Trabajo práctico 3: **Algo2Landia**

Normativa

Límite de entrega: Domingo 14 de Noviembre, 23:59hs.

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://campus.exactas.uba.ar`)

Versión: 1.0 del 6 de octubre de 2021

El objetivo de este TP es implementar en C++ todos los módulos correspondientes al diseño presentado en el TP2. El código que entreguen debería respetar el diseño propuesto en el TP 2 de la manera más fiel posible. Obviamente se permite y se espera que corrijan todos los potenciales *bugs* que puedan llegar a encontrar en el diseño. Las implementaciones deben cumplir con las complejidades definidas en su solución del TP 2, incluyendo las restricciones de complejidad establecidas en el enunciado.

Código producido

La resolución debe tener un archivo `.h` y `.cpp` por cada módulo del TP 3 (o eventualmente un archivo `.hpp` si se trata de un módulo paramétrico que se implemente con `templates`). Estos archivos deberán ubicarse en el directorio `src`, respetando el esqueleto disponible en la página de la materia.

Código de la cátedra y tests

Como parte del enunciado, la cátedra provee un **esqueleto de TP3**. El esqueleto tiene la misma estructura de directorio que los talleres, con un directorio `src` para los archivos fuente y un directorio `tests` para el código de los tests. Proveemos un conjunto de casos de test que deberán ser pasados con éxito. Además, se recomienda *escribir sus propios test de unidad* para cada una de las clases que implementen.

Los archivos `src/aed2_simulacion.{h,cpp}` y `src/aed2_mapa.{h,cpp}` provistos como parte del esqueleto del TP incluye las interfaces para los módulos Simulacion y Mapa que se utiliza en los tests. Deben completar estos archivos agregando instancias de las clases diseñadas por ustedes en la parte privada de la clase `aed2_simulacion`, e implementando los métodos de forma tal que utilicen la interfaz provista por sus propios módulos.

El archivo `src/Tipos.h` define algunos tipos auxiliares y renombres de tipos (como el tipo `Coordenada`, `Direccion`, `Color` o `TipoCasillero`).

La adaptación de la interfaz de sus módulos a los requeridos en `aed2_simulacion` y `aed2_mapa` puede conllevar operaciones con un costo no inmediato (ej. copiar un conjunto a una lista, recorrer un diccionario, etc.). Los requisitos de complejidad a cumplir aplican solamente a las funciones de la interfaz de los módulos. Los costos asociados a la traducción de su interfaz a la nuestra no tienen restricciones.

Importante: sugerimos no implementar la lógica del juego en la clase `aed2_simulacion`, sino hacerlo en una clase independiente `Simulacion` que respete el diseño hecho por ustedes en el TP2. La clase `aed2_simulacion` únicamente debe hacer las veces de "fachada" que delega todos los mensajes que recibe a la clase `Simulacion`. Así, la implementación de todos los métodos de la clase `aed2_simulacion` debería ser breve (ej. una o dos líneas). Este mismo criterio aplica para la clase `aed2_mapa` y el módulo `Mapa`.

Módulos básicos

Pueden utilizar las siguientes clases de la STL de C++ para los respectivos módulos básicos:

Módulo	**Clase**
Lista Enlazada	`std::list`
Pila	`std::stack`
Cola	`std::queue`
Vector	`std::vector`
Diccionario Lineal	`std::map`
Conjunto Lineal	`std::set`

Entrega

Para la entrega deben hacer `commit` y `push` de todo el esqueleto, incluyendo el código fuente (`*.cpp`, `*.h`, `*.hpp`), los tests, y el archivo `CMakeLists.txt` en el repositorio **grupal** en el directorio `tpg3/`. No incluir los archivos binarios generados por el compilador (`*.o`, `*.a`, `*.exe`) en el repositorio.

Fechas de entrega

El TP 3 tiene como fecha de entrega el Domingo 14 de Noviembre hasta las 23:59. Su devolución será realizada el 24 de Noviembre, durante el horario de consultas. Su recuperatorio se podrá entregar hasta el 1ero de Diciembre, hasta las 23:59.

Rubricas

Agregamos a continuación rúbricas que exponen qué se espera de la entrega. Las mismas presentan una serie criterios con los que se evaluarán las producciones entregadas. En términos generales, se considera que entregas con soluciones que solo logren los criterios parcialmente deberán ser reentregados con correcciones en estos aspectos en particular.

Por ser criterios generales, pueden no cubrir todos los detalles relacionados con este enunciado. No obstante buscamos que sean lo más completas posibles. Esperamos que las mismas les sirvan de orientación para la resolución del TP.

	Logra Totalmente	Logra	Logra Parcialmente	No Logra
Correctitud	La implementación respeta la especificación y satisface los tests de la cátedra.	La implementación respeta la especificación y satisface los tests de la cátedra.	La implementación no respeta la especificación y o no satisface todos los tests de la cátedra.	La implementación no respeta la especificación y o no satisface todos los tests de la cátedra.
Complejidad	Se cumplen todas las restricciones de complejidad del TP de diseño, teniendo en cuenta los costos de copia de variables y las complejidades de las estructuras auxiliares.	Se cumplen todas las restricciones de complejidad del TP de diseño, teniendo en cuenta los costos de copia de variables y las complejidades de las estructuras auxiliares.	No se cumplen todas las restricciones de complejidad del TP de diseño.	No se cumplen varias de las restricciones de complejidad del TP de diseño.
Uso de memoria	La implementación no presenta malos usos de memoria (pérdidas, doble deletes, escritura/accesos inválidos).	La implementación presenta malos usos de memoria (pérdidas, doble deletes, escritura/accesos inválidos).	La implementación presenta malos usos de memoria (pérdidas, doble deletes, escritura/accesos inválidos).	La implementación presenta malos usos de memoria (pérdidas, doble deletes, escritura/accesos inválidos).
Prolijidad	El código es legible, está bien formateado, los nombres de variables y funciones son declarativos.	El código es mayormente legible, está bien formateado, los nombres de variables y funciones comunican la idea aunque quieren contexto.	Hay secciones del código donde es necesario leerlo al detalle para entender que función cumple.	Hay secciones del código que no pueden comprenderse.
Modularización (clases)	Los módulos sólo se comunican mediante su interfaz pública. No exhiben su representación mediante punteros o referencias (la referencia explicita detalles de implementación).	Los módulos sólo se comunican mediante su interfaz pública. No exhiben su representación mediante punteros o referencias (la referencia explicita detalles de implementación).	En alguna circunstancia se necesitan conocer la representación interna de otros módulos. Se exportan punteros o referencias que explicitan detalles de implementación.	Varios módulos necesitan conocer la representación interna de otros módulos. Se exportan punteros o referencias que explicitan detalles de implementación.
Modularización (funciones)	Las funciones resuelven problemas acotados y entendibles. Lógicas muy sofisticadas o extensas se dividen en funciones auxiliares.	Alguna función es particularmente larga pero mediante auxiliares es fácil de leer.	Alguna función es particularmente larga y difícil de entender.	Es común tener funciones súmamente extensas que resuelven diversos problemas simultáneamente.
Buenas prácticas	El código no presenta situaciones que constituyen malas prácticas en cuanto a declaratividad, correctitud o eficiencia que sean importante aprender como corregir.	El código no presenta situaciones que constituyen malas prácticas en cuanto a declaratividad, correctitud o eficiencia que sean importante aprender como corregir.	El código presenta alguna situación que constituye malas prácticas.	El código presenta numerosas situaciones que constituye malas prácticas.