**ARCHIVOS .CPP DEL PROYECTO**

**Integrantes:**

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

**BANDIDO.CPP**

#include "Bandido.h";

#include "Vagon.h";

#include "Caja.h";

#include <SDL.h>

#include <SDL\_image.h>

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

bandido.fila=fila;

bandido.columna=columna;

bandido.imagen=IMG\_LoadTexture(renderer,"img/bandido.png");

bandido.rectImag.y=(bandido.fila\*altoCelda);

bandido.rectImag.x=(bandido.columna\*anchoCelda);

bandido.rectImag.w=anchoCelda;

bandido.rectImag.h=altoCelda;

bandido.codItem=codItem;

bandido.cantidadItem=cantidadItem;

bandido.intervaloFinal=intervaloFinal;

bandido.vidaBandido=vidaBandido;

}

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

int getFila(Bandido \*bandido){

return bandido->fila;

}

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

int getColumna(Bandido \*bandido){

return bandido->columna;

}

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

string getCodItem(Bandido \* bandido){

return bandido->codItem;

}

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

int getCantidadItem(Bandido \*bandido){

return bandido->cantidadItem;

}

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

void setCantidadItem(Bandido &bandido, int cantidadItem){

bandido.cantidadItem = cantidadItem;

}

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

void setFila(Bandido \*bandido,int fila){

bandido->fila=fila;

}

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

void setColumna(Bandido \*bandido,int columna){

bandido->columna=columna;

}

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

void setCodItem(Bandido \*bandido,string codItem){

bandido->codItem=codItem;

}

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

void setCantidadItem(Bandido \*bandido,int cantidadItem){

bandido->cantidadItem=cantidadItem;

}

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

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

SDL\_RenderCopy(renderer,bandido->imagen,NULL,&(bandido->rectImag));

}

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

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

/\* srand(time(NULL));

int i=0;

i=rand()%5;

stringstream aux;

aux <<"img/suelo\_"<<i<<".png";

\*/

bandido->imagen=IMG\_LoadTexture(renderer,"img/suelo\_02.png");

}

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

void destruirBandido(Bandido \*bandido){

SDL\_DestroyTexture(bandido->imagen);

}

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

**CAJA.CPP**

#include "Caja.h"

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

caja.cantidadItem=cantidadItem;

caja.capacidadActual=capacidadActual;

caja.capacidadMaxima=capacidadMaxima;

caja.codItem=codItem;

}

string getCodItem (Caja &caja) {

return caja.codItem;

}

int getCapacidadMaxima(Caja &caja) {

return caja.capacidadMaxima;

}

int getCapacidadActual(Caja &caja) {

return caja.capacidadActual;

}

int getCantidadItem(Caja &caja){

return caja.cantidadItem;

}

void setCantidadItem(Caja &caja, int cantidadItem) {

caja.cantidadItem = cantidadItem;

}

void setCapacidadMaxima(Caja &caja, int capacidadMaxima) {

caja.capacidadMaxima=capacidadMaxima;

}

void setCapacidadActual(Caja &caja, int capacidadActual) {

caja.capacidadActual=capacidadActual;

}

void setCodItem(Caja &caja, string codItem) {

caja.codItem=codItem;

}

void eliminarCaja(Caja & caja){

}

**COLA.CPP**

#include "Cola.h"

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

/\* Funciones auxiliares \*/

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

PtrNodoCola crearNodoCola(PtrDato ptrDato) {

/\* reserva memoria para el nodo y luego completa sus datos \*/

PtrNodoCola ptrAux = new NodoCola;

ptrAux->ptrDato = ptrDato;

ptrAux->sgte = NULL;

return ptrAux;

}

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

void eliminarNodoCola(Cola &cola) {

PtrNodoCola ptrExPrimero;

/\* verifica que la cola no esté vacia y que nodo no sea fin\*/

if (! colaVacia(cola)) {

ptrExPrimero = cola.primero;

cola.primero = colaFrente(cola)->sgte;

// Si el dato es un TDA, acá habría que llamar al destructor.

delete ptrExPrimero;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearCola(Cola &cola) {

cola.primero = finCola();

cola.ultimo = finCola();

}

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

PtrNodoCola finCola(){

return NULL;

}

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

bool colaVacia(Cola &cola){

return (cola.primero == finCola());

}

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

PtrNodoCola colaFrente(Cola &cola){

return cola.primero;

}

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

PtrNodoCola colaFin(Cola &cola){

return cola.ultimo;

}

PtrNodoCola encolar(Cola &cola, PtrDato ptrDato){

/\* crea el nodo \*/

PtrNodoCola ptrNuevoNodo = crearNodoCola(ptrDato);

/\* lo incorpora al final de la cola \*/

if (colaVacia(cola)){

cola.primero = ptrNuevoNodo;

cola.ultimo = ptrNuevoNodo;

}

else{

cola.ultimo->sgte = ptrNuevoNodo;

cola.ultimo = ptrNuevoNodo;

}

return ptrNuevoNodo;

}

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

PtrDato desencolar(Cola &cola){

PtrDato dato;

dato = colaFrente(cola)->ptrDato;

if ((! colaVacia(cola))){

eliminarNodoCola(cola);

}

return dato;

}

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

void eliminarCola(Cola &cola){

while((! colaVacia(cola))){

eliminarNodoCola(cola);

}

}

**COMANDA.CPP**

#include "Comanda.h"

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

using namespace std;

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

comanda.codItem=codItem;

comanda.cantidad=cantidad;

}

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

void destruirComanda(Comanda &comanda){}

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

void setcodItem(Comanda &comanda,string codItem){

comanda.codItem=codItem;

}

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

string getCodItem(Comanda &comanda){

return comanda.codItem;

}

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

void setCantidad(Comanda &comanda,int cantidad){

comanda.cantidad=cantidad;

}

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

int getCantidad(Comanda &comanda){

return comanda.cantidad;

}

**ESTACION.CPP**

#include "Estacion.h"

#include <SDL.h>

#include <SDL\_image.h>

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

estacion.c=c;//coordenada logica x

estacion.f=f;//coordenada logica y

estacion.imagen=IMG\_LoadTexture(renderer,"img/estacion.png");

estacion.rectImag.y=(estacion.f\*altoCelda);//coordenada de dibujo y

estacion.rectImag.x=(estacion.c\*anchoCelda);//coordenada de dibujo x

estacion.rectImag.w=anchoCelda;//ancho

estacion.rectImag.h=altoCelda;//alto

}

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

int getFila(Estacion \*estacion){

return estacion->f;

}

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

void setFila(Estacion &estacion, int fila){

estacion.f = fila;

}

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

int getColumna(Estacion \*estacion){

return estacion->c;

}

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

void setColumna(Estacion &estacion, int columna){

estacion.c = columna;

}

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

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

SDL\_RenderCopy(renderer,estacion->imagen,NULL,&(estacion->rectImag));

}

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

int venderVagon(Estacion &estacion, int cantidadMonedas){

return CAPACIDAD\_VAGON \* cantidadMonedas;

}

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

void destruirEstacion(Estacion &estacion){

SDL\_DestroyTexture(estacion.imagen);

}

**GAME.CPP**

#include <string.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <SDL.h>

#include "Game.h"

#include "Celda.h"

#include "Terreno.h"

#include "Bandido.h"

#include "Lista.h"

#include "Vagon.h"

#include "Moneda.h"

#include <ctime>

#include <stdlib.h>

#include "Caja.h"

#include "Mina.h"

#include <SDL\_ttf.h>

#include <string>

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

game.gameover=false;

game.intervalo=0;

game.contadorSegundo=0;

game.contadorMonedas=0;

game.anchoCelda=anchoCelda;

game.altoCelda=altoCelda;

game.fila=fila;

game.columna=columna;

game.terreno= new Terreno\*[game.columna];

for (int f = 0; f < game.fila; f++){

for(int c=0;c<game.columna;c++){

game.terreno[f] = new Terreno[c];

}

}

strcpy(game.direccion,"direccion");

crearLista(game.minas,NULL);

crearLista(game.comanda,NULL);

crearParametros(game.parametros);

}

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

int getAnchoCelda(Game &game){

return game.anchoCelda;

}

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

int getAltoCelda(Game &game){

return game.altoCelda;

}

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

int getIntervalo(Game &game){

return game.intervalo;

}

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

void setIntervalo(Game &game, int intervalo){

game.intervalo=intervalo;

}

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

int getContadorSegundo(Game &game){

return game.contadorSegundo;

}

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

void setContadorSegundo(Game &game, int contadorSegundo){

game.contadorSegundo=contadorSegundo;

}

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

int getContadorMonedas(Game &game){

return game.contadorMonedas;

}

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

void setContadorMonedas(Game &game, int contadorMonedas){

game.contadorMonedas=contadorMonedas;

}

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

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

srand(time(0));

for(int f=0;f<game.fila;f++){

for(int c=0;c<game.columna;c++){

Terreno terreno;

char miniatura [2];

itoa((rand()%6),miniatura,10);

crearTerreno(terreno,f,c,getAnchoCelda(game),getAltoCelda(game),renderer,miniatura);

game.terreno[f][c]=terreno;

}

}

}

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

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

return game.terreno;

}

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

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

ifstream entrada("Minas.txt");

string strposx,strposy,strcoditem,strip,strseq1,strseq2,strseq3,strseq4,strseq5;

int posX,posY,ip,seq1,seq2,seq3,seq4,seq5;

string cadena;

while(!entrada.eof()){

getline(entrada,cadena,'\n');

if(cadena != ""){

stringstream cadena2(cadena);

Mina \*mina = new Mina;

getline(cadena2,strposx,';');

posX=atoi(strposx.c\_str());

getline(cadena2,strposy,';');

posY=atoi(strposy.c\_str());

getline(cadena2,strcoditem,';');

getline(cadena2,strip,';');

ip=atoi(strip.c\_str());

getline(cadena2,strseq1,';');

seq1=atoi(strseq1.c\_str());

getline(cadena2,strseq2,';');

seq2=atoi(strseq2.c\_str());

getline(cadena2,strseq3,';');

seq3=atoi(strseq3.c\_str());

getline(cadena2,strseq4,';');

seq4=atoi(strseq4.c\_str());

getline(cadena2,strseq5,';');

seq5=atoi(strseq5.c\_str());

if(ip>atoi(getClaveIP(game.parametros).c\_str())){ //el intervalo de la mina no puede ser mayor al de parametros

ip = atoi(getClaveIP(game.parametros).c\_str());

}

int secuencia[5] = {seq1,seq2,seq3,seq4,seq5};

crearMina(\*mina, renderer, posY, posX, getAnchoCelda(game), getAltoCelda(game), ip, secuencia,strcoditem);

ubicarMina(game, mina);

adicionarFinal(game.minas,mina);

cadena2.clear();

}

}

entrada.close();

fflush(stdin);

}

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

Lista getMinas(Game &game){

return game.minas;

}

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

void leerComanda(Game &game){

ifstream entrada("Comanda.txt");

string codItem,strcantidad;

int cantidad;

string cadena;

while(!entrada.eof()){

getline(entrada,cadena,'\n');

if(cadena != ""){

stringstream cadena2(cadena);

Comanda \*comanda = new Comanda;

getline(cadena2,codItem,';');

getline(cadena2,strcantidad,';');

cantidad=atoi(strcantidad.c\_str());

crearComanda(\*comanda,codItem,cantidad);

adicionarFinal(game.comanda,comanda);

cadena2.clear();

}

}

entrada.close();

fflush(stdin);

}

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

Lista getComanda(Game &game){

return game.comanda;

}

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

void leerParametros(Game &game){

ifstream entrada("Parametros.txt");

string clave,valor;

if(entrada.fail()){

cout<<"No se pudo abrir el archivo Parametros.txt ";

exit(EXIT\_FAILURE);

}

while(!entrada.eof()){

getline(entrada,clave,'=');

getline(entrada,valor);

setParametros(game,clave,valor);

}

entrada.close();

fflush(stdin);

}

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

if (clave == "s" || clave == "S")

game.parametros.claveS = valor;

else if (clave == "p" || clave == "P")

game.parametros.claveP = valor;

else if (clave == "a" || clave == "A")

game.parametros.claveA = valor;

else if (clave == "posXE")

game.parametros.claveposXE = valor;

else if (clave == "posYE")

game.parametros.claveposYE = valor;

else if (clave == "im" || clave == "IM")

game.parametros.claveIM = valor;

else if (clave == "vm" || clave == "VM")

game.parametros.claveVM = valor;

else if (clave == "ib" || clave == "IB")

game.parametros.claveIB = valor;

else if (clave == "vb" || clave == "VB")

game.parametros.claveVB =valor;

else if (clave == "ip" || clave == "IP")

game.parametros.claveIP =valor;

}

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

Parametros getParametros(Game &game){

return game.parametros;

}

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

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

for(int f=0;f<game.fila;f++){

for(int c=0;c<game.columna;c++){

dibujarTerreno(game.terreno[f][c], renderer);

}

}

}

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

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

for(int f=0;f<game.fila;f++){

for(int c=0;c<game.columna;c++){

dibujarEntidad(game.terreno[f][c], renderer, game.intervalo);

}

}

}

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

bool getGameOver(Game &game){

return game.gameover;

}

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

void setGameOver(Game &game, bool flag){

game.gameover=flag;

}

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

void ubicarVagon(Game &game,PtrNodoLista ptrNodo){

game.terreno[getFila(\*(Vagon\*)ptrNodo->ptrDato)][getColumna(\*(Vagon\*)ptrNodo->ptrDato)].ptrNodoVagon=ptrNodo;//celda.ptrNosoVagon=ptrNodo

}

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

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

game.terreno[getFila(bandido)][getColumna(bandido)].ptrBandido=bandido;

}

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

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

game.terreno[getFila(mina)][getColumna(mina)].ptrMina=mina;

}

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

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

game.terreno[getFila(moneda)][getColumna(moneda)].ptrMoneda=moneda;

}

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

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

game.terreno[getFila(estacion)][getColumna(estacion)].ptrEstacion=estacion;

}

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

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

strcpy(game.direccion,direccion);

}

char\* getDireccion(Game &game){

return game.direccion;

}

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

int getColumna(Game &game){

return game.columna;

}

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

int getFila(Game &game){

return game.fila;

}

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

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

int desplazamientoHorizontal=0;

int desplazamientoVertical=0;

Vagon \*vagon = (Vagon\*)primero(locomotora)->ptrDato;

PtrNodoLista nodo = primero(locomotora);

if(strcmp(getDireccion(\*vagon),"aba")==0)desplazamientoVertical=1;

if(strcmp(getDireccion(\*vagon),"arr")==0)desplazamientoVertical=-1;

if(strcmp(getDireccion(\*vagon),"der")==0)desplazamientoHorizontal=1;

if(strcmp(getDireccion(\*vagon),"izq")==0)desplazamientoHorizontal=-1;

int columna=getColumna(\*vagon)+desplazamientoHorizontal;

int fila=getFila(\*vagon)+desplazamientoVertical;

if(game.terreno[fila][columna].ptrMina!=NULL && !(vagon->detenido)){

recolectarCajas(\*(game.terreno[fila][columna].ptrMina), locomotora);

while(nodo!=finLista()){

vagon = (Vagon\*)nodo->ptrDato;

vagon->detenido = true;

nodo = siguiente(locomotora, nodo);

}

}

else if(game.terreno[fila][columna].ptrEstacion!=NULL && !(vagon->detenido)){

agregarVagon(game, locomotora, \*(game.terreno[fila][columna].ptrEstacion));

while(nodo!=finLista()){

vagon = (Vagon\*)nodo->ptrDato;

vagon->detenido = true;

nodo = siguiente(locomotora, nodo);

}

}

else if(game.terreno[fila][columna].ptrMoneda!=NULL && !(vagon->detenido)){

reemplazarMoneda(game.terreno[fila][columna].ptrMoneda,renderer);

PtrNodoLista ptrNodo = primero(monedas);

PtrNodoLista ptrEliminar;

while(ptrNodo!=finLista()){

if((Moneda\*)ptrNodo->ptrDato == game.terreno[fila][columna].ptrMoneda){

ptrEliminar = ptrNodo;

}

ptrNodo = siguiente(monedas, ptrNodo);

}

eliminarNodo(monedas,ptrEliminar);

destruirMoneda(game.terreno[fila][columna].ptrMoneda);

game.terreno[fila][columna].ptrMoneda=NULL;

game.contadorMonedas=game.contadorMonedas+1;

}

if(game.terreno[fila][columna].ptrMina==NULL && game.terreno[fila][columna].ptrEstacion==NULL && vagon->detenido){

while(nodo!=finLista()){

vagon = (Vagon\*)nodo->ptrDato;

vagon->detenido = false;

nodo = siguiente(locomotora, nodo);

}

}

}

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

void actualizarMinas(Game &game){

PtrNodoLista ptrMina = primero(game.minas);

while(ptrMina!=finLista()){

Mina \* mina = (Mina\*)ptrMina->ptrDato;

if(getIntervarloActual(\*mina)==getIP(\*mina)){

Caja \*caja = new Caja;

int \*secuencia = getSecuencia(\*mina);

crearCaja(\*caja,secuencia[getSecuenciaActual(\*mina)],secuencia[getSecuenciaActual(\*mina)],

secuencia[getSecuenciaActual(\*mina)],getCodItem(mina));

encolar(mina->cajas,caja);

setIntervaloActual(\*mina,0);

if(getSecuenciaActual(\*mina)==4){

setSecuenciaActual(\*mina,0);

}

else{

setSecuenciaActual(\*mina, getSecuenciaActual(\*mina)+1);

}

}

else{

setIntervaloActual(\*mina, getIntervarloActual(\*mina)+1);

}

ptrMina = siguiente(game.minas,ptrMina);

}

}

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

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

int oro=0; int plata=0; int bronce=0; int platino=0; int carbon=0; int roca=0;

int oroMax,plataMax,bronceMax,platinoMax,carbonMax,rocaMax;

PtrNodoLista ptrVagon = primero(locomotora);

while(ptrVagon!=finLista()){

Vagon \* vagon = (Vagon\*)ptrVagon->ptrDato;

if(getCodItem(\*vagon)=="oro"){

oro += getCantidadItem(\*vagon);

}

else if(getCodItem(\*vagon)=="plata"){

plata += getCantidadItem(\*vagon);

}

else if(getCodItem(\*vagon)=="bronce"){

bronce += getCantidadItem(\*vagon);

}

else if(getCodItem(\*vagon)=="platino"){

platino += getCantidadItem(\*vagon);

}

else if(getCodItem(\*vagon)=="carbon"){

carbon += getCantidadItem(\*vagon);

}

else if(getCodItem(\*vagon)=="roca"){

roca += getCantidadItem(\*vagon);

}

ptrVagon = siguiente(locomotora, ptrVagon);

}

PtrNodoLista ptrComanda = primero(game.comanda);

while(ptrComanda!=finLista()){

Comanda \* comanda = (Comanda\*)ptrComanda->ptrDato;

if(getCodItem(\*comanda)=="oro"){oroMax = getCantidad(\*comanda);}

else if(getCodItem(\*comanda)=="plata"){plataMax = getCantidad(\*comanda);}

else if(getCodItem(\*comanda)=="bronce"){bronceMax = getCantidad(\*comanda);}

else if(getCodItem(\*comanda)=="platino"){platinoMax = getCantidad(\*comanda);}

else if(getCodItem(\*comanda)=="roca"){rocaMax = getCantidad(\*comanda);}

else if(getCodItem(\*comanda)=="carbon"){carbonMax = getCantidad(\*comanda);}

ptrComanda = siguiente(game.comanda,ptrComanda);

}

stringstream ss;

ss << "Oro: " << oro << "/" << oroMax <<

" Plata: "<< plata << "/" << plataMax <<

" Bronce: " << bronce << "/" << bronceMax <<

" Platino: " << platino << "/" << platinoMax <<

" Roca: " << roca << "/" << rocaMax <<

" Carbon: " << carbon << "/" << carbonMax<<

" Monedas:"<<game.contadorMonedas;

string temp = ss.str();

char const \* puntuacion = temp.c\_str();

if(oro==oroMax && plata==plataMax && bronce==bronceMax && platino==platinoMax && roca==rocaMax && carbon==carbonMax){

SDL\_ShowSimpleMessageBox(SDL\_MESSAGEBOX\_INFORMATION,"Fin del juego", "Ha completado todos los objetivos", NULL);

setGameOver(game,true);

}

TTF\_Font\* arial = TTF\_OpenFont("arial.ttf", 16);

SDL\_Color blanco = {255, 255, 255};

SDL\_Surface\* surfaceMessage = TTF\_RenderText\_Blended(arial,puntuacion,blanco);

SDL\_Texture\* Message = SDL\_CreateTextureFromSurface(renderer, surfaceMessage);

int texW = 0;

int texH = 0;

TTF\_SizeText(arial, puntuacion, &texW, &texH);

SDL\_Rect Message\_rect = {0, 0, texW, texH};

SDL\_RenderCopy(renderer, Message, NULL, &Message\_rect);

TTF\_CloseFont(arial);

SDL\_DestroyTexture(Message);

SDL\_FreeSurface(surfaceMessage);

}

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

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

Moneda \*moneda= new Moneda;

int intervaloFinal=game.contadorSegundo+VM;

bool is=false;

srand(time(NULL));

while(is!=true){

int fila = rand()%getFila(game); //Definir los drops para que no colisiones con los sprites

int columna=rand()%getColumna(game);

if(game.terreno[fila][columna].ptrEstacion==NULL && game.terreno[fila][columna].ptrMoneda==NULL &&

game.terreno[fila][columna].ptrMina==NULL && game.terreno[fila][columna].ptrBandido==NULL && game.terreno[fila][columna].ptrNodoVagon==NULL){

crearMoneda(\*moneda,renderer,fila,columna,game.anchoCelda,game.altoCelda,VM,intervaloFinal);

ubicarMoneda(game,moneda);

adicionarFinal(monedas,moneda);

is=true;

}

}

}

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

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

PtrNodoLista ptrMoneda= primero(monedas);

while(ptrMoneda!=finLista()){

Moneda \*moneda= (Moneda\*) ptrMoneda->ptrDato;

if(getContadorSegundo(game)==moneda->intervaloFinal){

reemplazarMoneda(moneda,renderer); //Reemplazar por suelo original

destruirMoneda(moneda);

game.terreno[moneda->fila][moneda->columna].ptrMoneda=NULL;

eliminarNodo(monedas,ptrMoneda);

}

ptrMoneda=siguiente(monedas,ptrMoneda);

}

}

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

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

Bandido \*bandido= new Bandido;

int intervaloFinal=game.contadorSegundo+VB;

string codItem;

bool is=false;

srand(time(NULL));

while(is!=true){

int fila =rand()%getFila(game); //Definir los drops para que no colisiones con los sprites

int columna=rand()%getColumna(game);

//Me fijo que la fila y columna dada de forma random no este ya ocupada, si esta ocupada genero otra fila y columna

if(game.terreno[fila][columna].ptrEstacion==NULL && game.terreno[fila][columna].ptrMoneda==NULL &&

game.terreno[fila][columna].ptrMina==NULL && game.terreno[fila][columna].ptrBandido==NULL && game.terreno[fila][columna].ptrNodoVagon==NULL){

int P= numeroStringRand(game.parametros.claveP); //cantidad del item, pudiendo ser un número entero entre 1 y el máximo permitido (P).

int opcion=numeroStringRand("6"); //numero random entre 1 y 6 para elegir un codItem

switch (opcion) {

case 1:

codItem="oro";

break;

case 2:

codItem="plata";

break;

case 3:

codItem="bronce";

break;

case 4:

codItem="platino";

break;

case 5:

codItem="roca";

break;

case 6:

codItem="carbon";

break;

}

crearBandido(\*bandido,renderer,fila,columna,game.anchoCelda,game.altoCelda,codItem,P,VB,intervaloFinal);

ubicarBandido(game,bandido);

adicionarFinal(bandidos,bandido);

is=true;

}

}

}

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

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

PtrNodoLista ptrBandido= primero(bandidos);

while(ptrBandido!=finLista()){

Bandido \*bandido= (Bandido\*) ptrBandido->ptrDato;

if(getContadorSegundo(game)==bandido->intervaloFinal){

reemplazarBandido(bandido,renderer); //Reemplazar por suelo original

destruirBandido(bandido);

game.terreno[bandido->fila][bandido->columna].ptrBandido=NULL;

eliminarNodo(bandidos,ptrBandido);

}

ptrBandido=siguiente(bandidos,ptrBandido);

}

}

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

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

for(int f=0;f<game.fila;f++){

for(int c=0;c<game.columna;c++){

destruirTerreno(game.terreno[f][c]);

}

}

eliminarLista(locomotora);

eliminarLista(game.minas);

eliminarLista(bandidos);

eliminarLista(monedas);

TTF\_Quit();

IMG\_Quit();

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

SDL\_Quit();

}

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

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

if(getContadorMonedas(game)!=0){

Vagon \* vagon = (Vagon\*)ultimo(locomotora)->ptrDato;

int posX, posY;

char \* direccion = new char;

nuevaPosicionVagon(game, vagon, posX, posY, direccion);

if(strcmp(direccion,"nada")!=0){

Vagon \* nuevoVagon = new Vagon;

crearVagon(\*nuevoVagon,"c2", posX, posY,direccion,getAnchoCelda(\*vagon), getAltoCelda(\*vagon), 5\*getContadorMonedas(game),"nada");

ubicarVagon(game,adicionarFinal(locomotora,nuevoVagon));

setContadorMonedas(game, 0);

}

}

}

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

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

bool arribaNull = false; bool abajoNull = false; bool derechaNull = false; bool izquierdaNull = false;

int fila = getFila(\*ultimo);

int columna = getColumna(\*ultimo);

if(game.terreno[fila-1][columna].ptrMina == NULL &&

game.terreno[fila-1][columna].ptrEstacion ==NULL &&

game.terreno[fila-1][columna].ptrNodoVagon == NULL &&

fila!=0) arribaNull = true;

if(game.terreno[fila+1][columna].ptrMina == NULL &&

game.terreno[fila+1][columna].ptrEstacion ==NULL &&

game.terreno[fila+1][columna].ptrNodoVagon == NULL &&

fila!=getFila(game)-1) abajoNull = true;

if(game.terreno[fila][columna-1].ptrMina == NULL &&

game.terreno[fila][columna-1].ptrEstacion ==NULL &&

game.terreno[fila][columna-1].ptrNodoVagon == NULL &&

columna!=0) izquierdaNull = true;

if(game.terreno[fila][columna+1].ptrMina == NULL &&

game.terreno[fila][columna+1].ptrEstacion ==NULL &&

game.terreno[fila][columna+1].ptrNodoVagon == NULL &&

columna!=getColumna(game)-1) derechaNull = true;

strcpy(direccion,"nada");

if(strcmp(getDireccion(\*ultimo),"arr")==0){

if(abajoNull){

strcpy(direccion,"arr");

posX = fila+1;

posY = columna;

}

else if(derechaNull){

strcpy(direccion,"izq");

posX = fila;

posY = columna+1;

}

else if (izquierdaNull){

strcpy(direccion,"der");

posX = fila;

posY = columna-1;

}

}

else if(strcmp(getDireccion(\*ultimo),"aba")==0){

if(arribaNull){

strcpy(direccion,"aba");

posX = fila-1;

posY = columna;

}

else if(derechaNull){

strcpy(direccion,"izq");

posX = fila;

posY = columna+1;

}

else if (izquierdaNull){

strcpy(direccion,"der");

posX = fila;

posY = columna-1;

}

}

else if(strcmp(getDireccion(\*ultimo),"der")==0){

if(izquierdaNull){

strcpy(direccion,"der");

posX = fila;

posY = columna-1;

}

else if(arribaNull){

strcpy(direccion,"aba");

posX = fila-1;

posY = columna;

}

else if (abajoNull){

strcpy(direccion,"arr");

posX = fila+1;

posY = columna;

}

}

else if(strcmp(getDireccion(\*ultimo),"izq")==0){

if(derechaNull){

strcpy(direccion,"izq");

posX = fila;

posY = columna+1;

}

else if(arribaNull){

strcpy(direccion,"aba");

posX = fila-1;

posY = columna;

}

else if (abajoNull){

strcpy(direccion,"arr");

posX = fila+1;

posY = columna;

}

}

}

int numeroStringRand(string numero){

srand(time(NULL));

int i=0;

int j = atoi(numero.c\_str());

i=1+rand()%j;

return i;

}

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

PtrNodoLista ptrNodoVagon= primero(locomotora);

while(ptrNodoVagon!=finLista()){

Vagon \* vagon = (Vagon\*)ptrNodoVagon->ptrDato;

int area = atoi(getClaveA(game.parametros).c\_str());

int minI = 0;

int minJ = 0;

int maxI = getFila(game)-1;

int maxJ = getColumna(game)-1;

if((getFila(\*vagon)-area)>0){

minI = getFila(\*vagon) - area;

}

if((getColumna(\*vagon)-area)>0){

minJ = getColumna(\*vagon) - area;

}

if((getFila(\*vagon)+area)<getFila(game)-1){

maxI = getFila(\*vagon)+ area;

}

if((getColumna(\*vagon)+area)<getColumna(game)-1){

maxJ = getColumna(\*vagon) + area;

}

for(int i = minI;i<=maxI;i++){

for(int j = minJ;j<=maxJ;j++){

if(game.terreno[i][j].ptrBandido!=NULL){

pedirItem(game,locomotora,\*(game.terreno[i][j].ptrBandido));

cout<<"bandido pideItem";

if(!listaVacia(locomotora)){

PtrNodoLista ptrBandido = primero(bandidos);

PtrNodoLista ptrEliminar;

while(ptrBandido!=finLista()){

if((Bandido\*)ptrBandido->ptrDato == game.terreno[i][j].ptrBandido){

ptrEliminar = ptrBandido;

}

ptrBandido = siguiente(bandidos, ptrBandido);

}

eliminarNodo(bandidos,ptrEliminar);

destruirBandido(game.terreno[i][j].ptrBandido);

game.terreno[i][j].ptrBandido=NULL;

}

}

}

}

ptrNodoVagon = siguiente(locomotora,ptrNodoVagon);

}

}

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

PtrNodoLista ptrNodoVagon = primero(locomotora);

while(ptrNodoVagon!=finLista() && getCantidadItem(&bandido)>0){

Vagon \* vagon = (Vagon\*)ptrNodoVagon->ptrDato;

if(getCodItem(\*vagon)==getCodItem(&bandido)){

Pila \* cajas = getCajas(\*vagon);

while(getCantidadItem(&bandido)!=0 && !pilaVacia(\*cajas)){

Caja \* caja = (Caja\*)cima(\*cajas)->ptrDato;

if((getCapacidadActual(\*caja)-getCantidadItem(&bandido))<=0){

setCantidadItem(bandido, getCantidadItem(&bandido)-getCapacidadActual(\*caja));

eliminarCaja(\*caja);

sacar(\*cajas);

}

else{

setCapacidadActual(\*caja, getCapacidadActual(\*caja)-getCantidadItem(&bandido));

setCantidadItem(\*vagon, getCantidadItem(\*vagon)-getCantidadItem(&bandido));

setCantidadItem(bandido, 0);

}

}

}

ptrNodoVagon = siguiente(locomotora,ptrNodoVagon);

}

if(getCantidadItem(&bandido)!=0){

Vagon \* vagon = (Vagon\*)ultimo(locomotora)->ptrDato;

game.terreno[getFila(\*vagon)][getColumna(\*vagon)].ptrNodoVagon = NULL;

eliminarNodoUltimo(locomotora);

if(listaVacia(locomotora)){

SDL\_ShowSimpleMessageBox(SDL\_MESSAGEBOX\_INFORMATION,"Perdiste", "El bandido destruyo la locomotora", NULL);

setGameOver(game,true);

}

}

}

bool evaluarColisiones(Game &game,PtrNodoLista ptrNodo){

bool colision=true;

int desplazamientoHorizontal=0;

int desplazamientoVertical=0;

Vagon \*vagon = (Vagon\*)ptrNodo->ptrDato;

if(strcmp(getDireccion(\*vagon),"aba")==0)desplazamientoVertical=1;

if(strcmp(getDireccion(\*vagon),"arr")==0)desplazamientoVertical=-1;

if(strcmp(getDireccion(\*vagon),"der")==0)desplazamientoHorizontal=1;

if(strcmp(getDireccion(\*vagon),"izq")==0)desplazamientoHorizontal=-1;

int c=getColumna(\*vagon)+desplazamientoHorizontal;

int f=getFila(\*vagon)+desplazamientoVertical;

if((c>=0 && c<getColumna(game)) && (f>=0 && f<getFila(game))){

Terreno terrenoAux =getTerreno(game)[f][c];

if(terrenoAux.ptrNodoVagon==NULL){

colision=false;

}else {

SDL\_ShowSimpleMessageBox(SDL\_MESSAGEBOX\_INFORMATION,"Perdiste", "Chocamos con un vagon", NULL);

setGameOver(game,true);

}

}else{

SDL\_ShowSimpleMessageBox(SDL\_MESSAGEBOX\_INFORMATION,"Perdiste", "Nos salimos del terreno", NULL);

setGameOver(game,true);

}

return colision;

}

void cambiarColumna(Game &game,PtrNodoLista ptrNodo){

if(ptrNodo!=NULL){

int desplazamiento=0;

Vagon \*vagon = (Vagon\*)ptrNodo->ptrDato;

if(strcmp(getDireccion(\*vagon),"der")==0)desplazamiento=1;

if(strcmp(getDireccion(\*vagon),"izq")==0)desplazamiento=-1;

if(desplazamiento!=0){

Terreno \*\*terrenoAux=getTerreno(game);

terrenoAux[getFila(\*vagon)][getColumna(\*vagon)].ptrNodoVagon=NULL;

terrenoAux[getFila(\*vagon)][getColumna(\*vagon)+desplazamiento].ptrNodoVagon=ptrNodo;

setColumna(\*vagon,desplazamiento);

}

cambiarColumna(game,ptrNodo->sgte);

}

}

void cambiarFila(Game &game, PtrNodoLista ptrNodo){

if(ptrNodo!=NULL){

int desplazamiento=0;

Vagon \*vagon = (Vagon\*)ptrNodo->ptrDato;

if(strcmp(getDireccion(\*vagon),"aba")==0)desplazamiento=1;

if(strcmp(getDireccion(\*vagon),"arr")==0)desplazamiento=-1;

if(desplazamiento!=0){

Terreno \*\*terrenoAux=getTerreno(game);

if(ptrNodo->sgte==finLista()){

terrenoAux[getFila(\*vagon)][getColumna(\*vagon)].ptrNodoVagon=NULL;

}

terrenoAux[getFila(\*vagon)+desplazamiento][getColumna(\*vagon)].ptrNodoVagon=ptrNodo;

setFila(\*vagon,desplazamiento);

}

cambiarFila(game,ptrNodo->sgte);

}

}

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

PtrNodoLista ptrNodo = primero(locomotora);

char direccion[4];

strcpy(direccion, getDireccion(game));

while(ptrNodo!=finLista()){

Vagon \*vagon = (Vagon\*)ptrNodo->ptrDato;

char dirAnterior[4];

strcpy(dirAnterior, getDireccion(\*vagon));

if(

((strcmp(dirAnterior,"izq")==0 && strcmp(direccion,"der")!=0) ||

(strcmp(dirAnterior,"der")==0 && strcmp(direccion,"izq")!=0) ||

(strcmp(dirAnterior,"arr")==0 && strcmp(direccion,"aba")!=0) ||

(strcmp(dirAnterior,"aba")==0 && strcmp(direccion,"arr")!=0)) &&

!(vagon->detenido && vagon!=(Vagon\*)primero(locomotora)->ptrDato)

){

setDireccion(\*vagon,direccion);//cargo la nueva dirección

}

strcpy(direccion,dirAnterior);

ptrNodo = siguiente(locomotora, ptrNodo);

}

}

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

if(SDL\_PollEvent(&event)){

switch(event.type){

case SDL\_QUIT:

setGameOver(game, true);

break;

case SDL\_KEYDOWN:

if(keys[SDL\_SCANCODE\_ESCAPE]){

setGameOver(game, true);

}

if(keys[SDL\_SCANCODE\_LEFT]){

setDireccion(game,"izq");

}

if(keys[SDL\_SCANCODE\_RIGHT]){

setDireccion(game,"der");

}

if(keys[SDL\_SCANCODE\_UP]){

setDireccion(game,"arr");

}

if(keys[SDL\_SCANCODE\_DOWN]){

setDireccion(game,"aba");

}

if(keys[SDL\_SCANCODE\_SPACE]){

}

break;

}

}

}

**LISTA.CPP**

#include "Lista.h"

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

/\* Implementación de Primitivas \*/

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

void crearLista(Lista &lista, PFComparacion funcComparacion) {

lista.primero = finLista();

lista.compara = funcComparacion;

}

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

bool listaVacia(Lista &lista) {

return (primero(lista) == finLista());

}

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

PtrNodoLista finLista() {

return NULL;

}

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

PtrNodoLista primero(Lista &lista) {

return lista.primero;

}

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

PtrNodoLista siguiente(Lista &lista, PtrNodoLista ptrNodo) {

/\* verifica si la lista está vacia o si ptrNodo es el último \*/

if ((! listaVacia(lista)) && (ptrNodo != finLista()))

return ptrNodo->sgte;

else

return finLista();

}

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

PtrNodoLista anterior(Lista &lista, PtrNodoLista ptrNodo) {

PtrNodoLista ptrPrevio = finLista();

PtrNodoLista ptrCursor = primero(lista);

while (( ptrCursor != finLista()) && (ptrCursor != ptrNodo)) {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

return ptrPrevio;

}

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

PtrNodoLista ultimo(Lista &lista) {

/\* el último nodo de la lista es el anterior al finLista() \*/

return anterior(lista,finLista());

}

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

PtrNodoLista crearNodoLista(PtrDato ptrDato) {

/\* reserva memoria para el nodo y luego completa sus datos \*/

PtrNodoLista ptrAux = new NodoLista;

ptrAux->ptrDato = ptrDato;

ptrAux->sgte = finLista();

return ptrAux;

}

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

PtrNodoLista adicionarPrincipio(Lista &lista, PtrDato ptrDato) {

/\* crea el nodo \*/

PtrNodoLista ptrNuevoNodo = crearNodoLista(ptrDato);

/\* lo incorpora al principio de la lista \*/

ptrNuevoNodo->sgte = lista.primero;

lista.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

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

PtrNodoLista ptrNuevoNodo = finLista();

/\* si la lista está vacia se adiciona la principio \*/

if (listaVacia(lista))

ptrNuevoNodo = adicionarPrincipio(lista,ptrDato);

else {

if (ptrNodo != finLista()) {

/\* crea el nodo y lo intercala en la lista \*/

ptrNuevoNodo = crearNodoLista(ptrDato);

ptrNuevoNodo->sgte = ptrNodo->sgte;

ptrNodo->sgte = ptrNuevoNodo;

}

}

return ptrNuevoNodo;

}

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

PtrNodoLista adicionarFinal(Lista &lista, PtrDato ptrDato) {

/\* adiciona el dato después del último nodo de la lista \*/

return adicionarDespues(lista,ptrDato,ultimo(lista));

}

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

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

PtrNodoLista ptrNuevoNodo = finLista();

if (! listaVacia(lista)) {

if (ptrNodo != primero(lista))

ptrNuevoNodo = adicionarDespues(lista,ptrDato,anterior(lista,ptrNodo));

else

ptrNuevoNodo = adicionarPrincipio(lista,ptrDato);

}

return ptrNuevoNodo;

}

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

void eliminarNodo(Lista &lista, PtrNodoLista ptrNodo) {

PtrNodoLista ptrPrevio;

/\* verifica que la lista no esté vacia y que nodo no sea fin\*/

if ((! listaVacia(lista)) && (ptrNodo != finLista())) {

if (ptrNodo == primero(lista))

lista.primero = siguiente(lista,primero(lista));

else {

ptrPrevio = anterior( lista , ptrNodo );

ptrPrevio->sgte = ptrNodo->sgte;

}

// Si el dato es un TDA, acá habría que llamar al destructor.

delete ptrNodo;

}

}

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

void eliminarNodoPrimero(Lista &lista) {

if (! listaVacia(lista))

eliminarNodo(lista,primero(lista));

}

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

void eliminarNodoUltimo(Lista &lista) {

if (! listaVacia(lista))

eliminarNodo(lista,ultimo(lista));

}

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

void eliminarLista(Lista &lista) {

/\* retira uno a uno los nodos de la lista \*/

while (! listaVacia(lista))

eliminarNodo(lista,primero(lista));

}

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

PtrNodoLista localizarDato(Lista &lista, PtrDato ptrDato) {

bool encontrado = false;

PtrDato ptrDatoCursor;

PtrNodoLista ptrCursor = primero(lista);

/\* recorre los nodos hasta llegar al último o hasta

encontrar el nodo buscado \*/

while ((ptrCursor != finLista()) && (! encontrado)) {

/\* obtiene el dato del nodo y lo compara \*/

ptrDatoCursor = ptrCursor->ptrDato;

if (lista.compara(ptrDatoCursor,ptrDato) == IGUAL)

encontrado = true;

else

ptrCursor = siguiente(lista,ptrCursor);

}

/\* si no lo encontró devuelve fin \*/

if (! encontrado)

ptrCursor = finLista();

return ptrCursor;

}

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

void eliminarDato(Lista &lista, PtrDato ptrDato) {

/\* localiza el dato y luego lo elimina \*/

PtrNodoLista ptrNodo = localizarDato(lista,ptrDato);

if (ptrNodo != finLista())

eliminarNodo(lista,ptrNodo);

}

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

PtrNodoLista insertarDato(Lista &lista, PtrDato ptrDato) {

PtrNodoLista ptrPrevio = primero(lista);

PtrNodoLista ptrCursor = primero(lista);

PtrNodoLista ptrNuevoNodo;

PtrDato ptrDatoCursor;

bool ubicado = false;

/\* recorre la lista buscando el lugar de la inserción \*/

while ((ptrCursor != finLista()) && (! ubicado)) {

ptrDatoCursor= ptrCursor->ptrDato;

if (lista.compara(ptrDatoCursor,ptrDato) == MAYOR)

ubicado = true;

else {

ptrPrevio = ptrCursor;

ptrCursor = siguiente(lista,ptrCursor);

}

}

if (ptrCursor == primero(lista))

ptrNuevoNodo = adicionarPrincipio(lista,ptrDato);

else

ptrNuevoNodo = adicionarDespues(lista,ptrDato,ptrPrevio);

return ptrNuevoNodo;

}

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

void reordenar(Lista &lista) {

Lista temp = lista;

PtrNodoLista ptrCursor = primero(temp);

crearLista(lista, temp.compara);

while ( ptrCursor != finLista() ) {

PtrDato ptrDato;

ptrDato = ptrCursor ->ptrDato;

insertarDato( lista, ptrDato );

eliminarNodo( temp, ptrCursor );

ptrCursor = primero(temp);

}

eliminarLista( temp );

}

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

int longitud(Lista &lista){

PtrNodoLista ptrCursor = primero(lista);

int longitud = 0;

while ( ptrCursor != finLista() ) {

longitud++;

ptrCursor = siguiente( lista, ptrCursor);

}

return longitud;

}

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

**MINA.CPP**

#include "Mina.h"

#include "Vagon.h"

#include "Caja.h"

#include <SDL.h>

#include <SDL\_image.h>

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

// IMPLEMENTACION DE PRIMITIVAS

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

void crearMina(Mina &mina,SDL\_Renderer\* renderer, int f,int c, int anchoCelda, int altoCelda, int ip, int secuencia[5], string codItem){

mina.c=c;//coordenada logica x

mina.f=f;//coordenada logica y

mina.ip=ip;//intervalo de produccion

mina.codItem=codItem;//item de la mina

for(int i=0;i<5;i++){

mina.secuencia[i] = secuencia[i];

}

mina.intervaloActual = 0;

mina.imagen=IMG\_LoadTexture(renderer,"img/mina.png");

mina.rectImag.y=(mina.f\*altoCelda);//coordenada de dibujo y

mina.rectImag.x=(mina.c\*anchoCelda);//coordenada de dibujo x

mina.rectImag.w=anchoCelda;//ancho

mina.rectImag.h=altoCelda;//alto

crearCola(mina.cajas);

}

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

int getFila(Mina \*mina){

return mina->f;

}

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

void setFila(Mina &mina, int fila){

mina.f = fila;

}

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

int getColumna(Mina \*mina){

return mina->c;

}

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

void setColumna(Mina &mina, int columna){

mina.c = columna;

}

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

int getIP(Mina &mina){

return mina.ip;

}

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

void setIP(Mina &mina, int IP){

mina.ip = IP;

}

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

int\* getSecuencia(Mina &mina){

return mina.secuencia;

}

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

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

for(int i=0;i<5;i++){

mina.secuencia[i] = secuencia[i];

}

}

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

int getSecuenciaActual(Mina &mina){

return mina.secuenciaActual;

}

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

void setSecuenciaActual(Mina &mina, int secuencia){

mina.secuenciaActual = secuencia;

}

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

int getIntervarloActual(Mina &mina){

return mina.intervaloActual;

}

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

void setIntervaloActual(Mina &mina, int intervalo){

mina.intervaloActual = intervalo;

}

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

string getCodItem(Mina \*mina){

return mina->codItem;

}

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

void setCodItem(Mina &mina, string codItem){

mina.codItem = codItem;

}

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

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

SDL\_RenderCopy(renderer,mina->imagen,NULL,&(mina->rectImag));

}

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

void destruirMina(Mina \*mina){

SDL\_DestroyTexture(mina->imagen);

eliminarCola(mina->cajas);

}

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

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

PtrNodoLista ptrVagon = primero(locomotora);

bool produccionEntregada = false;

while(ptrVagon!=finLista()){

Vagon \*vagon = (Vagon\*)ptrVagon->ptrDato;

if(getCodItem(\*vagon)==getCodItem(&mina) || getCodItem(\*vagon)=="nada"){

PtrNodoCola ptrCaja = colaFrente(mina.cajas);

while(ptrCaja!=finCola()){

Caja \* caja = (Caja\*) ptrCaja->ptrDato;

if((getCapacidad(\*vagon)-getCantidadItem(\*vagon))>=getCapacidadMaxima(\*caja)){

produccionEntregada = true;

if(getCodItem(\*vagon)=="nada"){

setCodItem(\*vagon, getCodItem(&mina));

}

Caja \* cajaLista = new Caja;

\*cajaLista = \*(Caja\*)desencolar(mina.cajas);

agregar(\*getCajas(\*vagon), cajaLista);

setCantidadItem(\*vagon,getCapacidadMaxima(\*cajaLista)+getCantidadItem(\*vagon));

ptrCaja = colaFrente(mina.cajas);

}

else{ ptrCaja = finCola();}

}

}

ptrVagon = siguiente(locomotora, ptrVagon);

}

if(!produccionEntregada){

while(!colaVacia(mina.cajas)){

desencolar(mina.cajas);

}

}

}

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

**MONEDA.CPP**

#include "Moneda.h";

#include <SDL.h>

#include <SDL\_image.h>

#include <iostream>

#include <string.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#include <fstream>

#include <sstream>

using namespace std;

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

moneda.fila=fila;//coordenada logica y

moneda.columna=columna;//coordenada logica x

moneda.imagen=IMG\_LoadTexture(renderer,"img/moneda.png");

moneda.rectImag.y=(moneda.fila\*altoCelda);

moneda.rectImag.x=(moneda.columna\*anchoCelda);

moneda.rectImag.w=anchoCelda;

moneda.rectImag.h=altoCelda;

moneda.monedaVida=monedaVida;

moneda.intervaloFinal=intervaloFinal;

}

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

int getFila(Moneda \*moneda){

return moneda->fila;

}

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

int getColumna(Moneda \*moneda){

return moneda->columna;

}

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

int getIntervaloVida(Moneda &moneda){

return moneda.monedaVida;

}

int getIntervaloFinal(Moneda &moneda){

return moneda.intervaloFinal;

}

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

bool setMonedaVida(Moneda &moneda, int monedaVida){

moneda.monedaVida=monedaVida;

}

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

bool setIntervaloFinal(Moneda &moneda, int intervaloFinal){

moneda.intervaloFinal=intervaloFinal;

}

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

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

SDL\_RenderCopy(renderer,moneda->imagen,NULL,&(moneda->rectImag));

}

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

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

/\* srand(time(NULL));

int i=0;

i=rand()%5;

stringstream aux;

aux <<"img/suelo\_"<<i<<".png";

\*/

moneda->imagen=IMG\_LoadTexture(renderer,"img/suelo\_01.png");

}

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

void destruirMoneda(Moneda \*moneda){

SDL\_DestroyTexture(moneda->imagen);

}

**PARAMETROS.CPP**

#include "Parametros.h"

#include <iostream>

#include <string>

using namespace std;

/\*\* Definicion de 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){

// parametros.clave="NULL";

parametros.valor="NULL";

parametros.claveS=parametros.valor;

parametros.claveP=parametros.valor;

parametros.claveA=parametros.valor;

parametros.claveposXE=parametros.valor;

parametros.claveposYE=parametros.valor;

parametros.claveIM=parametros.valor;

parametros.claveVM=parametros.valor;

parametros.claveIB=parametros.valor;

parametros.claveIP=parametros.valor;

parametros.claveVB=parametros.valor;

}

/\*\* 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 haber sido creada y no debe estar destruida

post: el atributo clave queda seteado en Parametros

Parametros: instancia sobre la que se invoca la primitiva

clave: valor a asignar en Parametros \*\*/

void setClaveS(Parametros &parametros, string clav){

parametros.claveS=clav;

}

void setClaveP(Parametros &parametros, string clav){

parametros.claveP=clav;

}

void setClaveA(Parametros &parametros, string clav){

parametros.claveA=clav;

}

void setClavePosXE(Parametros &parametros, string clav){

parametros.claveposXE=clav;

}

void setClavePosYE(Parametros &parametros, string clav){

parametros.claveposYE=clav;

}

void setClaveIM(Parametros &parametros, string clav){

parametros.claveIM=clav;

}

void setClaveVM(Parametros &parametros, string clav){

parametros.claveVM=clav;

}

void setClaveIB(Parametros &parametros, string clav){

parametros.claveIB=clav;

}

void setClaveVB(Parametros &parametros, string clav){

parametros.claveVB=clav;

}

void setClaveIP(Parametros &parametros, string clav){

parametros.claveIP=clav;

}

string getClaveS(Parametros &parametros){

return parametros.claveS;

}

string getClaveP(Parametros &parametros){

return parametros.claveP;

}

string getClaveA(Parametros &parametros){

return parametros.claveA;

}

string getClavePosXE(Parametros &parametros){

return parametros.claveposXE;

}

string getClavePosYE(Parametros &parametros){

return parametros.claveposYE;

}

string getClaveIM(Parametros &parametros){

return parametros.claveIM;

}

string getClaveVM(Parametros &parametros){

return parametros.claveVM;

}

string getClaveIB(Parametros &parametros){

return parametros.claveVB;

}

string getClaveVB(Parametros &parametros){

return parametros.claveVB;

}

string getClaveIP(Parametros &parametros){

return parametros.claveIP;

}

/\*\* pre: la instancia Parametros debe haber sido creada y no debe estar destruida

post: el atributo valor queda seteado en Parametros

Parametros: instancia sobre la que se invoca la primitiva

clave: valor a asignar en el Parametro \*\*/

void setValor(Parametros &parametros, string val){

parametros.valor=val;

}

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

post: retorna el atributo valor de Parametros

Parametros: instancia sobre la que se invoca la primitiva \*\*/

string getValor(Parametros &parametros){

return parametros.valor;

}

**PILA.CPP**

#include "Pila.h"

using namespace std;

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

/\* Funciones auxiliares \*/

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

PtrNodoPila crearNodoPila(PtrDato ptrDato) {

/\* reserva memoria para el nodo y luego completa sus datos \*/

PtrNodoPila ptrAux = new NodoPila;

ptrAux->ptrDato = ptrDato;

ptrAux->sgte = NULL;

return ptrAux;

}

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

void eliminarNodoPila(Pila &pila) {

PtrNodoPila ptrExCima;

/\* verifica que la pila no esté vacia y que nodo no sea fin\*/

if (! pilaVacia(pila)) {

ptrExCima = cima(pila);

pila.primero = cima(pila)->sgte;

// Si el dato es un TDA, acá habría que llamar al destructor.

delete ptrExCima;

}

}

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

/\* Implementación de Primitivas \*/

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

void crearPila(Pila &pila) {

pila.primero = finPila();

}

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

PtrNodoPila finPila() {

return NULL;

}

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

bool pilaVacia(Pila &pila){

return (pila.primero == finPila());

}

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

PtrNodoPila cima(Pila &pila){

return pila.primero;

}

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

PtrNodoPila agregar(Pila &pila, PtrDato ptrDato){

/\* crea el nodo \*/

PtrNodoPila ptrNuevoNodo = crearNodoPila(ptrDato);

ptrNuevoNodo->sgte = pila.primero;

pila.primero = ptrNuevoNodo;

return ptrNuevoNodo;

}

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

PtrDato sacar(Pila &pila){

PtrDato dato = NULL;

PtrNodoPila nodo = cima(pila);

if ((! pilaVacia(pila))){

dato = nodo->ptrDato;

eliminarNodoPila(pila);

}

return dato;

}

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

void eliminarPila(Pila &pila){

while((! pilaVacia(pila))){

eliminarNodoPila(pila);

}

}

**TERRENO.CPP**

#include "Terreno.h"

#include "Bandido.h"

#include "Mina.h"

#include "Moneda.h"

#include "Vagon.h"

#include "Lista.h"

#include "Estacion.h"

#include <stdio.h>

#include <string.h>

#include <SDL\_image.h>

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

// IMPLEMENTACION DE PRIMITIVAS

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

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

terreno.fila=fila;

terreno.columna=columna;

terreno.ptrNodoVagon=NULL;

terreno.ptrBandido=NULL;

terreno.ptrMina=NULL;

terreno.ptrMoneda=NULL;

terreno.ptrEstacion=NULL;

char imagen[30]="img/suelo\_";

strcat(imagen,rand);

strcat(imagen,".png");

terreno.imagen=IMG\_LoadTexture(renderer,imagen);

terreno.rectImag.y=(terreno.fila\*alto);

terreno.rectImag.x=(terreno.columna\*ancho);

terreno.rectImag.w=ancho;

terreno.rectImag.h=alto;

}

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

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

SDL\_RenderCopy(renderer,terreno.imagen,NULL,&(terreno.rectImag));

}

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

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

if(terreno.ptrBandido!=NULL){

dibujarBandido(terreno.ptrBandido,renderer);

}

if(terreno.ptrMina!=NULL){

dibujarMina(terreno.ptrMina,renderer);

}

if(terreno.ptrMoneda!=NULL){

dibujarMoneda(terreno.ptrMoneda,renderer);

}

if(terreno.ptrNodoVagon!=NULL){

dibujarVagon(\*(Vagon\*)terreno.ptrNodoVagon->ptrDato,renderer,intervalo);

} if(terreno.ptrEstacion!=NULL){

dibujarEstacion(terreno.ptrEstacion,renderer);

}

}

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

void destruirTerreno(Terreno &terreno){

SDL\_DestroyTexture(terreno.imagen);

if(terreno.ptrBandido!=NULL){

destruirBandido(terreno.ptrBandido);

delete terreno.ptrBandido;

}

if(terreno.ptrMina!=NULL){

destruirMina(terreno.ptrMina);

delete terreno.ptrMina;

}

if(terreno.ptrMoneda!=NULL){

destruirMoneda(terreno.ptrMoneda);

delete terreno.ptrMoneda;

}

if(terreno.ptrEstacion!=NULL){

destruirEstacion(\*(terreno.ptrEstacion));

delete terreno.ptrEstacion;

}

}

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

**VAGON.CPP**

#include "Vagon.h"

#include "Item.h"

#include "Pila.h"

#include <string.h>//permite utilizar strcat(char,char) strcpy(char,char);

#include <stdlib.h>//permite utilizar itoa(int,char,int);

#include <SDL.h>

#include <SDL\_image.h>

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

strcpy(vagon.miniatura,miniatura);

vagon.fila= fila;//coordenada logica y

vagon.columna=columna;//coordenada logica x

strcpy(vagon.direccion,direccion);

vagon.anchoCelda=anchoCelda;

vagon.altoCelda= altoCelda;

vagon.detenido=false;

crearPila(vagon.cajas);

vagon.capacidad = capacidad;

vagon.cantidadItem = 0;

vagon.codItem=item;

}

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

char\* getMiniatura(Vagon &vagon){

return vagon.miniatura;

}

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

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

strcpy(vagon.miniatura,miniatura);

}

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

int getFila(Vagon &vagon){

return vagon.fila;

}

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

void setFila(Vagon &vagon,int desplazamiento){

vagon.fila=vagon.fila+desplazamiento;

}

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

int getColumna(Vagon &vagon){

return vagon.columna;

}

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

void setColumna(Vagon &vagon,int desplazamiento){

vagon.columna=vagon.columna+desplazamiento;

}

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

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

int getAnchoCelda(Vagon &vagon){

return vagon.anchoCelda;

}

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

void setAnchoCelda(Vagon &vagon,int anchoCelda){

vagon.anchoCelda=anchoCelda;

}

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

int getAltoCelda(Vagon &vagon){

return vagon.altoCelda;

}

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

void setAltoCelda(Vagon &vagon,int altoCelda){

vagon.altoCelda=altoCelda;

}

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

int getAltoSprite(Vagon &vagon){

return vagon.altoSprite;

}

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

void setAltoSprite(Vagon &vagon,int altoSprite){

vagon.altoSprite=altoSprite;

}

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

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

char imagen[10] = "img/"; //strcat(<cadena\_destino>, <cadena\_fuente>);

strcat(imagen,vagon.miniatura); //La función strcat añade el contenido de la <cadena\_fuente> a la <cadena\_destino>.

strcat(imagen,"/");

strcat(imagen,vagon.direccion);strcat(imagen,"/");

if(!(vagon.detenido)){

char cadenaIntervalo[5];

itoa(intervalo,cadenaIntervalo,10);

strcat(imagen,cadenaIntervalo);

}

else{

strcat(imagen,"0");

}

strcat(imagen,".png");

//img\c1\aba\0.png

vagon.imagen=IMG\_LoadTexture(renderer,imagen);

//SDL\_QueryTexture(vagon.imagen,NULL,NULL,0,0);//tal vez este no haga filaalta, se utiliza para buscar el ancho/alto de la imagen pero nosotros ya lo tenemos de antemano

//evaluamos los desplazamientos

int desplazamientoHorizontal=0;

int desplazamientoVertical=0;

if(strcmp(vagon.direccion,"aba")==0)desplazamientoVertical=1;

if(strcmp(vagon.direccion,"arr")==0)desplazamientoVertical=-1;

if(strcmp(vagon.direccion,"der")==0)desplazamientoHorizontal=1;

if(strcmp(vagon.direccion,"izq")==0)desplazamientoHorizontal=-1;

if(!(vagon.detenido)){

vagon.rectImag.y= vagon.fila\*vagon.altoCelda +(desplazamientoVertical\*(vagon.altoCelda/10)\*intervalo);//coordenada de dibujo y

vagon.rectImag.x= vagon.columna\*vagon.anchoCelda +(desplazamientoHorizontal\*(vagon.anchoCelda/10)\*intervalo);//coordenada de dibujo x

vagon.rectImag.w=vagon.anchoCelda;

vagon.rectImag.h=vagon.altoCelda;

}

else{

vagon.rectImag.y= vagon.fila\*vagon.altoCelda;//coordenada de dibujo y

vagon.rectImag.x= vagon.columna\*vagon.anchoCelda;//coordenada de dibujo x

vagon.rectImag.w=vagon.anchoCelda;

vagon.rectImag.h=vagon.altoCelda;

}

SDL\_RenderCopy(renderer,vagon.imagen,NULL,&(vagon.rectImag));

}

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

void destruirVagon(Vagon &vagon){

SDL\_DestroyTexture(vagon.imagen);

eliminarPila(vagon.cajas);

//delete &vagon;

}

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

char\* getDireccion(Vagon &vagon){

return vagon.direccion;

}

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

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

strcpy(vagon.direccion,direccion);

}

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

int getCapacidad(Vagon &vagon){

return vagon.capacidad;

}

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

void setCapacidad(Vagon &vagon, int capacidad){

vagon.capacidad = capacidad;

}

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

int getCantidadItem(Vagon &vagon){

return vagon.cantidadItem;

}

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

void setCantidadItem(Vagon &vagon, int cantidadItem){

vagon.cantidadItem = cantidadItem;

}

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

string getCodItem(Vagon &vagon){

return vagon.codItem;

}

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

void setCodItem(Vagon &vagon, string codItem){

vagon.codItem = codItem;

}

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

Pila \* getCajas(Vagon &vagon){

return &(vagon.cajas);

}

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

void setCajas(Vagon &vagon, Pila cajas){

vagon.cajas = cajas;

}

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