Trabajo Práctico — TP2: Escape Pokémon

[7541/9515] Algoritmos y Programación II

Primer cuatrimestre de 2022

Alumno:
Vainstein Aranguren, Tomás

Número de padrón:
109043

Email:
tvainstein@fi.uba.ar

1. Introducción

La idea general del TP es cargar la información de dos archivos utilizando estructuras de datos para luego poder imprimir la información por pantalla y, por último, liberar la memoria utilizada.

2. Teoría

- La definición de la estructura de la sala fue movida al archivo sala.c para poder manipular la forma en la que se guardan los objetos y las interacciones. Previamente, en el TP1, se hacía en vectores dinámicos pero ahora opté por hacerlo en TDAs.
- Utilicé tres TDA Hash para almacenar los objetos de la sala. Uno para todos los objetos, otro para los objetos conocidos y otro para los objetos recibidos. Con esta implementación se logró reducir la complejidad en las operaciones de inserción, búsqueda o eliminación de objetos, pues no existen dos o más objetos con la misma clave y las listas de la tabla de hash abierta no superan la cantidad de un elemento almacenado por cada una. Esto hace que cada operación sea O(1) en lugar de O(n), como podría haber sido el caso de haber utilizado un TDA Lista.
  Por otro lado, utilicé dos TDA Lista: uno para almacenar todos los objetos almacenados en los TDA Hash y otro para almacenar todas las interacciones. El TDA Lista de todos los objetos simplificó la liberación de memoria en la función de sala_destruir.
  Si bien las funciones primitivas del TDA Lista de interacciones son O(n), es más eficiente que utilizar un TDA Hash pues, a diferencia de los objetos, las interacciones si pueden llegar a tener claves repetidas.
- Al correr las pruebas del TP1 en el TP2, el archivo no compila ya que se está accediendo directamente a la estructura sala_t que dejó de estar en estructuras.h y ahora pasó a estar en sala.c. Para evitar esto, utilicé una biblioteca auxiliar que contenga la estructura de sala dentro del archivo de pruebas, y así utilizar las funciones de hash_cantidad y lista_tamaño usando los campos de sala->objetos y sala->interacciones como parámetros para obtener la cantidad de objetos e interacciones respectivamente.
  Otra prueba que falla es la que se encarga de verificar si el vector de nombres está ordenado, lo cual sucede porque la estructura hash en la que ahora se encuentran todos los objetos no los contiene de forma ordenada.
- Se agregó al enumerado el tipo de acción ESCAPAR que corresponde al carácter 'g' en los archivos de acción para que ahora el usuario pueda utilizar un objeto conocido para "escapar" de la sala y poder ganar el juego.

3. Detalles de implementación

Para "adaptar" el TP1 al TP2, primero modifiqué las funciones y procedimientos que se encargan de manipular los datos. Como se mencionó anteriormente, en el TP1 se utilizaban funciones que recibían vectores dinámicos, mientras que en el TP2 se utilizan estructuras de tipo TDA.

En lugar de utilizar *realloc* para reservar memoria a medida que se agregan los objetos o interacciones, las funciones correspondientes de Hash y Lista se encargan de hacerlo cada vez que se inserta un nuevo elemento.
También las funciones que se encargan de devolver un vector con todos los nombres de los objetos y verificar si las interacciones son válidas fueron modificadas para recorrer el Hash y la Lista correspondiente en lugar de recorrer el vector de forma iterativa.
Luego de hacer todas las modificaciones correspondientes para adaptar el código a las nuevas estructuras implementadas, comencé a desarrollar las funciones y procedimientos que fueron agregados en el nuevo sala.h utilizado en el TP2.

3.1. Detalles de sala_agarrar_objeto

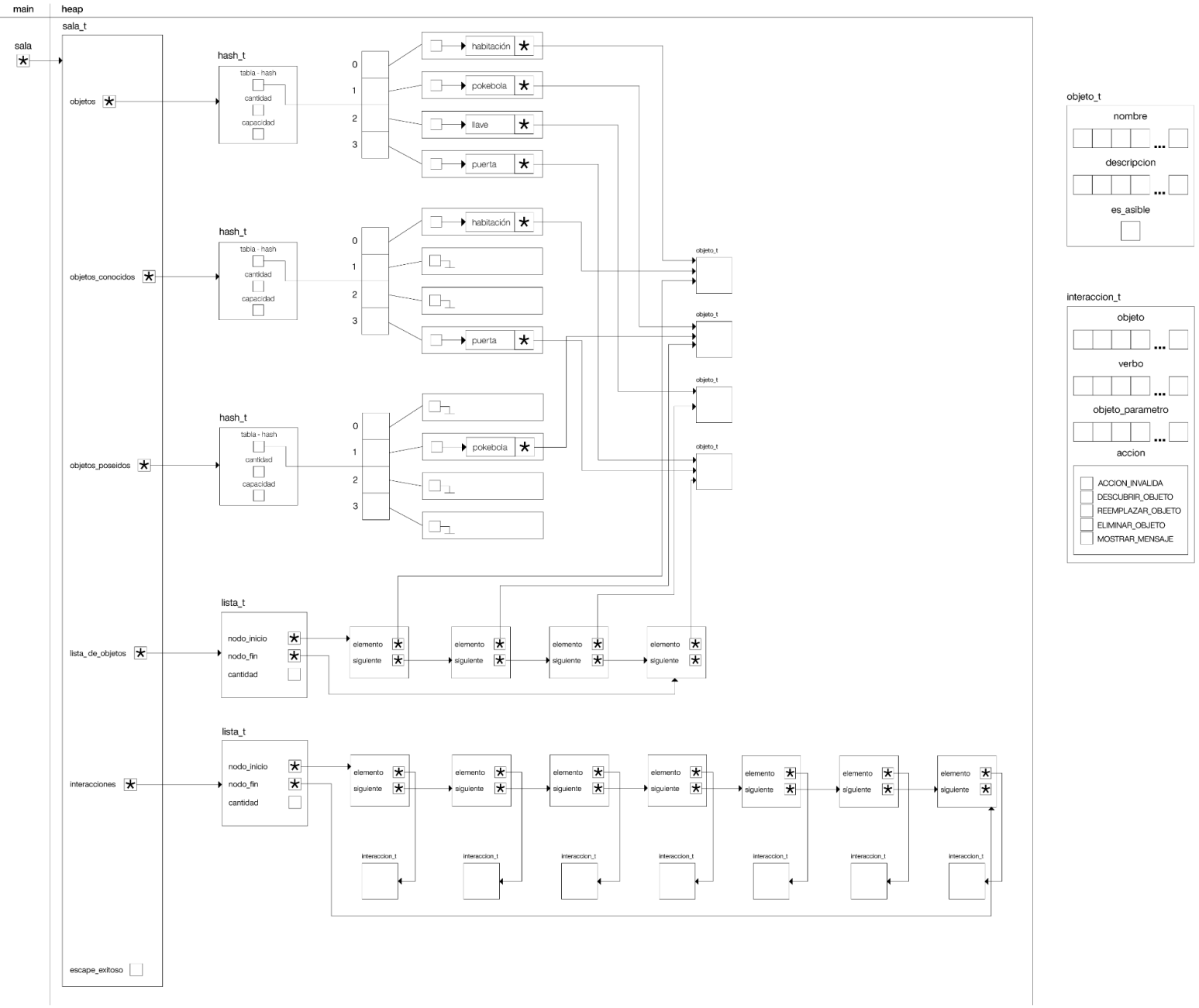
La función tiene como objetivo almacenar en el Hash de objetos poseídos el elemento con el nombre recibido por parámetro siempre y cuando el mismo ya no se encuentre en posesión del usuario pero si esté visible en la sala, es decir que sí se encuentre en el Hash de objetos conocidos.
Luego se obtiene un puntero al objeto con la función hash_obtener y se verifica si el mismo es asible o no. En el caso de que lo sea, se quita del Hash de conocidos y se agrega al Hash de poseídos.

3.2. Detalles de sala_describir_objeto

Esta función se encarga de devolver un string que contiene la descripción del objeto con el nombre recibido por parámetro siempre y cuando el objeto esté visible para el usuario, es decir que el mismo se encuentre en el Hash de objetos poseídos o en el de conocidos.

3.3. Detalles de sala_ejecutar_interaccion

La función se encarga de modificar las estructuras de objetos, objetos_poseidos y objetos_conocidos de acuerdo a los parámetros verbo, objeto1 y objeto2 recibidos.
Primero se utiliza el iterador de lista para buscar la interacción que coincida con el verbo y los objetos recibidos y luego se llama a la función ejecutar_interacción que verifica el tipo de acción de la interacción encontrada.
Si el tipo de acción coincide con
- ELIMINAR_OBJETO: Se elimina del hash que contiene todos los objetos y se verifica si está en objetos conocidos u objetos poseidos, eliminando el objeto del correspondiente.
- DESCUBRIR_OBJETO: Se verifica que el objeto y el objeto parámetro estén visibles, y si es así, que el objeto dentro de la acción no lo esté para así agregarlo al listado de objetos conocidos.
- ESCAPAR: Modifica el estado del juego asignándole el valor *true* a sala->escape_exitoso para indicar que el jugador pudo ganar el juego.
- REEMPLAZAR_OBJETO: Se verifica que el objeto parámetro sea visible para el usuario y si es así, se quita de todos las estructuras de objetos y se inserta el objeto que pertenece a la acción dentro del hash de objetos conocidos.
- ACCION INVALIDA: No se realiza ninguna interacción.
Cuando se realiza una interacción exitosamente o la acción recibida coincide con MOSTRAR_MENSAJE, y se recibe una función de mostrar_mensaje, se imprime por pantalla la descripción correspondiente al campo acción del objeto.



Para mejorar la legibilidad del gráfico, se simplificaron las estructuras de objeto_t e interaccion_t en cuadrados vacíos dándose a entender que cada uno de los mismos está compuesto correspondientemente por los campos especificados por los dispuestos en la parte derecha del diagrama.

Para compilar el TP2, se utilizó el archivo makefile adjuntado en la entrega.
Los comandos en cuestión son:
- *make escape_pokemon*, para compilar el trabajo.
- *make valgrind*, para ejecutar el programa con Valgrind.
- *make pruebas*, para compilar el archivo de pruebas.
- *make valgrind-pruebas*, para ejecutar las pruebas con Valgrind.