FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TRABAJO PRACTICO 2: Simulador

Nombre	Lautaro De Nobili
Padron	107394
Email	ldenobili@fi.uba.ar

Funcionalidad del simulador:

Crear un .c que permita ejecutar las tres funciones principales que nos fueron pedidas, junto a los 7 eventos que se encuentran explicados en el eventos.txt.

Principalmente, ha de ser posible, como usuario, utilizar el simulador para poder reclutar, entre las nuevas enfermeras del hospital pokemon, lo mejor de lo mejor. Esto se define en base a cuantos puntos lograsen una vez terminada la simulacion, puntos los cuales se obtienen adivinando los niveles de los pokemon.

Funcionalidad teorica (para la materia):

Aplicar todo lo que vimos en la materia, hasta antes de ver grafos. Esto incluye:

- Utilizar las estructuras de datos implementadas, al igual que seleccionar la mejor para cada caso;
- Evitar el uso de vectores dinamicos para almacenar elementos (para eso tenemos las estructuras de datos).

Reimplementacion del TP1, hospital:

Ya que el TP2 incluye al TP1 y no podemos usar vectores dinamicos, hemos de redefinir nuestro hospital.c.

Para ello yo implemente las estructuras de la siguiente manera:

- Hospital:
  - Lista de entrenadores_t: los entrenadores son insertados en la lista a medida que se leen en el archivo. Esto es para despues, en el Evento.AtenderProximoEntrenador y en el simulador_t, poder tener una copia de esta lista, implementada como cola, de forma de ir sacando siempre el primer entrenador (pasa a ser atendido) y tener ordenado en un heap de pokemones_a_atender el orden de entrada;
  - Contador de cantidad de pokemones: simplemente para llevar la cuenta de cuantos pokemones poseo. No uso un contador de entrenadores porque eso lo obtendria con lista_tamanio(lista de entrenadores).

Quiero destacar que todas estas estructuras las defini en un .h separado al hospital.c/.h de forma tal que pudiese acceder a ellas desde el simulador.c

continuacion...

- Pokemon:
  - Nombre: me guardo el nombre. No hay razon para no guardarmelo y se usa bastante;
  - Nivel: tambien me guardo el nivel, se usa mucho muy en el simulador;
  - Entrenador: me guardo una referencia a su entrenador_t, de forma de acceder mas rapido a el;
- Entrenador:
  - Nombre: me lo guardo, pues se usa en Evento.ObtenerInformacionPokemonEnTratamiento;
  - Id: no se usa pero por si las moscas, me lo guardo;
  - Arbol de pokemones: use un arbol meramente porque me gustan mas que las listas, por como la implemente. Pero en si, no hay ninguna funcionalidad superior a listas.

    Podria haber usado una lista, por ser su insercion en la ultima posicion O(1).

Algo a destacar del TP1:

Para la funcion hospital_con_cada_pokemon, use un ABB auxiliar, en el cual inserto todos los pokemones de todos los entrenadores, siendo mi comparador el nombre de esos pokemones.

Esto me permite tener un orden definido y requerido para el iterador de hospital (orden alfabetico ascendente).

Es por esta razon que no era muy relevante que usase para almacenar los pokemones en el entrenador (si lista o ABB), pues de todas maneras iba a usar un ABB auxiliar aca.

Muchos .h:

Puede ser que haya sido abusiva la cantidad de bibliotecas extra que cree, pero mantienen el orden y me hacen sentir mas pro.

Todas las bibliotecas adicionales que cree fueron:

- Estructura del hospital: para poder acceder desde el simulador.c;
- Biblioteca del heap: un “nuevo TDA”, sacado de RPL;
- Biblioteca de dificultades: para un mejor manejo de las dificultades;
- Colores: para tener mas comodidad al usar los colores en los printf del main.c.

Algo sobre el TDA Heap:

Si bien no habia que hacer pruebas sobre el heap, pues es sacado de RPL, y no hice, mi manera de comprobar el correcto funcionamiento de este fue:

- Insertar en preorden desde el ABB de pokemones al heap;
- Hacer printf's para ver si se ordenaban correctamente a medida que insertaba;
- Al extraer la raiz, lo mismo, hacia printf's para ver si se reordenaba correctamente la "nueva raiz".

Dificultades.h:

En esta biblioteca implemente:

- Obtener dificultades estandar del simulador: facil, normal y dificil;
- Duplicar dificultad: para tener una copia de la dificultad que me pasa el usuario en Evento.AgregarDificultad. Opte implementar el copiar aca en lugar del simulador.c porque me parecia que encajaba bien con el concepto de ser una biblioteca de manejo de dificultades;
- Destructor de dificultades: uso memoria, tengo que liberar memoria.

Ahora si... Estructura del simulador_t:

- Hospital: obvio me guardo el hospital, pues yo desde el simulador tomo el control del mismo;
- Sala de espera, entrenadores: si bien es una lista, la manejo como cola, de manera que voy sacando siempre el primer entrenador que entro a ella para luego poner sus pokemones en un heap. Destaco, esta es una lista a parte de la del hospital, esto es para no modificar el hospital;
- Sala de espera, pokemones: es un heap, el cual se ordena por nivel de los pokemones. Es esta manera, siempre que extraiga la raiz, siendo esta un pokemon, va a ser el pokemon de menor nivel.
- Pokemon a atender: exprese mal el nombre, pues es el pokemon que ESTA siendo atendido. Obtengo este pokemon extrayendo la raiz de la sala de espera pokemon.

que estructura grande...

- Adicionalmente me guardo las estadisticas. Estas son inicializadas con los datos del pokemon y modificadas por cada evento. Es una manera mas rapida de poder ejecutar el Evento.ObtenerEstadisticas;
- Tengo una lista de dificultades, donde me guardo tanto las dificultades estandar (al momento de crear el simulador) y las agregadas por Evento. AgregarDificultad (en forma de copia de ella). El ID de la dificultad va a ser la posicion en la lista ☺;
- Dificultad actual: me guardo los datos de la dificultad que este seleccionada (la Normal al inicializar el simulador), para tener un acceso mas rapido a ella, sin tener que recorrer la lista hasta esa posicion de dificultad cada vez que tenga que usarla.

sigue esto?

Si, sigue, pero poco.

- Un boolano que me dicta si el simulador esta activo, el cual arranca en true (quien quiere crear un simulador que ya se encuentre desactivado?), y lo modifico en Evento.FinalizarSimulacion;
- Un contador de la cantidad de intentos actual, el cual sube con Evento.AdivinarNivelPokemon. Es mas facil llevar una cuenta para la funcion de la dificultad que nos da los puntos, en funcion de cuantos intentos le tomo adivinar.

Funciones del simulador:

Honestamente, no se que decir aca. No hay mucho para implementar de funciones, lo mas complicado es elegir la estructura del simulador, el resto es manejo de las estructuras de datos que seleccionamos.

- simulador_crear(): reserva memoria para el simulador e inicializa todos sus campos según consigna y las estadisticas iniciales;
- simulador_simular_evento(): simplemente alterno entre los posibles eventos y ejecuto una funcion auxilar en base a que evento me piden simular. Las funciones no son, la verdad, mucha cosa, solo me muevo por las estructuras para hacer funcionar todo;
- simulador_destruir(): libero toda la memoria usada, al igual que destruir el hospital.

Comentarios:

- La parte mas complicada de las funciones de eventos fue saber cuando devolver ExitoSimulacion o ErrorSimulacion, junto a saber cuando aumentar el contador de cantidad de simulaciones;
- Ahora si hay pruebas. :(

Otros comentarios...

Muchisimas gracias por la materia. Tanto el contentido como la forma en la que fue dada, es pura genialidad. Muy dificil, pero con mucha recompensa.

Muchas gracias Mauricio por el tiempo dedicado a corregir y resolver dudas.