Enunciado Trabajo Práctico

Clash of UNLA

Algoritmos y Estructuras de Datos

Ing. Damián Santos – Lic. Romina Mansilla – Agustín Di Stefano

UNLA 2019

|  |  |  |
| --- | --- | --- |
| DNI | Nombre | Evaluación Individual |
|  | Rocio Trinidad |  |
|  | Gabriel Esperon |  |
|  | Pablo De Lafore |  |
|  | Tomas |  |
|  | Juan Gutierrez |  |
| 33401014 | Carlos Nicolas Goday |  |
| Evaluación Trabajo |  | |

Contenido

[Enunciado 3](#_Toc8332660)

[1. Objetivo 3](#_Toc8332661)

[2. Descripción 3](#_Toc8332662)

[3. Archivos. 6](#_Toc8332663)

[4. Presentación 7](#_Toc8332664)

[5. Revisiones 9](#_Toc8332665)

[Primera Entrega 12](#_Toc8332666)

[1. Desarrollo de la estrategia de resolución 12](#_Toc8332667)

[2. Cronograma 12](#_Toc8332668)

[3. División de tareas 13](#_Toc8332669)

[4. Diagrama TDA 14](#_Toc8332670)

[5. Interfaces .h 14](#_Toc8332671)

# Enunciado

## Objetivo

Desarrollar una aplicación en C++ que permita administrar el juego denominado Clash of UNLA.

## Descripción

El objetivo del juego es que un personaje recorra un terreno en búsqueda de una determinada cantidad de ítems. Para hacerlo, cuenta con un vehículo conformado por una locomotora a la que se le pueden ir incorporando vagones. Cada vagón sólo puede almacenar un tipo de ítem (no se pueden mezclar).

El terreno es una matriz de X por Y posiciones. En cada posición podremos encontrar:

* Terreno: Una posición en la que no hay nada en particular.
* Locomotora: Es el vehículo principal.
* Vagón: Es un contenedor que se agrega al Vehículo para poder transportar ítems. Los vagones tienen una capacidad máxima de carga.
* Ítem: Los ítems son recursos naturales que deben ser recolectados. Hay distintos tipos: oro, plata, bronce, platino, roca, carbón.
* Mina: Una mina es una instalación que genera un determinado ítem. La producción de ítems es en la forma de lingotes de 1 Kg.
* Caja: Las cajas son contenedores para los lingotes.
* Bandido: Los bandidos se encuentran distribuidos a lo largo del terreno.
* Moneda: Las monedas son el dinero que permite comprar los vagones.
* Estación: La estación es el puesto en el que se compran vagones.

*Tiempo de juego.*

El tiempo de juego deberá estar definido en intervalos de S segundos.

En cada intervalo, el vehículo se moverá una posición, arrastrando a la locomotora y a todos sus vagones. Si choca consigo mismo o sale del terreno, se pierde la partida.

El jugador podrá alterar el recorrido del vehículo utilizando las teclas del teclado.

El vehículo sólo podrá detenerse al momento de estacionar frente a una mina o una estación (ver más adelante). Para retomar el movimiento habrá que direccionarlo con el teclado.

*Comanda*

La comanda es el objetivo que define la cantidad de ítems que debe recolectar el jugador para finalizar el juego. Se carga desde un archivo de texto.

*Monedas.*

Deberá existir un algoritmo que genere monedas según una cantidad de intervalos aleatorios. Es decir, la primera puede generarse luego de transcurridos 3 intervalos; la segunda, luego de 5; la tercera a los 10 y así. La diferencia entre la aparición de una moneda y la siguiente no puede superar los IM intervalos. El tiempo máximo de vida de una moneda se determina en forma aleatoria al momento de crearla y no podrá superar los VM intervalos.

*Vagones.*

En primer lugar, el jugador deberá recolectar monedas. Para comprar vagones, será necesario estacionar enfrente de la estación (“chocar” con la estación). En ese momento, la cantidad de monedas que tenga serán consumidas en un vagón que podrá soportar un peso máximo según la siguiente ecuación: Peso\_máximo = 5kg\*cantidad\_monedas.

Una vez que un vagón almacenó un determinado ítem no puede ser utilizado para otro, aún cuando haya quedado vacío.

*Producción en las minas.*

Las minas producen un único ítem, en forma de lingotes, con una secuencia establecida. Por ejemplo, una mina de oro puede tener la secuencia 2, 2, 4, 3, 5. Esto significa que, en el primer intervalo, generará una caja con dos lingotes; en el segundo, otra con 2; en el tercero, una con 4; en el cuarto, una con 3; y en el quinto, una con 5. Al finalizar, vuelve a iniciar la secuencia. La producción de cada caja ocurre cada IP intervalos.

Las cajas se crean en función de la producción de la mina y tienen una capacidad máxima de lingotes que es igual a los lingotes producidos al momento de generarla. En el ejemplo anterior, tendríamos 5 cajas con máximos de 2, 2, 4, 3 y 5 respectivamente.

Para recolectar la producción, la locomotora tiene que estacionarse enfrente de la mina (“chocar” contra la mina). En ese momento, la primera caja producida se incorpora al vehículo. Para que esto ocurra, el mismo deberá contar con un vagón específico para ese material y con capacidad de almacenamiento suficiente para la nueva caja, o con un vagón vacío en el que se pueda almacenar un nuevo ítem. Si ninguna de las dos ocurre, se pierde toda la producción de la mina por penalidad.

Las minas están definidas en un archivo de texto.

*Bandidos.*

Si el vehículo se topa con un bandido, este le pedirá que pague con determinado ítem para dejarlo pasar. En caso de no poder realizar el pago, el bandido nos romperá el último vagón. Si el vehículo no tiene vagones, se romperá la locomotora y perdemos el juego.

Los bandidos aparecen y desaparecen en forma aleatoria y tienen un campo de acción que abarca una superficie cuadrada del terreno delimitada a partir de moverse A posiciones (parámetro de la aplicación) en las cuatro direcciones.

Cualquier parte del vehículo que esté dentro del rango de acción de un bandido significará que el mismo fue atacado. Luego de atacar, el bandido desaparece.

Los ítems se consumen de los vagones a partir del último ingresado. Por ejemplo, supongamos que en la recolección de oro un vagón quedó de la siguiente manera:

* (segunda caja) 2 lingotes.
* (primer caja) 3 lingotes.

Nos daría una estructura similar a la siguiente:

|  |  |
| --- | --- |
| 2/2 | 3/3 |

Donde: x/y significa:

* x cantidad actual de lingotes.
* y capacidad máxima.

Si el bandido nos pide 1 lingote, tendremos que desmontar la segunda caja, retirar un lingote, pagarle y volver a insertar la caja con un lingote menos, quedando:

|  |  |
| --- | --- |
| 1/2 | 3/3 |

Si pidiera cuatro, debería sacar 2 y 2, quedando:

|  |
| --- |
| 1/3 |

Las cajas vacías se descartan, es decir, no vuelven a ingresarse al vagón.

*Generación de Bandidos.*

Deberá existir un algoritmo que genere un bandido según una cantidad de intervalos aleatorios. Es decir, el primero puede generarse luego de transcurridos 3 intervalos; el segundo, luego de 5; el tercero a los 10 y así. La diferencia entre la aparición de un bandido y el siguiente no puede superar los IB intervalos. El tiempo máximo de vida de un bandido se determina en forma aleatoria al momento de crearlo y no podrá superar los VB intervalos.

La generación del bandido también determina en forma aleatoria el ítem que va a requerir y la cantidad del mismo, pudiendo ser un número entero entre 1 y el máximo permitido (P).

## Archivos.

* Parámetros.txt

Los parámetros de la aplicación se cargarán de un archivo

Formato de cada línea:

clave=valor

Campos:

* Clave: cadena
* Valor: cadena

Claves:

* S: Segundos que dura un intervalo.
* P: cantidad máxima que puede pedir un bandido.
* A: posiciones para definir el área de Ataque de un bandido.
* posXE: posición x de la estación.
* posYE: posición y de la estación.
* IM: intervalo máximo para generación de monedas.
* VM: intervalos máximos de vida de una moneda.
* IB: máxima cantidad de intervalos para la aparición de bandidos.
* VB: tiempo máximo de vida de un bandido.
* IP: intervalos entre producciones de las minas.
* Comanda.txt

Formato de cada línea:

codItem;cantidad

Campos:

* *codItem: una cadena con los siguientes valores: oro, plata, bronce, platino, roca, carbón.*
* *cantidad: entero.*
* Minas.txt

Formato de cada línea:

posX;posY;codItem;IP;seq1;seq2;seq3;seq4;seq5

Campos:

* *posX, posY: posición donde se ubica la mina.*
* *codItem: código del ítem que va a producir.*
* *IP: intervalo de producción.*
* *SeqX: especificación de la secuencia de producción. Siempre serán 5 en total.*

NOTA:

En todos los casos, el separador de campos debe ser “;”.

Como parte del trabajo, los alumnos deberán desarrollar un archivo de pruebas con datos válidos para ejecutar las validaciones.

## Presentación

A continuación se detalla el cronograma de actividades para el trabajo práctico:

26/04/19: Presentación del enunciado.

03/05/19: Consulta del trabajo práctico.

10/05/19: **Primer Entrega:** “Estrategia de Resolución”. Se presentará la estrategia de resolución propuesta por el equipo.

17/05/19: Consulta del trabajo práctico.

24/05/19: Consulta del trabajo práctico.

31/05/19: Consulta del trabajo práctico.

07/06/19: Parcial.

14/06/19: **Segunda Entrega:** Presentación y Evaluación del TP.

21/06/19: Entrega de notas del parcial y revisión.

28/06/19: Recuperatorio del Parcial.

05/07/19: **Recuperatorio del TP**. Entrega de notas del parcial y revisión.

**Primer Entrega:** “Estrategia de Resolución”.

Para esta entrega deberá presentarse un documento impreso (en un folio o carpeta) conteniendo, como mínimo:

* Carátula de presentación con los datos de los integrantes del equipo (primer hoja de este documento).
* Indice de contenidos (en caso de ser necesario).
* Desarrollo de la estrategia de resolución detallando:
  + Estructuras a utilizar (arrays, pilas, listas, colas).
  + TDAs y sus relaciones (diagrama de interacción de todos los componentes).
  + Estrategia de resolución de operaciones.
* División del trabajo y cronograma: se deberá presentar la división de actividades dentro del grupo y un cronograma de trabajo a alto nivel.

Todas las hojas deben estar numeradas.

**Segunda Entrega**.

Para esta entrega deberá presentarse:

* Una copia impresa del enunciado del trabajo práctico (TODO este documento, incluyendo los anexos).
* Una copia impresa de la estrategia de resolución final del trabajo práctico (con los mismos requerimientos que en la “Primer Entrega”.
* Una copia impresa de todos los archivos de prueba presentados por los alumnos.
* Una copia impresa de todos los archivos del proyecto (.h y .cpp). Poner como encabezado de cada hoja el nombre del archivo.
* Un CD conteniendo en formato digital todos los puntos anteriores y el proyecto completo.
* Se deberá incluir la primer entrega corregida.

La presentación deberá ser en un folio o carpeta, en forma prolija y debidamente identificada. Los CDs deberán contener el número de grupo y el nombre y los padrones o documentos de identidad de cada uno de los integrantes y deberán estar correctamente adjuntos al resto del trabajo práctico de forma tal que no puedan perderse. Además, deberá incluirse **todo** el proyecto desarrollado (**la carpeta completa** generada por el IDE, con los archivos del proyecto y el código fuente) incluyendo los archivos de pruebas.

Todas las hojas deben estar numeradas.

El incumplimiento de cualquiera de las normas de entrega implicará la desaprobación del trabajo práctico.

Metodología de evaluación:

La Evaluación de los trabajos prácticos contará con una etapa grupal y una individual.

* Grupal: Se realizará un conjunto de pruebas sobre el trabajo presentado por los alumnos en presencia de los mismos. Se deberá aprobar la totalidad de las pruebas. En caso de que una prueba falle, los alumnos podrán intentar corregir el código mientras dure la evaluación.
* Individual: Se realizará una evaluación individual oral o escrita para cada alumno. Los temas a evaluar podrán ser, por ejemplo: preguntas teóricas sobre el contenido de la materia, preguntas sobre el trabajo práctico, codificación de alguna primitiva o modificación del trabajo práctico, etc.

La nota final de la cursada se calculará en función de las notas obtenidas en forma grupal e individual. La nota grupal será el promedio entre la primer presentación y el recuperatorio (en caso de necesitarlo). Por este motivo, SOLO deberán presentarse aquellos grupos que hayan concluido TODO el trabajo práctico ya que no se harán evaluaciones parciales.

## Revisiones

Anexo – Correcciones

|  |  |  |
| --- | --- | --- |
| Prueba | Resultado | Comentario |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Anexo – Correcciones (copia para los alumnos)

|  |  |  |
| --- | --- | --- |
| Prueba | Resultado | Comentario |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Primera Entrega

## Desarrollo de la estrategia de resolución

Para la resolución del trabajo práctico se utilizaran listas, pilas y colas como se describirán en los TDA´s detallados a continuación

El grupo pensó en desarrollar primeramente las declaraciones de las estructuras que se utilizaran, es decir el "que", y luego una vez terminado este pasó, continuaremos con la implementación de sus primitivas.

Una vez listas las implementaciones vamos a realizar la integración (que también se ira llevando a cabo durante la fase de implementación por requerirse por las relaciones entre ellas). En paralelo al desarrollo de los TDA se desarrollara en la medida de lo posible la interfaz gráfica.

Algunos aspectos a considerar durante el desarrollo serán los siguientes:

* Respecto al tiempo de juego se implementara un conteo de reloj mediante la librería ctime de c++ "#include <ctime>" para por cada "S" segundos mover el tren implementando un flag que se incrementara cuando pase cierto tiempo. El juego estará activo continuamente para detectar los ingresos de teclado y cada S segundos procederá a mover la locomotora.
* Para la generación de monedas se utilizara la función random que generara el tiempo de la primera [0..IM] ocurrencia y fijara su tiempo de vida [0..VM] también con un valor random, cuando se llegue a insertar la moneda en el terreno se disparara la creación de la segunda moneda fijando su tiempo de creación y su tiempo de vida y así sucesivamente.
* Para la generación de minas se usara en principio una lista circular, al llegar al último elemento se volverá al inicio.
* Para la generación de bandidos se usara un mecanismo similar al de las monedas. Con la consideración de tiempo de creación en el intervalo [0..IB] y tiempo de vida [0..VB]. A esto se suma el tipo de ítem y la cantidad requerida que será también random [1..P]
* Para la lectura de archivos, se implementara un TDA que encapsule la lectura, el mismo retornada un TDA de parámetros y una lista tanto para comanda de tipo "Comanda" como para Minas de tipo "Mina".

## Cronograma

Semana 3/5 a 10/5:

* Cumplimiento primera entrega TP y desarrollo de .h
* Desarrollo e implementación de lectura y carga de archivos de entrada
* Integración de github de todos los integrantes del grupo
* División de tareas

Semana 10/5 a 24/5:

* Desarrollo de implementación para los TDA definidos
* Integración de TDA
* Comienzo de integración de estructuras de listas y pilas al proyecto
* Desarrollo de pantallas

Semana 24/5 al 31/5

* Pruebas de integración de proyecto, de integración de las estructuras dinámicas.
* Renderizado de las diferentes estructuras.
* Desarrollo de pantallas

Semana 31/5 al 14/5

* Terminado de renderizado
* Pruebas de TP
* Refinamiento de código y mejoras posibles.

## División de tareas

Minas, Cajas 🡪 Roció

Terreno 🡪 Pablo

Estación, posición, lectura archivos 🡪 Nicolas

Locomotora, Vagón 🡪 Tomas

Moneda, Diagramas 🡪 Gabriel

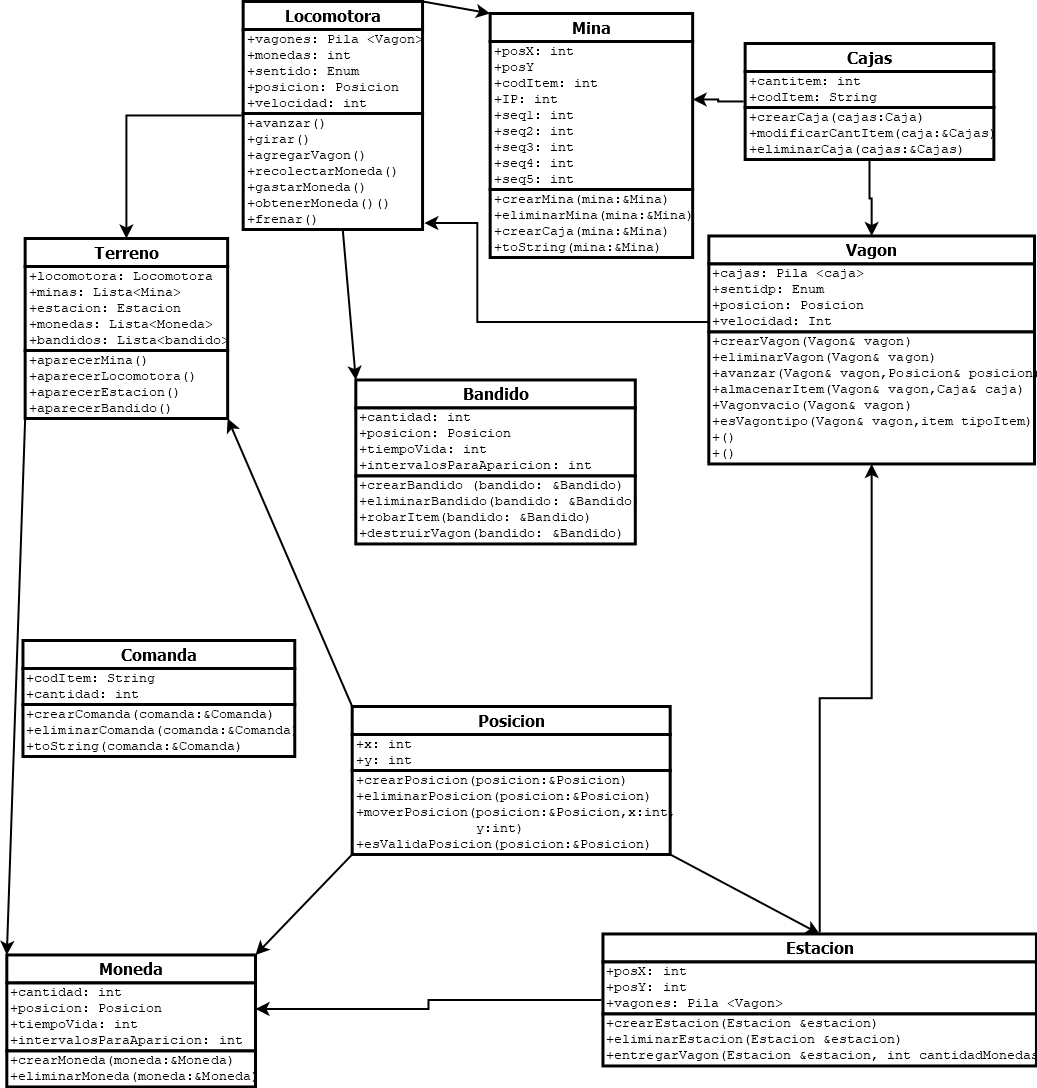
Bandido 🡪 Juan

Renderizado 🡪 grupal

Integración 🡪 grupal

Pruebas 🡪 A definir

## Diagrama TDA



## 

## Interfaces .h

#ifndef \_\_CAJAS\_H\_\_

#define \_\_CAJAS\_H\_\_

#include <iostream>

**using namespace std**;

/\*

Axiomas

No puede contener mas de 5 item.

\*/

/\* Tipo de estructura de Cajas \*/

**typedef struct** Cajas{

**int** cantitem;

**string** codItem;

};

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

/\*PRE: Que la caja existe

Post: Devuelve cantidad de item

\*/

**int** getItem(Cajas &cajas);

/\*PRE:

Post: inicializa parametros de la caja

\*/

**void** crearCaja(Cajas &cajas);

/\*PRE: Que la caja existe

Post: Devuelve cantidad de item

\*/

**void** modificarCantItem(Cajas &cajas);

/\*PRE: Estacion creada

Post: libera recursos solicitados por estacion

\*/

**void** eliminarCaja(Cajas &cajas);

#endif // \_\_PARAMETROS\_H\_\_

#ifndef \_\_COMANDA\_H\_\_

#define \_\_COMANDA\_H\_\_

#include <iostream>

**using namespace std**;

/\*

Definicion de tipo de dato Comanda para cargar items de archivo

Cada comanda va a representar un reto a vencer en el juego

Atributos:

\* codItem

\* cantidad

Axiomas

\* codItem > 0, existente

\* cantidad < 100

\*/

/\* Tipo de estructura de Comanda \*/

**typedef struct** Comanda{

**string** codItem;

**int** cantidad;

}Comanda;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**string** getCodItem(Comanda &comanda);

**int** getCantidad(Comanda &comanda);

**void** setCodItem(Comanda &comanda, **string** codItem);

**void** setCantidad(Comanda &comanda, **int** cantidad);

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

/\*

PRE:

Post: inicializa parametros de comanda, codIten="", cantidad=0

\*/

**void** crearComanda(Comanda &comanda);

/\*

PRE: comada creada

Post: libera recursos solicitados por comanda

\*/

**void** eliminarComanda(Comanda &comanda);

/\*

PRE: Comanda existente y cargada

Post: Muestra por pantalla los atributos

\*/

**void** toString(Comanda &comanda);

#endif // \_\_PARAMETROS\_H\_\_

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

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

#include <iostream>

#include "Posicion.h"

#define COFICIENTE\_PESO\_VAGON 5

**using namespace std**;

/\*

Definicion de tipo de dato Estacion para representar el lugar de compra de vagones

Atributos:

Axiomas

\* Entregara vagon segun formula Peso\_mÃ¡ximo = 5kg\*cantidad\_monedas.

\* cantidad de vagones a entregar ilimitado

\*/

/\* Tipo de estructura de Estacion \*/

**typedef struct** Estacion{

Posicion posicion;

}Estacion;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

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

/\*

PRE:

Post: inicializa parametros de estacion

\*/

**void** crearEstacion(Estacion &estacion);

/\*

PRE: Estacion creada

Post: libera recursos solicitados por estacion

\*/

**void** eliminarEstacion(Estacion &estacion);

/\*

PRE: Estacion creada

Post: calcula y devuelve peso que soporta vagon

\*/

**int** entregarVagon(Estacion &estacion, **int** cantidadMonedas);

#endif // \_\_PARAMETROS\_H\_\_

#ifndef \_\_LECTURAARCHIVOS\_H\_\_

#define \_\_LECTURAARCHIVOS\_H\_\_

#include<iostream>

#include <fstream>

#include <sstream>

#include "Parametros.h"

#include "Comanda.h"

#include "Mina.h"

**using namespace std**;

/\*

Definicicion de tipo de dato Lector para manejo de Archivos en proyecto

Atributos:

\* fichero

Axiomas

\* Existen tres tipos de archivos Paramatros.txt,Comanda.txt, Minas.txt

\* Los archivos estan creados con las estructuras correctas, no hay datos erroneos

\*/

/\* Tipo de estructura del Lector \*/

**typedef struct** Lector{

**fstream** ficheroEntrada;

}Lector;

/\*

PRE: Lector no existente

Post:

\*/

**void** crearLector(Lector & lector);

/\*

PRE: Lector creado

Post: si Archivo abierto y asignado a lector, devuelve true

si error al abrir devuelve false

\*/

**bool** abrirArchivo(Lector & lector, **string** nombreArchivo);

/\*

PRE: Lector creado y archivo abierto

Post: Cierra archivo

\*/

**void** cerrarArchivo(Lector & lector);

/\*

PRE: Lector creado y archivo abierto sin errores, SE ESTA LEYENDO UN ARCHIVO DE PARAMETROS

Post: se devuelve instancia de Parametros con valores cargados

\*/

Parametros leerArchivoParametros(Lector & lector);

/\*

PRE: Lector creado y archivo abierto SE ESTA LEYENDO UN ARCHIVO DE COMANDAS /// ACA TIENE Q DEVOLVER UNA LISTA

Post: se devuelve lista con instancias de Comandas con valores cargados

\*/

Comanda leerArchivoComandas(Lector & lector);

/\*

PRE: Lector creado y archivo abierto SE ESTA LEYENDO UN ARCHIVO DE MINAS /// ACA TIENE Q DEVOLVER UNA LISTA

Post: se devuelve lista con instancias de Comandas con valores cargados

\*/

Mina leerArchivoMinas(Lector & lector);

/\*

PRE: Lector creado y archivo abierto

Post: devuelve true si es fin o false caso contrario

\*/

**bool** esFinDeArchivo(Lector & lector);

/\*

PRE: Lector existente

Post:

\*/

**void** eliminarLector(Lector & lector);

#endif // \_\_LECTURAARCHIVOS\_H\_\_

#include "posicion.h"

//#include "pila.h" para cuando se agreguen los vagones a la locomotora

#define velocidad 5//macro

**enum** sentido{//para definir los eventos del teclado

IZQUIERDA,DERECHA,ARRIBA,ABAJO

};

**typedef struct** Locomotora{

**int** monedasAdquiridas;

Posicion posicion;

sentido movimientoL;

**int** velocidadL;

//Pila vagones; ?

}Locomotora;

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Axiomas\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//velocidad >= 0

//la locomotora empieza con 0 monedas

//la locomotora empieza con velocidad 0 es decir esta quieta

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Primitivas\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*gets y setters\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** setMonedasAdquiridas(Locomotora& locomotora,**int** monedasAdquiridas);

**int** getMonedasAdquiridas(Locomotora& locomotora);

**void** setMovimientoL(Locomotora& locomotora,sentido movimientoL);

sentido getMovimientoL(Locomotora& locomotora);

**void** setPosicion(Locomotora& locomotora,Posicion posicion);

Posicion getPosicion(Locomotora& locomotora);

**void** setVelocidadL(Locomotora& locomotora,**int** velocidadL);

**int** getVeclodiadL(Locomotora& locomotora);

//pre:

//post:se inicializan los parametros de locomotora;

**void** crearLocomotora(Locomotora& locomotora);

//pre:locomotora creada

//post:se liberan los recursos utilizados por la locomotora

**void** eliminarLocomotora(Locomotora& locomotora);

//pre: locomotora creada

//post:se le asigna '0' al campo velocidadL de la locomotora

**void** frenarLocomotora(Locomotora& locomotora);

//pre: locomotora creada

//post: se le asigna el 'macro velocidad' al campo velocidadL de la locomotora

**void** arrancarLocomotora(Locomotora& locomotora);

//pre:Locomotora creada

//post:se le suma o resta el 'macro velocidad' a el campo 'x' de posicion

// dependiendo del evento del teclado

**void** avanzar(Locomotora& locomotora,Posicion& posicion);

//pre:Locomotora creada

//post: se le suma o resta el 'macro velocidad' a el campo 'y' de posicion

// dependiendo del evento del teclado

**void** girar(Locomotora& locomotora,Posicion& posicion);

//pre:Locomotora creada

//post: se suma la moneda ontenida a la cantidad total de monedasAdquiridas

**void** obtenerMoneda(Locomotora& locomotora);

//pre:Locomotora creada

//post: se le resta cantMonedas al campo monedasAdquiridas de locomotora y devuelve el nuevo total de monedasAdquiridas disponibles

// si no se cumple con la cantMonedas a pagar retorna 0

**int** gastarMonedas(Locomotora& locomotora,**int** cantMonedas);

#endif // \_locomotora\_h\_

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

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

#include <iostream>

#include "Cajas.h"

**using namespace std**;

/\*Declaracion de cola de cajas \*/

**typedef struct** Colacaja{

Cajas caja;

**struct** nodo \*sgte;

}Colacaja;

/\*

Definicion de tipo de dato mina para representar estructura en donde se crean las cajas de item

\*/

**typedef struct** Mina{

**int** posX;

**int** posY;

**int** codItem;

**int** IP;

**int** seq1;

**int** seq2;

**int** seq3;

**int** seq4;

**int** seq5;

Colacaja colacaja;

}Mina;

/\*

AXIOMAS

Los item pueden ser: oro, plata, bronce, platino, roca y carbÃ³n.

La secuencia esta establecida por 5 numeros enteros del 1 al 5 inclusive

/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** getPosX(Mina &mina);

**int** getPosY(Mina &mina);

**int** getCodItem(Mina &mina);

**int** getIp(Mina &mina);

**int** getSeq1(Mina &mina);

**int** getSeq2(Mina &mina);

**int** getSeq3(Mina &mina);

**int** getSeq4(Mina &mina);

**int** getSeq5(Mina &mina);

**void** setPosX(Mina &mina, **int** posX);

**void** setPosY(Mina &mina, **int** posY);

**void** setCodItem(Mina &mina, **int** codItem);

**void** setIp(Mina &mina, **int** ip);

**void** setSeq1(Mina &mina, **int** seq1);

**void** setSeq2(Mina &mina, **int** seq2);

**void** setSeq3(Mina &mina, **int** seq3);

**void** setSeq4(Mina &mina, **int** seq4);

**void** setSeq5(Mina &mina, **int** seq5);

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

/\*

PRE:

POST: Inicializar los parametros de la mina

\*/

**void** crearMina (Mina &mina);

/\*

PRE: Mina creada

POST: Eliminar recursos solicitados por la mina

\*/

**void** eliminarMina (Mina &mina);

/\*

PRE: Mina creada

POST: Devuelve cola de cajas

\*/

**void** crearcaja (Mina &mina);

/\*

PRE: Mina existente y cargada

Post: Muestra por pantalla los atributos

\*/

**void** toString(Mina &mina);

#endif // \_\_PARAMETROS\_H\_\_

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

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

#include <iostream>

**using namespace std**;

/\*

Definicion de tipo de dato Moneda

Las monedas son el elemento que debera recolectar el tren para comprar vagones

Atributos:

\* cantidad

\* posicion

\* Tiempo aparicion

\* Tiempo duracion

Axiomas

\* codItem > 0, existente

\* cantidad < 100

\* tiempo aparicion [0..IM]

\* tiempo vida [0..IV]

\*/

/\* Tipo de estructura de Moneda\*/

**typedef struct** Moneda{

**int** cantidad;

Posicion posicion;

**int** aparicion;

**int** duracion;

}Moneda;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

Posicion getPosicion(Moneda &moneda);

**int** getCantidad(Moneda &moneda);

**int** getAparicion(Moneda &moneda);

**int** getDuracion(Moneda &moneda);

**void** setPosicion(Moneda &moneda, Posicion &posicion);

**void** setAparicion(Moneda &moneda, **int** duracion);

**void** setDuracion(Moneda &moneda, **int** aparicion);

**void** setCantidad(Moneda &moneda, **int** cantidad);

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

/\*

PRE:

Post: inicializa parametros de moneda, posicion=(0,0), duracion, aparicion,cantidad se calculan en forma random

\*/

**void** crearMoneda(Moneda &moneda);

/\*

PRE: Moneda creada

Post: libera recursos solicitados por comanda

\*/

**void** eliminarMoneda(Moneda &moneda);

/\*

PRE: Moneda existente y cargada

Post: Muestra por pantalla los atributos

\*/

**void** toString(Moneda &moneda);

#endif // \_\_PARAMETROS\_H\_\_

#ifndef \_\_PARAMETROS\_H\_\_

#define \_\_PARAMETROS\_H\_\_

#include <iostream>

**using namespace std**;

/\*

Definicicion de tipo de dato Parametro para cargar items de archivo

Atributos:

s, p, a, PosXE, posYE,iM, vM, iB, vB, iP

Axiomas

\* codItem > 0, existente

\* cantidad < 100

\*/

/\* Tipo de estructura de Comanda \*/

**typedef struct** Parametros{

**int** s;

**int** p;

**int** a;

**int** posXE;

**int** posYE;

**int** iM;

**int** vM;

**int** iB;

**int** vB;

**int** iP;

}Parametros;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**int** getS(Parametros &parametros);

**int** getP(Parametros &parametros);

**int** getA(Parametros &parametros);

**int** getPosXE(Parametros &parametros);

**int** getPosYE(Parametros &parametros);

**int** getIm(Parametros &parametros);

**int** getVm(Parametros &parametros);

**int** getIb(Parametros &parametros);

**int** getVb(Parametros &parametros);

**int** getIp(Parametros &parametros);

**void** setS(Parametros &parametros, **int** s);

**void** setP(Parametros &parametros, **int** p);

**void** setA(Parametros &parametros, **int** a);

**void** setPosXE(Parametros &parametros, **int** posYE);

**void** setPOSYE(Parametros &parametros, **int** posXE);

**void** setIm(Parametros &parametros, **int** im);

**void** setVm(Parametros &parametros, **int** vm);

**void** setIb(Parametros &parametros, **int** ib);

**void** setIp(Parametros &parametros, **int** ip);

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

/\*

PRE: Mina existente y cargada

Post: Muestra por pantalla los atributos

\*/

**void** toString(Parametros &parametros);

#endif // \_\_PARAMETROS\_H\_\_

#ifndef \_\_POSICION\_H\_\_

#define \_\_POSICION\_H\_\_

#include <iostream>

**using namespace std**;

/\*

Definicion de tipo de dato Posicion para referencias posicion X e Y de un elemento

Se busca representar univocamente una posicion en el juego

Atributos:

\* X -> posicion eje X

\* Y -> posicion eje Y

Axiomas

\* X >= 0

\* Y >= 0

\*/

/\* Tipo de estructura de Comanda \*/

**typedef struct** Posicion{

**int** x;

**int** y;

}Posicion;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**int** getX(Posicion &posicion);

**int** getY(Posicion &posicion);

**void** setX(Posicion &posicion, **int** x);

**void** setY(Posicion &posicion, **int** y);

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

/\*

PRE:

Post: inicializa parametros de posicion, x=0, y=0

\*/

**void** crearPosicion(Posicion &posicion);

/\*

PRE: posicion creada

Post: libera recursos solicitados por posicion

\*/

**void** eliminarPosicion(Posicion &posicion);

/\*

PRE: posicion creada

Post: libera recursos solicitados por posicion

\*/

**void** moverPosicion(Posicion &posicion, **int** x, **int** y);

/\*

PRE: posicion creada

Post: verifica si la posicion es valida en el juego, es

decir verifica que sea una posicion posible en el juego,

devuelve false si esta fuera del rango del juego

\*/

**bool** esValidaPosicion(Posicion &posicion);

#endif // \_\_PARAMETROS\_H\_\_

#ifndef TERRENO\_H\_\_

#define TERRENO\_H\_\_

#include <iostream>

**using namespace std**;

**typedef struct** Terreno{

Locomotora locomotora;

Lista minas;

Lista estaciones;

Lista moneda;

Lista bandidos;

}Terreno;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//getters y setters

**void** setLocoomotora(Vagon& vagon,**int** pesoMaximo);

**int** getPesoMaximo(Vagon& vagon);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRIMITIVAS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//pre:

//post: se inicializan los parametros de Terreno

void crearTerreno(Terreno& terreno);

//pre:Tiene que existir el terreno

//post:Se elimina el terreno en cascada con los atributos internos

//se usa al cerrar el juego

void eliminarTerreno(Terreno& terreno)

//pre:Terreno tiene que estar creado e inicializado

//post: se posiciona la locomotora en el terreno

void aparecerLocomotora(Terreno& terreno);

//pre:Terreno tiene que estar creado e inicializado

//post: se posiciona una mina en el terreno

void aparecerMina(Terreno& terreno);

//pre:Terreno tiene que estar creado e inicializado

//post:se posiciona una estacion en el terreno

void aparecerEstacion(Terreno& terreno);

//pre:Terreno tiene que estar creado e inicializado

//post:se posiciona una moneda checkeando si es posible en el terreno

void aparecerMoneda(Terreno& terreno);

//pre:Terreno tiene que estar creado e inicializado

//post: se posiciona una moneda checkeando si es posible en el terreno

void aparecerBandido(Terreno& terreno);

#endif // TERRENO\_H\_\_

#ifndef \_vagon\_h\_

#define \_vagon\_h\_

#include "posicion.h"

**enum** item{//ITEMS

ORO,PLATA,BRONCE,

PLATINO,ROCA,CARBON,

SIN\_DEFINIR

};

**typedef struct** Vagon{

**int** pesoMaximo;

item tipoVagon;

Posicion posicion;

//Lista cajasAlmacenadas ?

}Vagon;

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Axiomas\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//el vagon incia con tipoVagon SIN\_DEFINIR

//el vagon inicia con un pesoMaximo definido por 5kg\*cantMonedas

//el vagon solo puede almacenar items de su mismo tipo

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Primitivas\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

//getters y setters

**void** setPesoMaximo(Vagon& vagon,**int** pesoMaximo);

**int** getPesoMaximo(Vagon& vagon);

**void** setTipoVagon(Vagon& vagon,item tipoVagon);

item getTipoVagon(Vagon& vagon);

**void** setPosicion(Vagon& vagon,Posicion posicion);

Posicion getPosicion(Vagon& vagon,Posicion posicion);

//pre:

//post: se inicializan los parametros de Vagon

**void** crearVagon(Vagon& vagon);

//pre:vagon creado

//post:se liberan los recursos usados en Vagon

**void** eliminarVagon(Vagon& vagon);

//pre:vagon creado

//post:el vagon se movera en base a los eventos que genera la locomotora

**void** avanzar(Vagon& vagon,Posicion& posicion);

//pre: vagon creado y con capMaxima total disponible o el (tipoVagon tiene que ser el mismo que caja

// y tener suficiente capacidad para poder almacenarlo)

//post: guarda una caja en un vagon,si no se cumple ni una de las 2 condiciones anteriores no hace nada

**void** almacenarItem(Vagon& vagon,Caja& caja);

//pre:Vagon creado

//post; devuelve '1' si el vagon esta vacio osea si no tiene ni un item o '0' si no esta vacio

**bool** vagonVacio(Vagon& vagon);

//pre: vagon creado y asignado con 1 o mas items

//post: devuelte 1 si el vagon es del tipoItem o 0 si no es del tipoItem

**bool** esVagonTipo(Vagon& vagon,item tipoItem);

#endif // \_vagon\_h\_

#ifndef \_\_BANDIDO\_H\_\_

#define \_\_BANDIDO\_H\_\_

#include <iostream>

#include "locomotora.h"

**using namespace std**;

/\*

Definicion de tipo de dato Bandido para representar el rival del juego

Atributos:

\* cantidad

\* posicion

\* tiempo Vida

\* intervalos que pasaran hasta aparicion

Axiomas

\* cantidad < 100

\* posicion -> x e y > 0

\* tiempo vida > 0 [0..VB]

\* intervalos > 0 [0..IB]

\*/

/\* Tipo de estructura de Bandido \*/

**typedef struct** Bandido{

**int** cantidad;

Posicion posicion;

**int** tiempoVida;

**int** intervaloHastaAparicion;

}Bandido;

**/\*\*\*\*\*\*\*\*\*\*\*\*GETTERS AND SETTERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**int** getCantidad(Bandido &bandido);

**int** getPosicion(Bandido &bandido);

**int** getTiempoVida(Bandido &bandido);

**int** getIntervaloHastaAparicion(Bandido &bandido);

**void** setCantidad(Bandido &bandido, **int** cantidad);

**void** setCantidad(Bandido &bandido, Posicion posicion);

**void** setCantidad(Bandido &bandido, **int** tiempoVida);

**void** setCantidad(Bandido &bandido, **int** intervaloHastaAparicion);

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

/\*

PRE:

Post: inicializa parametros de bandido, cantidad=0; posicion (0,0) tiempoVida y intervaloHastaAparicion random

\*/

**void** crearBandido(Bandido &bandido);

/\*

PRE: comada creada

Post: libera recursos solicitados por bandido

\*/

**void** eliminarBandido(Bandido &bandido);

/\*

PRE: Bandido existente y cargada

Post: Muestra por pantalla los atributos

\*/

**void** toString(Bandido &bandido);

/\*

PRE: Bandido existente

Post: "roba" el item de la locomotora

\*/

**void** robarItem(Bandido &bandido, Locomotora& locomotora);

/\*

PRE: Bandido existente

Post: si la locomotora queda en 0, destruye el vagon

\*/

**void** destruirVagon(Bandido &bandido, Locomotora& locomotora);

#endif // \_\_PARAMETROS\_H\_\_

# Segunda Entrega

## Cambios en estrategia final

De acuerdo a lo detallado en la primera entrega se utilizaron listas, pilas y colas de tipo void\*, para eliminar los TDA instanciados en los nodos de la listas se implemento el pasaje por parámetro de un destructor al momento de la creación de la lista y al eliminar el nodo se llama a esta función.

El grueso de la lógica del juego se encuentra contenida en terreno, en cuanto a la logica grafica se desarrolla en el TDA Juego.

A su vez cada TDA tendrá contenido sus imágenes y texturas, sabiendo que debe agregar al render para representarse.

Algunos aspectos a considerar durante el desarrollo fueron los siguientes:

* Los intervalos son contados en Terreno y para temporizar cada dibujo en pantalla al final de cada ciclo se realiza un sleep según parámetro “S” cargado de archivo Parametros.h
* Para la generación de monedas se utilizo la función random y al momento de la creación de una moneda en pantalla se setea una variable de terreno (intevaloAparicionProximaMoneda) que indica cuando será la aparición de la próxima.
* Para generación de bandidos se usa el mismo concepto pero con otra variable de terreno (intevaloAparicionProximoBandido)
* Para la generación de minas se uso un contador para saber que secuencia correspondia crear, cuando se llega al final se reinicializa dicho contador.
* Para acelerar el proceso de verificación de colisiones se uso una matriz para no recorrer cada lista cada vez que se avanza la locomotora, la matriz de char contiene los valores T=Terreno, M=Mina, m=moneda, B=Bandido. Según el valor que se tenga en la posición de la matriz y este la locomotra se efectua una acción correspondiente.

## Pruebas

Para los modulos individuales se realizo un Prueba.h y Prueba.cpp donde se realizaron pruebas de los modulos individuales.

Respecto a los archivos de configuración de prueba se encuentran en la carpeta “Pruebas” con una numeración adicional al nombre para identificar las ternas de pruebas.

* Se prueba un caso con una gran densidad de monedas y de bandidos con largos tiempos de duración y cortos intervalos de aparición.
* Se prueba una comanda simple con solo un elemento de prueba y una comanda con varios ítems a recoletar.