Trabajo Práctico n2

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno:	Saluuzi, Luca
Número de padrón:	108088
Email:	lsalluzzi@fi.uba.ar

Como jugar

Para jugar con los objetos brindados por la catedra se puede ejecutar el comando make valgrind-escape_pokemon. Sino, se puede compilar manualmente utilizando `gcc -std=c99 -Wall -Wconversion -Wtype-limits -g -Werror escape_pokemon.c src/*.c -o escape_pokemon` y luego ejecutado `./escape_pokemon ejemplo/objetos.txt ejemplo/interacciones.txt`. Reemplazando los .txt por los archivos de preferencia.

1. Introducción

En este TP2 se nos dio mucha libertad a la hora de implementar lo pedido. De esta forma se evaluó si el alumno conocía o no los temas impartidos a lo largo del cuatrimestre y si era capaz de moverse con libertad entre ellos.
Se le pidió, entonces, reescribir el TP1 reemplazando los vectores dinámicos por otras estructuras de datos y, adicionalmente, implementar funciones nuevas que permitieran ejecutar interacciones y desarrollar adecuadamente un "loop jugable".

Elección de TDAs

Decidí reemplazar mis vectores dinámicos con hashes. De este modo obtenía un acceso muy directo y sencillo al dato (utilizando como clave el nombre del objeto o el string "objeto1verboobjeto2"). Para esto fue necesario eliminar ciertas partes del TDA que sobreescribían elementos con la misma clave. En vez de hacer eso, el hash actual almacena en una misma lista interacciones de igual clave y a la hora de ser llamadas pueden ser ejecutadas una atrás de otra.
Luego, en cuanto a la estructura general del juego, decidí implementar un struct sala con su hash de objetos e interacciones y un struct jugador, anidado en sala. Esta segunda estructura contienes los objetos poseídos y los objetos conocidos por el jugador.
Cuando el susodicho descubre un objeto, este se pasa del hash de objetos de la sala al de objetos_conocidos ubicado en el jugador. Si el usuario agarra el objeto, este se mueve nuevamente de su hash de objetos conocidos a su hash de objetos poseídas.

Resolución de pruebas

En cuanto a las pruebas, hubo que hacer modificaciones:
Para resolver las pruebas de caja blanca del TP1, agregué (con permiso de mi correctora, o sea, vos), la estructura de la sala al entorno de pruebas mediante un .h privado (a `escape_pokemon.c` se le sigue pasando exclusivamente `sala.h`).
Otra prueba a corregir fue aquella que pedía todos los objetos de la sala y los comparaba con un vector. El orden en el que estos aparecen en el hash no es el mismo que el de .dat o .txt, porque el criterio de ordenamiento es otro y esta regido por la funcion hash. Para arreglar esta prueba, simplemente cambié el orden del vector `esperados` (que es el que se compara con las claves del hash).

Detalles de la implementación

`hash_obtener_claves` Esta es una funcion que recorre todo el hash obteniendo del par `<clave,elemento>` la clave y guardándola en un vector dinámico. Me parece oportuno aclarar que no reutilicé `hash_con_cada_clave` por un tema de simplicidad a la hora de programar. Para utilizar `con_cada_clave` habría que haberle pasado como parámetro un struct con un vector e información extra, también se podría haber modificado la funcion anteriormente mencionada para que reciba más parámetros la funcion booleana auxiliar. Pero me pareció más sencillo crear una funcion desde cero que cumpliera con lo que yo estaba necesitando en ese momento.

La mayoría de las funciones tienen un funcionamiento similar, intercambian, pasan o eliminan información de los 3 hashes de objetos (dos del jugador, uno de la sala).
En `ejecutar_interaccion` por ejemplo, se comprueba que los objetos estén en dentro

del inventario del jugador. Se concatenan los 3 strings objeto1, verbo, objeto2 (de ser necesario este segundo objeto). Y luego se busca la interacción dentro del hash correspondiente. Si se encuentra, se analiza que tipo de acción tiene y se la ejecuta sumando uno a la cantidad de interacciones ejecutadas. Se elimina la interacción actual y se busca luego la próxima que cumpla con la clave correspondiente, repitiendo el proceso de análisis de tipo de acción. La funcion termina cuando no existen más interacciones con la clave otorgada. Se devuelven, finalmente, la cantidad de interacciones ejecutadas.