Informe TP2 ~ ESCAPE POKEMON

Padrón: 108951

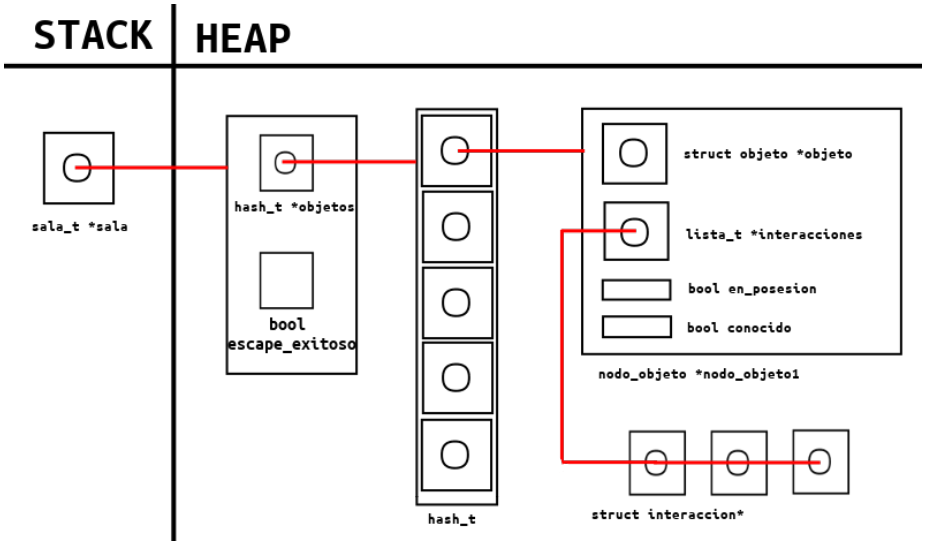
Nombre: Luciano Andrés Lorenzo

Implementación

Mi idea principal para poder agregar los tda implementados durante la materia, fue que la sala consista principalmente de un hash, siendo cada clave el nombre de cada objeto. Y dentro de cada elemento del hash, se encuentren todas las interacciones que se puedan agregar con ese objeto (el principal de la interacción) en formato de lista enlazada, el propio struct objeto y dos booleanos para saber si es un objeto conocido o poseido.

```c
typedef struct nodo_objeto {
    struct objeto *objeto;
    lista_t *interacciones;
    bool en_posesion;
    bool conocido;
} nodo_objeto_t;

struct sala {
    hash_t *objetos;
    bool escape_exitoso;
};
```



Elegí guardar las interacciones en listas porque pueden haber interacciones con nombres repetidas para un mismo objeto, y que hagan cosas diferentes. Por lo que no había otra opción más que usar una lista para guardarlas todas.

Con esta implementación se encuentra muy fácil cada elemento que quiero usar, ya que al tener una tabla de hash no tengo que iterar nada y puedo conseguir cualquier objeto en tiempo constante. Además tener los booleanos en el nodo del objeto, hace que sea muy fácil interactuar con los objetos y saber cuando puedo o no realizar una acción con ellos, sin necesidad de iterar otras listas para poder saber si el usuario ya posee el objeto o lo conoce.

La única desventaja que le encuentro a mi implementación es que para encontrar todos los objetos conocidos o poseidos, tengo que iterar todo el hash completo con todos los objetos. Pero para lograr eso reutilicé la misma función de obtener nombres, pero generalicé la función que agrega cada objeto del hash al vector, para que pueda recibir como parametro si quiero que sean solo los objetos conocidos o en posesión o todos. Para use un struct auxiliar que guarda el vector de char y la condicion.

Compilación

Para la compilación de mi programa creé un makefile que contiene los comandos para correrlo, los cuales son:

- Para compilar la sala de escape: `make escape`.
- Para ejecutar la sala de escape: `make run`.
- Para compilar las pruebas: `make pruebas` para compilar.
- para correr las pruebas con valgrind `make valgrind-pruebas`.

Cambios en las pruebas

```
pa2m_afirmar(sala->cantidad_objetos == 9, "Se leyeron 9 objetos");
pa2m_afirmar(sala->cantidad_interacciones == 9, "Se leyeron 9 interacciones");
```

Esta prueba tuvo que ser eliminada, porque es una prueba de caja blanca para el anterior `struct sala`, el cual que puede ser modificado en algún momento, y no pasaría las pruebas.

```
char **objetos2 = sala_obtener_nombre_objetos(sala, &cantidad);
const char *esperados[] = { "habitacion",    "mesa",  "interruptor", "pokebola",
                "cajon", "cajon-abierto", "llave", "anillo",         "puerta" };

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

    pa2m_afirmar(comparaciones_exitosas == cantidad, "Todos los nombres
                    de objeto son los esperados");
```

Esta prueba tuvo que ser eliminada ya que asume que los objetos esperados salen en un orden específico, cuando el contrato de la función en ningún momento dice que tengan que salir en un orden específico.

Fueron agregadas pruebas para las nuevas funciones agregadas en sala.h, que prueban la ejecución de interacciones, descripcion y agarre de objetos hasta que se sale de la sala.