_Renderer\* renderer);

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

/\*

pre : la mina debe haber sido creado.

post: se destruye la mina

mina: estructura de datos a ser usado.

\*/

void destruirMina(Mina \*mina);

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

/\*

pre : la mina debe haber sido creado.

post: se le asignan las cajas de la mina a los vagones correspondientes.

mina: estructura de datos a ser usado.

locomotora: lista de vagones.

\*/

void recolectarCajas(Mina &mina, Lista &locomotora);

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

#endif

**MONEDA.H**

#ifndef \_\_MONEDA\_H\_\_

#define \_\_MONEDA\_H\_\_

#include <SDL.h>

/\*\*

AXIOMAS:

fila y columna no pueden estar fuera de rango.

fila > 0

columna > 0

invervaloFinal > 0 [0..IM]

monedaVida > 0 [0..VM]

\*/

/\*\*

Definición del tipo de dato.

\*/

typedef struct{

int fila;

int columna;

int monedaVida;

int intervaloFinal;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

}Moneda;

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

// PRIMITIVAS

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

/\*

pre : la moneda no debe haber sido creada.

post: la moneda queda creada y preparada para ser usada.

moneda : estructura de datos a ser creado.

\*/

void crearMoneda(Moneda &moneda,SDL\_Renderer\* renderer, int fila,int columna, int anchoCelda, int altoCelda,int intervaloVida,int intervaloFinal);

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

/\*

pre : la moneda debe haber sido creada.

post: se obtiene la fila donde se encuentra la moneda.

return : fila

\*/

int getFila(Moneda \*moneda);

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

/\*

pre : la moneda debe haber sido creada.

post: se obtiene la columna donde se encuentra la moneda.

return : columna

\*/

int getColumna(Moneda \*moneda);

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

/\*

pre : la moneda tiene que estar creada.

post: Se obtiene el intervaloVida

moneda : estructura de datos a ser usada.

\*/

int getIntervaloVida(Moneda &moneda);

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

/\*

pre : la moneda tiene que estar creada.

post: Se setea el intervaloVida

moneda : estructura de datos a ser usada.

return:intervaloVida

\*/

bool setIntervaloVida(Moneda &moneda, int monedaVida);

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

/\*

pre : la moneda tiene que estar creada.

post: Se obtiene el intervaloVida

moneda : estructura de datos a ser usada.

\*/

int getIntervaloFinal(Moneda &moneda);

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

/\*

pre : la moneda tiene que estar creada.

post: Se setea el intervaloVida

moneda : estructura de datos a ser usada.

return:intervaloVida

\*/

bool setIntervaloFinal(Moneda &moneda, int intervaloFinal);

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

/\*

pre : la moneda debe haber sido creada.

post: se dibuja la moneda

moneda: estructura de datos a ser usado.

renderer

\*/

void dibujarMoneda(Moneda \*moneda,SDL\_Renderer\* renderer);

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

/\*

pre : la moneda debe haber sido creado.

post: se destruye la moneda.

moneda: estructura de datos a ser usado.

\*/

void destruirMoneda(Moneda \*moneda);

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

/\*

pre : la moneda debe haber sido creado.

post: se reemplaza (setea) los datos de moneda.

moneda: estructura de datos a ser usado.

\*/

void reemplazarMoneda(Moneda \*moneda,SDL\_Renderer \*renderer);

#endif

**PARAMETROS.H**

#ifndef PARAMETROS\_H

#define PARAMETROS\_H

#include <string>

using namespace std;

/\*\*

AXIOMAS:

claveS > 0

claveP > 0

claveA > 0

claveposXE >= 0

claveposYE >= 0

claveIM > 0

claveVM > claveIM

claveIB > 0

claveVB > claveIB

claveIP > 0

valor > 0

\*/

/\*\* Definicion del tipo de datos \*\*/

typedef struct{

string claveS;

string claveP;

string claveA;

string claveposXE;

string claveposYE;

string claveIM;

string claveVM;

string claveIB;

string claveVB;

string claveIP;

string valor;

}Parametros;

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

// PRIMITIVAS

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

/\* pre: la instancia Parametros no debe haberse creado ni estar destruida.

post: Parametros creado y listo para ser usado.

Parametros: instancia sobre la que se invoca la primitiva .

\*/

void crearParametros(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado y no debe estar destruida.

post: destruye la instancia de Parametros y ya no podrá reutilizarse.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

void destruirParametros(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea los segundos del intervalo.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveS(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea la cantidad Max del pedido del bandido.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveP(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el area de ataque del bandido.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveA(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea la posicion X (fila) de la estacion.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClavePosXE(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea la posicion Y (cola) de la estacion.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClavePosYE(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el intervalo de generacion de moneda.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveIM(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el invervalo de vida de la moneda.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveVM(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el intervalo maximo de generacion de bandidos.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveIB(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el intervalo de vida del bandido.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveVB(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: setea el intervalo de produccion de las minas.

Parametros: instancia sobre la que se invoca la primitiva.

clav: dato a asignar.

\*/

void setClaveIP(Parametros &parametros, string clav);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene los segundos del intervalo.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveS(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene la cantidad Max que solicita el bandido.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveP(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el area del ataque del bandido.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveA(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene la posicion X (Fila) de la estacion.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClavePosXE(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene la posicion Y (Columna) de la estacion.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClavePosYE(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el intervalo de generacion de la moneda.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveIM(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el intervalo de vida de la moneda.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveVM(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el intervalo de generacion del bandido.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveIB(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el intervalo de vida deñ bandido.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveVB(Parametros &parametros);

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

/\* pre: la instancia Parametros debe haberse creado.

post: se obtiene el intervalo de produccion de las minas.

Parametros: instancia sobre la que se invoca la primitiva.

\*/

string getClaveIP(Parametros &parametros);

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

//void setValor(Parametros &parametros, string val);

//string getValor(Parametros &parametros);

#endif // CONFIGURACION\_H\_INCLUDED

**PILA.H**

#ifndef \_\_PILA\_H\_\_

#define \_\_PILA\_H\_\_

#ifndef NULL

#define NULL 0

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definiciones de Tipos de Datos \*/

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

/\* Tipo de Informacion que esta contenida en los Nodos de la

Pila, identificada como Dato. \*/

typedef void\* PtrDato;

/\* Tipo de Estructura de los Nodos de la Pila. \*/

struct NodoPila {

PtrDato ptrDato; // dato almacenado

NodoPila\* sgte; // puntero al siguiente

};

/\* Tipo de Puntero a los Nodos de la Pila, el cual se usa para recorrer

la Pila y acceder a sus Datos. \*/

typedef NodoPila\* PtrNodoPila;

/\* Tipo de Estructura de la Lista \*/

struct Pila{

PtrNodoPila primero; // puntero al primer nodo de la pila

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Definicion de Primitivas \*/

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

/\*

pre : la pila no debe haber sido creada.

post: pila queda creada y preparada para ser usada.

pila : estructura de datos a ser creado.

\*/

void crearPila(Pila &pila);

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

/\*

pre : pila Creada con crearPila().

post: devuelve la representacion de lo Siguiente al último Nodo de la pila,

o sea el valor Null, que en esta implementacion representa el final de

la pila.

return: representación del fin de la pila.

\*/

PtrNodoPila finPila();

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

/\*

pre : pila creada con crearPila().

post: Devuelve true si pila esta vacia, sino devuelve false.

pila: pila sobre la cual se invoca la primitiva.

\*/

bool pilaVacia(Pila &pila);

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

/\*

pre : pila creada con crearPila().

post: devuelve el puntero al primer elemento de la pila, o devuelve fin() si

esta vacia

pila: pila sobre la cual se invoca la primitiva.

return puntero al primer nodo.

\*/

PtrNodoPila cima(Pila &pila);

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

/\*

pre : pila creada con crearPila().

post: agrega un nodo nuevo al principio de la pila con el dato proporcionado

y devuelve un puntero a ese elemento.

pila: pila sobre la cual se invoca la primitiva.

dato : elemento a adicionar al principio de la pila.

return puntero al nodo adicionado.

\*/

PtrNodoPila agregar(Pila &pila, PtrDato ptrDato);

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

/\*

pre : pila creada con crearPila().

post: obtiene el ultimo dato agregado a la pila, elimina el nodo

y devuelve un puntero al elemento anteriormente agregado.

pila: pila sobre la cual se invoca la primitiva.

dato : elemento obtenido de la pila.

return puntero al nuevo ultimo nodo.

\*/

PtrDato sacar(Pila &pila);

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

/\*

pre : pila creada con crearPila().

post: elimina todos los Nodos de la pila quedando destruida e inhabilitada

para su uso.

pila: pila sobre la cual se invoca la primitiva.

\*/

void eliminarPila(Pila &pila);

#endif

**TERRENO.H**

#ifndef \_\_TERRENO\_H\_\_

#define \_\_TERRENO\_H\_\_

#include "Bandido.h"

#include "Mina.h"

#include "Moneda.h"

#include "Lista.h"

#include "Estacion.h"

/\*\*

AXIOMAS:

Representa la zona disponible para que el tren se mueva

fila >= 0

columna >= 0

\*/

/\*\*

Definición del tipo de dato.

\*/

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

// ESTRUCTURAS

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

typedef struct{

int fila;

int columna;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

PtrNodoLista ptrNodoVagon;

Bandido \*ptrBandido;

Mina \*ptrMina;

Moneda \*ptrMoneda;

Estacion \*ptrEstacion;

}Terreno;

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

// PRIMITIVAS

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

/\*

pre : el terreno no debe haber sido creado.

post: el terreno queda creado y preparado para ser usado.

terreno : estructura de datos a ser creado.

\*/

void crearTerreno(Terreno &terreno,int fila,int columna, int ancho, int alto, SDL\_Renderer\* renderer,char rand[]);

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

/\*

pre : el terreno debe haber sido creado.

post: se dibuja el terreno

bandido: estructura de datos a ser usado.

\*/

void dibujarTerreno(Terreno &terreno, SDL\_Renderer\* renderer);

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

/\*

pre : el terreno debe haber sido creado.

post: se dibuja la entidad

bandido: estructura de datos a ser usado.

\*/

void dibujarEntidad(Terreno &terreno, SDL\_Renderer\* renderer, int intervalo);

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

/\*

pre : el terreno debe haber sido creado.

post: se destruye el terreno

terreno: estructura de datos a ser usado.

\*/

void destruirTerreno(Terreno &terreno);

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

/\*

pre : lista Creada con crearLista().

post: devuelve el siguiente al último nodo de la lista,

\*/

#endif

**VAGON.H**

#ifndef \_\_VAGON\_H\_\_

#define \_\_VAGON\_H\_\_

#include "Item.h"

#include <SDL.h>

#include "Pila.h"

#include <string>

/\*\*

AXIOMAS:

El vagon inicia en la posicion dada por fila y columna

El vagon inicia con una capacidad definida por 5kg\*cantMonedas

El vagon solo puede almacenar items de su mismo tipo dado por codItem

miniatura: ["c1","c2"]

fila >= 0

columna >= 0

direccion: ["izq","der","arr","aba"]

anchoCelda > 0

altoCelda > 0

altoSprite > 0

cantMonedas > 0

capacidad = 5 \* cantMonedas

cantidadItems >= 0

\*/

/\*\*

Definición del tipo de dato.

\*/

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

// ESTRUCTURAS

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

typedef struct{

char miniatura[3];

int fila;

int columna;

char direccion[4];

int anchoCelda;

int altoCelda;

int altoSprite;

SDL\_Texture \*imagen;

SDL\_Rect rectImag;

int capacidad; //5kg\*cantidad\_monedas

int cantidadItem;

string codItem; //Indica el tipoVagon a traves del item que lleva

bool detenido;

Pila cajas;

}Vagon;

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

// PRIMITIVAS

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la altura del sprite del vagon.

Vagon: estructura de datos a ser creado.

altoSprite: dato que se asignara.

\*/

void setAltoSprite(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene el alto del sprite del vagon.

Vagon: estructura de datos a ser creado.

\*/

int getAltoSprite(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene el ancho de la celda del vagon.

Vagon: estructura de datos a ser creado.

\*/

int getAnchoCelda(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea el ancho de la celda del vagon.

Vagon: estructura de datos a ser creado.

anchoCelda: dato que se asignara.

\*/

void setAnchoCelda(Vagon &vagon,int anchoCelda);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene el alto de la celda del vagon.

Vagon: estructura de datos a ser creado.

\*/

int getAltoCelda(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene el alto de la celda del vagon.

Vagon: estructura de datos a ser creado.

altoCelda: dato que se asignara.

\*/

void setAltoCelda(Vagon &vagon,int altoCelda);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la ubicacion de la miniatura del vagon.

Vagon: estructura de datos a ser creado.

\*/

char\* getMiniatura(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la ubicacion de la miniatura del vagon.

Vagon: estructura de datos a ser creado.

miniatura: dato a asignar.

\*/

void setMiniatura(Vagon &vagon,char miniatura[]);

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

/\*

pre : el vagon no debe haber sido creado.

post: el vagon queda creado y preparado para ser usado.

Vagon: estructura de datos a ser creado.

\*/

void crearVagon(Vagon &vagon,char miniatura[], int fila,int columna, char direccion[], int anchoCelda, int altoCelda,int capacidad,string item);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la fila donde se encuentra el vagon.

return : fila

\*/

int getFila(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la fila donde se encuentra el vagon.

vagon: estructura de datos a ser usada.

desplazamiento: dato a asignar.

\*/

void setFila(Vagon &vagon,int desplazamiento);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la columna donde se encuentra el vagon.

return : columna

\*/

int getColumna(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la columna donde se encuentra el vagon.

vagon: estructura de datos a ser usada.

desplazamiento: dato a asignar.

\*/

void setColumna(Vagon &vagon,int desplazamiento);

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

/\*

pre : el vagon debe haber sido creado.

post: se dibuja el vagon

vagon: estructura de datos a ser usado.

renderer

\*/

void dibujarVagon(Vagon &vagon,SDL\_Renderer\* renderer,int intervalo);

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

/\*

pre : el vagon debe haber sido creado.

post: se destruye el vagon

vagon: estructura de datos a ser usado.

\*/

void destruirVagon(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la direccion del vagon.

return : direccion

\*/

char\* getDireccion(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la direccion del vagon.

vagon:estructura de datos a ser usado

direccion: datos a cargar

\*/

void setDireccion(Vagon &vagon,char direccion[]);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la capacidad del vagon.

vagon:estructura de datos a ser usado.

return: capacidad del vagon.

\*/

int getCapacidad(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la capacidad del vagon.

vagon:estructura de datos a ser usado

capacidad: capacidad del vagon.

\*/

void setCapacidad(Vagon &vagon, int capacidad);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene la cantidad del item del vagon.

vagon:estructura de datos a ser usado.

return: cantidad del item del vagon.

\*/

int getCantidadItem(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea la cantidad del item del vagon.

vagon:estructura de datos a ser usado.

cantidadItem: cantidad del item del vagon.

\*/

void setCantidadItem(Vagon &vagon, int cantidadItem);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtiene el codigo del item del vagon.

vagon:estructura de datos a ser usado.

return: codigo de item de vagon.

\*/

string getCodItem(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setea el condigo de item del vagon.

vagon:estructura de datos a ser usado.

codItem: codigo de item del vagon.

\*/

void setCodItem(Vagon &vagon, string codItem);

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

/\*

pre : el vagon debe haber sido creado.

post: se obtienen las cajas del vagon.

vagon:estructura de datos a ser usado.

return: pila de cajas del vagon.

\*/

Pila \* getCajas(Vagon &vagon);

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

/\*

pre : el vagon debe haber sido creado.

post: se setean las cajas del vagon.

vagon:estructura de datos a ser usado.

cajas: pila de cajas para el vagon.

\*/

void setCajas(Vagon &vagon, Pila cajas);

#endif