FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

TP2 Escape Pokemón

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno: Martín Abramovich
Padrón: 108762
Email: mabramovich@fi.uba.ar

Contenido

1. Introducción

Para este trabajo se reutiliza la mayor parte del TP1 implementado al principio del cuatrimestre. Para esta segunda parte se pide incorporar algunos cambios, incorporando además nuevas funciones que permitan desarrollar la lógica del juego.

2. Detalles de Implementación

Al haberse movido la estructura al archivo sala.c, ya no es posible acceder a su estructura por fuera de este archivo.
Para la estructura de sala, decidí agregar las interacciones en una lista, y los objetos, objetos conocidos y objetos poseídos en un hash. Estos últimos porque sabía que iba a tener que obtener elementos varias veces y esto era sencillo de hacer con la función hash_obtener. Además agregué un bool *escapó* que me fue útil para el desarrollo de algunas funciones.

Al realizar los cambios, tres pruebas del tp1 fallaron.

- Dos estaban relacionadas a que las pruebas intentaban acceder al struct de sala, lo cual ya no era posible por pasar a ser una estructura privada. Estas pruebas, que antes funcionaban accediendo a sala->cantidad, ahora funcionan porque incluí en mis pruebas lista.h, hash.h y un aux.h en el que copié el struct de sala para poder hacer las pruebas pertinentes. Una vez hecho esto, pude resolver las pruebas con hash_cantidad(sala->objetos) y lista_tamanio(sala->interacciones).

- La prueba "todos los nombres de objetos son los esperados" de la función del vector de nombres tampoco funcionaba al hacer los cambios ya que compara la posición de dos vectores que, al usar hash, ya no mantienen el mismo orden y por ende la comparación falla. Para solucionarlo, en vez de comparar si coincidian los objetos en las posiciones de los vectores, verifiqué que los objetos del vector de esperados estén en alguna posición del vector objetos2.

```c
for (int i = 0; i < cantidad; i++){
    if (strcmp(objetos2[i], esperados[i]) == 0)
        comparaciones_exitosas++;
}
```

Antes

```c
for (int i = 0; i < cantidad; i++){
    for (int j = 0; j < cantidad; j++){
        if (strcmp(objetos2[i], esperados[j]) == 0)
        comparaciones_exitosas++;
    }
}
```

Después

2.1. Adaptación de funciones viejas e implementación de funciones nuevas

Antes de arrancar con la implementación de las nuevas funciones, me aseguré de adaptar las funciones del tp1 para que funcionen con la nueva estructura de sala y sus respectivos tdas.

Para crear la sala, cree los hash y las listas correspondientes, abrí los archivos, y por cada linea leida del archivo de objetos, los fui insertando en el hash objetos.
De igual manera, por cada interaccion leida del archivo de interacciones, las fui insertando en la lista.

Para obtener el nombre de los objetos, creé un vector al que se le guardaron todos los objetos del hash de objetos, utilizando la funcion *hash_con_cada_clave.*
Usé esta misma lógica para *sala_obtener_nombre_objetos_conocidos* y *sala_obtener_nombre_objetos_poseidos.*

Con *sala_agarrar_objeto* utilicé la función *hash_contiene* para verificar que el objeto a agarrar no haya sido ya agarrado y para verificar que el objeto a agarrar es parte de los objetos conocidos. De no ser poseido, ser conocido y además asible, inserto con *hash_insertar* en el hash de poseidos y quito con *hash_quitar* en el hash de conocidos.

Para la función *sala_describir_objeto* también utilicé *hash_contiene* y *hash_obtener.* Si el hash de conocidos contiene el objeto, entonces lo obtiene y devuelve su descripción. De no ser así, chequea lo mismo en el hash de poseidos.

A la hora de desarrollar *sala_ejecutar_interacción* utilicé un iterador para la lista de interacciones que verificara si la interacción recibida se encuentra en la lista. De ser así, según el tipo de acción de la interacción es que la ejecuta:
- Mostrar mensaje muestra un mensaje.
- Descubrir objeto verifica si se conoce el objeto de la interacción y el objeto_parámetro de la interacción y, de ser así, obtiene del hash de objetos el objeto de la acción y lo inserta en el hash de conocidos.
- Reemplazar objeto quita el objeto_parametro de la interacción del hash en el que se encuentre e inserta el objeto de la acción.
- Eliminar objeto hace lo mismo que reemplazar pero no inserta nada en su lugar.

Sala_es_interaccion_valida la adapté con las funciones de lista y recorre la lista fijandose si la interacción recibida coincide con alguna de las interacciones de la lista.

Sala_escape_exitoso retorna el valor del campo escapo del struct sala, siendo este verdadero o falso.

En sala_destruir destruyo las listas y los hash del struct sala y luego libero la sala con free.

3. Aclaraciones del juego y del trabajo.

Para jugar se deben ingresar como máximo 3 comandos, siendo siempre el primero una acción y de existir el 2do y/o el 3ro, objetos.
Es decir, la implementación no garantiza el funcionamiento del juego si se agregan artículos o conectores entre las palabras.
Además, el juego distingue entre mayúsculas y minúsculas, por ende se solicita que todo sea ingresado en minúscula.

El juego se compila con la siguiente linea:
gcc -g -O1 -std=c99 -Wall -Wconversion -Wtype-limits -Werror src/*.c escape_pokemon.c -o escape_pokemon

Y se ejecuta bajo:
./escape_pokemon ejemplo/objetos.txt ejemplo/interacciones.txt

Lamentablemente por una cuestión de tiempos no llegué a realizar las pruebas de todas las funciones que implementé. Al haber tenido a mano las pruebas de la cátedra y al haber pasado con éxito, prioricé la realización del main y del informe.