Trabajo Práctico 2 - Escape Pokemon

[7541/9515] Algoritmos y Programación II
Primer cuatrimestre de 2022

Alumno:	Carranza, Lihuén
Número de padrón:	108981
Email:	lcarranza@fi.uba.ar

Índice

1. Compilación

El trabajo contiene un makefile, por lo tanto, para ejecutar el juego se utiliza "*make*". Para ejecutar las pruebas junto a valgrind se usa "*make valgrind-pruebas*".

2. Modificaciones

2.1. Estructuras

En el primer trabajo práctico implementamos las interacciones y objetos con vectores dinámicos. Luego del recorrido a lo largo del cuatrimestre, aprendimos nuevos métodos para agrupar de manera eficiente un dato. Es por eso que para guardar los objetos ahora se utilizarán hash ya que los objetos son únicos, por lo tanto tienen una clave. En este trabajo implementé tres hashes: los objetos que se obtienen desde un archivo, los objetos que están en el escenario y los objetos poseídos por el jugador. Por otra parte, para las interacciones válidas utilicé una lista ya que hay diferentes interacciones para un mismo objeto. De implementarse un hash, se reemplazarían las interacciones ya que un objeto no podría tener más de una interacción.

En resumen, la estructura de sala contiene los objetos (hash) y las interacciones (lista) leídas desde los archivos. Además tiene una nueva estructura llamada jugador. Ésta contiene un booleano que determina si pudo escapar. los objetos (hash) en el escenario de la partida y los que posee el jugador.

2.2. Pruebas anteriores

Luego de redefinir las estructuras del trabajo anterior, arreglé las pruebas que ahora fallan debido a los cambios implementados. En primer lugar, nos encontramos con:

✗ Todos los nombres de objeto son los esperados

Esto sucede debido a que en las pruebas al verificar los nombres de objetos recibidos a través del archivo .txt correspondiente están en el orden en el cual el archivo los leyó. Habiendo cambiado la estructura de vector a hash, cambia el orden en el que se guardan en el hash, ya que en éste se guardan de acuerdo a la clave. Para corroborar esto, dentro de la función sala_obtener_nombre_objetos creé una estructura que guarda los objetos, les crea una clave y los inserta en el hash. Entonces, el orden de los objetos es diferente, por lo tanto, cambié el orden del vector de objetos esperados en las pruebas.

En segundo lugar, tenemos un bloque de pruebas fallidas sobre interacciones:

✗ Puedo examinar la habitación
✗ Puedo usar la llave en el cajón
✗ Puedo abrir la pokebola

✗ Puedo examinar el cajón abierto

La causa es la misma que la de la prueba anterior: cambié las estructuras y no se puede acceder a las interacciones a través de un vector sino de una lista. Anteriormente, se iteraba el vector de interacciones comparando el verbo, el objeto1 y el objeto2. Si los tres coincidían, la interacción se podía ejecutar. Ahora las interacciones se guardan en una lista, por lo tanto, para la verificación de la interacción creo una interacción auxiliar que será el contexto de la función que se le pasa a lista_buscar_elemento. Si la búsqueda de la lista devuelve la interacción auxiliar que creé, entonces la interacción es válida.

3. Desarrollo

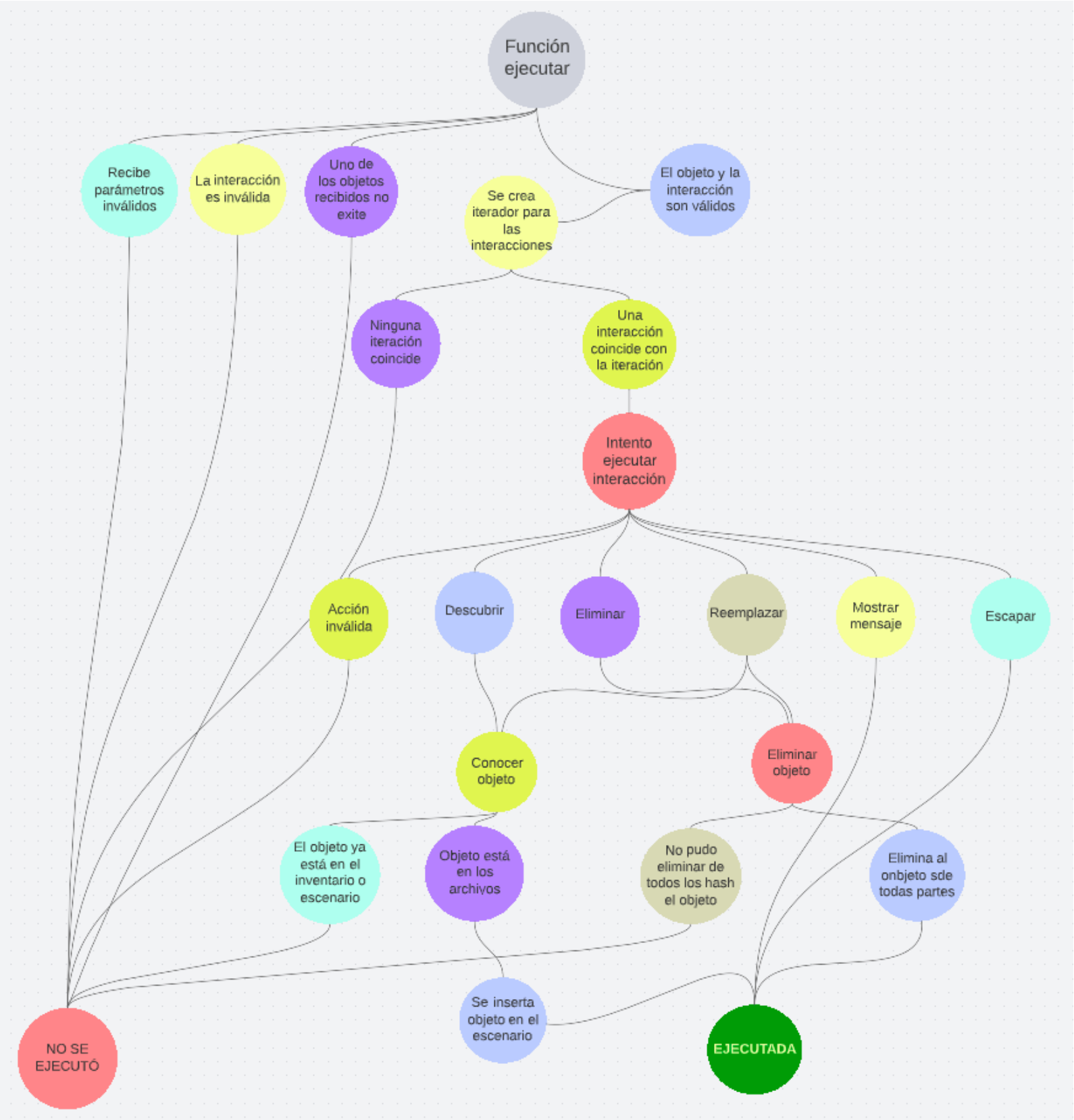
3.1. Nuevas Funciones

Dentro de las funciones que se incorporaron a esta nueva versión, hubo tres que pudieron ser utilizadas gracias a una función interna que creé en la primera versión. En el trabajo anterior, debíamos crear los nombres de los objetos a partir de un vector de objetos en la estructura de la sala. Ahora, se incorporaron funciones para obtener los objetos que están en la escena del jugador y en su posesión. Debido a que para los objetos utilizo la estructura de hash, la función se puede reutilizar para los tres casos.

La función más importante en esta segunda parte es la de ejecutar la interacción. Para realizarla, hay que tener en cuenta en primer lugar si la interacción es válida, luego si es objeto está en la sala. En caso de no estarlo, se termina la función. Pasadas estas dos pruebas, se crea un iterador de la lista de interacciones y se recorre la lista hasta el final, debido a que si hay más de una acción que se puede realizar con un objeto, pueda ejecutarlas todas. Si la interacción iterada coincide con la que fue recibida por parámetro, se intenta ejecutar la acción.

Tuve en cuenta la corrección previa y agregué descripciones de las funciones privadas.

En la siguiente página hice un árbol para que se entienda todo lo que hace la función.


Función ejecutar
Recibe parámetros inválidos
La interacción es inválida
Uno de los objetos recibidos no exite
El objeto y la interacción son válidos
Se crea iterador para las interacciones
Ninguna iteración coincide
Una interacción coincide con la iteración
Intento ejecutar interacción
Acción inválida
Descubrir
Eliminar
Reemplazar
Mostrar mensaje
Escapar
Conocer objeto
Eliminar objeto
El objeto ya está en el inventario o escenario
Objeto está en los archivos
No pudo eliminar de todos los hash el objeto
Elimina al onbjeto sde todas partes
Se inserta objeto en el escenario
NO SE EJECUTÓ
EJECUTADA

3.2. Escape Pokemon

Para poder interactuar con el juego decidí comenzar por mostrarle al usuario una pantalla de inicio seguida de las instrucciones. Luego se le solicita al jugador que ingrese un comando. Puede pedir “ayuda” y “ayuda-extra” en caso de necesitar recordar cómo se juega o las instrucciones.
Los datos ingresados por el usuario son almacenados en un string que luego se divide en tres palabras. Se puede ingresar mayúsculas y minúsculas.

Luego de cada interacción con el usuario, se actualiza la pantalla con los objetos visibles y los que están en su inventario.

Si no ingresa objeto y el la interacción utilizada es “salir”, se termina el juego.

Cuando se ingresa un input se verifica que sea visible o esté en el inventario. Si el verbo utilizado es agarrar, se intenta agarrar el objeto; de poder hacerlo se mostrará un mensaje por pantalla. Si la palabra es “describir”, se mostrará por pantalla la descripción del objeto. Si es otra palabra, se verifica que la acción sea válida con ese objeto, de serlo, se intenta ejecutar la acción.