**Estrategia de resolución:**

**Índice de contenidos:**

-Página 2: Estructura a utilizar.

-Página 3: Diagrama TDA y sus relaciones.

-Página 4-5: Estrategia de resolución de operaciones.

-Página 6: División de trabajo y tareas a realizar.

**Estructura a utilizar:**

-Luego de analizar la consigna en cuanto al trabajo a desarrollar “Clash of UNLa”, se estableció que la mejor forma de llevar a cabo este desarrollo de una forma óptima, es utilizando principalmente el concepto de la estructura “Lista”, ya que esta es una estructura que nos permite acceder a un dato sin importar el orden o posición en el que este, a comparación de las estructuras “Pilas” o “Colas” que permiten acceder al último o primer dato. En principio se utilizara una lista conforme a su TDA.  
También se utilizara como posibilidad la estructura de arrays para simplificar alguna operación con el manejo de datos.

**Modificación de resolución final:**

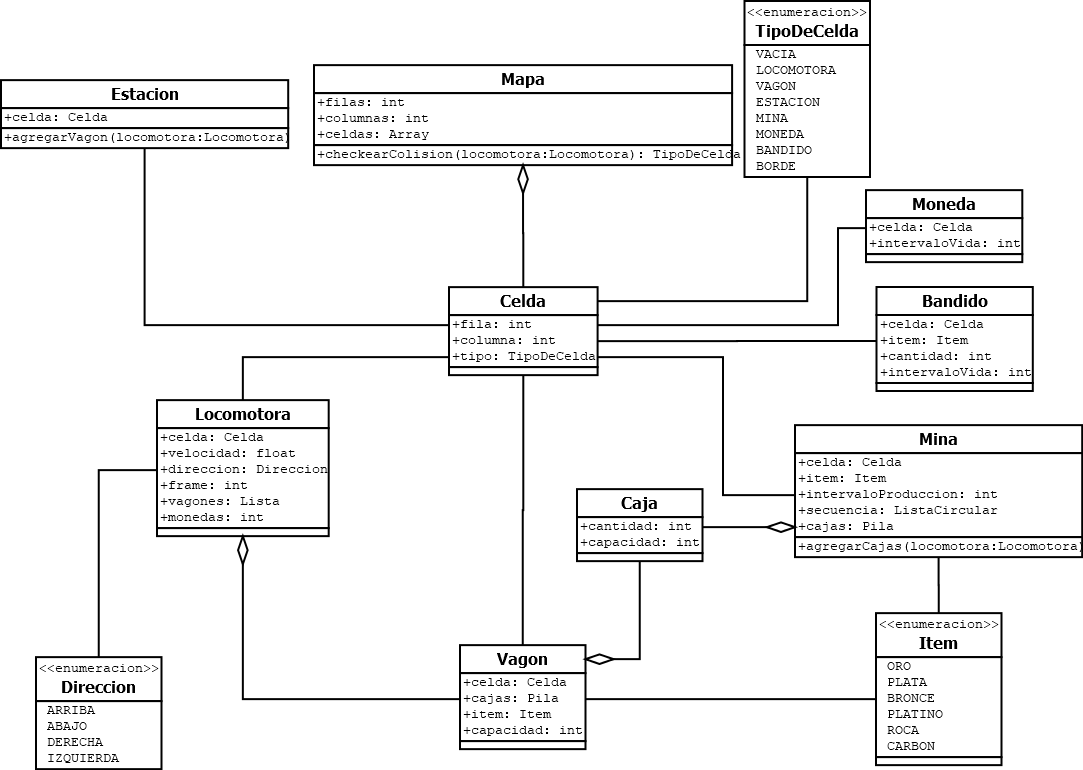
-Utilización de listas para la creación y ubicación del vehículo principal o locomotora, el manejo de colisiones (tanto para las minas como para los bordes o fin del mapa), la lectura de comanda, y la ubicación de las minas. Los vagones van siendo creados en forma de pila.

-También se utilizó listas para la generación de Monedas y Bandidos, estos se van agregando cada cierto tiempo a la lista, así como también se van borrando sin ningún orden

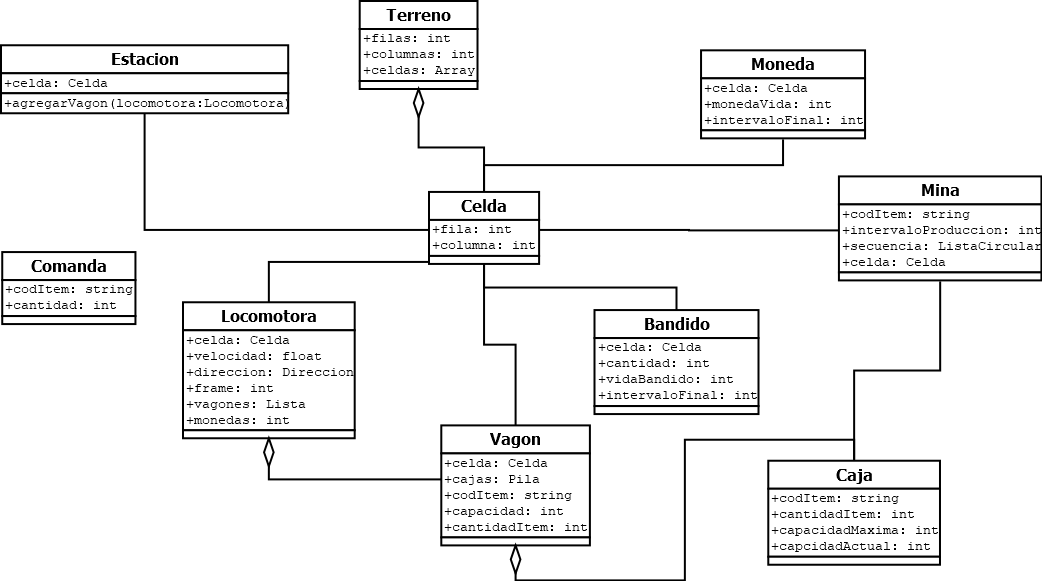
-Se utilizó el concepto de colas para poder generar las cajas que contienen los ítems.

-Se utilizó el concepto de pilas para que la extracción de minerales del bandido se haga según la forma propuesta

**Diagrama *TDA (relaciones):***



**Modificación de resolución final:**



**Estrategia de resolución de operaciones:**En cuanto a la resolución de operaciones, se analizó, debatió y acordó lo siguiente:   
-Se carga la información de las minas (tipo de ítems e intervalos de producción) y las estaciones (posiciones).

-Creemos que será necesario el uso de pilas para los bandidos y monedas, para las validaciones de id de cada uno. En caso contrario, un mismo bandido que pide lo mismo puede estar en más de un lugar a la vez.

-Se definirá en un archivo de texto, la meta de ítems que la locomotora debe obtener. Cuando el juego inicia, se debe verificar si el jugador obtuvo esos ítems y se le notifica al usuario QUE ítems son los que tiene que obtener o los faltantes.

-Cada cierta cantidad de tiempo se generará un bandido o cierta cantidad de vagones, estos no pueden generarse cerca de la locomotora (por ejemplo, a 3 bloques de distancia en cualquier dirección), ni dentro de algún vagón.

-Se dificultará la producción de las minas, hay que usar varios if en un orden específico para chequear si hay primero un vagón de ese ítem disponible, y luego si tiene el peso suficiente para las cajas que haya producido en ese intervalo de tiempo.

-Sera necesario un campo que posea todo el tiempo la posición de la locomotora (o cualquier identidad que necesitemos).

En conclusión, esta será la estrategia para poder resolver los distintos puntos del trabajo a desarrollar, esto no quiere decir que durante el mismo no se presenten modificaciones al respecto y que se deba modificar o incluso agregar algo en dicha estrategia de resolución. En ese caso, se expresará en las siguientes entregas.

**Modificación de resolución final:**

-Se descartó Item, Sprite, Ventana, Celda, Mapa como TDA utilizable y/o se combinaron con otros existentes (Game, Terreno).

-Decidimos hacer la carga y lectura de todos los objetos “físicos” (Minas, Estación, entre otros) que se pueden encontrar en el terreno desde Game.cpp, se validan y luego se imprimen en el main.cpp (función correrGame).

-El archivo de comanda contiene los objetivos principales que tiene que cumplir el jugador para ganar, teniendo que pasar por cada una de las minas (también con un archivo específico para los tiempos de generación de recursos) para sumar minerales viéndose afectado por la aparición aleatoria de un bandido que le roba cierta cantidad de recursos.

-Varias cadenas de ifs fueron necesarias para comprobar la ubicación del vehículo en movimiento para verificar si esa celda estaba ocupada (colisiones) o si el vehículo se detuvo tanto en una mina como en la estación.

-Utilizamos la ventana de línea de comandos para mostrar si el tren fue destruido (al salir del terreno o chocar consigo mismo) y otras funciones.

- No fue necesario ningún menú para crear y/o validar algún archivo para la carga de minas.

-Usamos la función random para las monedas y bandidos, generamos un IM (intervalo Moneda) VM(vida Moneda) e IB (intervalo Bandido) VB (vida Bandido) al azar , como valor de inicio para la aparición de estos sprites, más tarde se utilizara durante todo el programa para ir generando más bandidos y monedas aleatoriamente, con la formula intervaloFinal=IM + VM(vida Moneda) determinamos el tiempo de expiración de Moneda, mismo caso que Bandido. Además de utilizarse para los drops de las monedas y bandidos.

**División de tareas/cronograma:***Orden y Tareas principales:*   
-Definir TDA y primitivas  
-Implementar listas  
-Definir estructuras y cargar archivos  
-Armar menú con todas las operaciones a llevar a cabo  
-Desarrollar las validaciones principales

Las tareas se dividirán equitativamente entre los integrantes, en caso de haber modificaciones se especificará en la segunda entrega.

**Modificación final:**

El orden de las tareas fue bastante variable, comenzamos con la construcción de los TDA base entre los 4 integrantes del grupo, continuando con las estructuras que íbamos necesitando a medida que avanzábamos con el proyecto, comenzando con la creación de la ventana y el mapa base.

Fuimos agregando el vehículo y sus movimientos, luego minas para poder testear las colisiones, volviendo atrás para modificar ciertas primitivas que quedaron incompletas o no existentes.

A medida que avanzaba el proyecto nos dividimos el trabajo dependiendo lo que podía hacer cada uno (lógica, armado de estructuras, validación, archivos, etc) utilizando un sistema de control de versiones para tener un desarrollo iterativo.