TP 2 - Simulador

[7541/9515] Algoritmos y Programación II

Segundo Cuatrimestre - 2021

Alumno:	**González Calderón, Julián**
Padrón:	107938
Email:	gonzalezcalderonjulian@gmail.com

1. Introducción

La idea de este TP era utilizar el TP1 (hospital Pokemón) e implementar encima de el un simulador. Este simulador permitiría a las enfermeras entrenarse a la hora de descubrir el nivel de un Pokemón.

Antes de utilizar nuestro TP1, debíamos adaptarlo para que utilice los TDA vistos en la materia. Originalmente el hospital estaba implementado con vectores dinámicos.

Además, teníamos que implementar una interfaz por consola que permite a las enfermeras interactuar con el simulador.

2. Hospital

Para manejar los registros de la memoria, decidí usar una combinación de listas y un árbol binario de búsqueda.

Cada entrenador se añade a una lista de entrenadores. Cada entrenador a su vez, contiene una lista de Pokemones que le pertenecen. De esta forma, a la hora de atender a los entrenadores de un hospital, puedo acceder de forma eficiente a los Pokemones que posee.

Por otro lado, todos los Pokemones registrados en el hospital se añaden a un árbol binario de búsqueda de Pokemones. esto permite acceder rápidamente a todos los

Pokemones en orden, indiferentemente del entrenador que lo posee. El árbol ordena los Pokemones alfabéticamente, de esta forma a la hora de aplicar una función a cada Pokemón en orden alfabético, puedo simplemente recorrer el árbol de forma *INORDEN*.

3. Simulador

Para implementar el simulador, utilice como sala de espera un HEAP de Pokemones. Los Pokemones se debían atender en orden de nivel, por lo que al atender a un entrenador puedo agregar todos sus Pokemones a la sala es espera y atenderlos desde ahí.

Para determinar que entrenador sera el próximo a atender, no es necesario definir ninguna estructura adicional. Se podia usar simplemente la lista de entrenadores del hospital. Usando un contador de entrenadores atendidos para saber cual es el siguiente.

Estructura del Simulador

El simulador contiene los siguientes campos.

```
hospital_t *hospital;
EstadisticasSimulacion estadisticas;

heap_t *pokemones_en_espera;
pokemon_t *pokemon_atendido;

lista_t *dificultades;
unsigned dificultad_actual;
unsigned intentos;

int estado;
```

El campo estadísticas contiene las estadísticas de la simulación. En lugar de usar una estructura podría haber usado los campos individuales, pero me pareció mas practico hacerlo de esta forma.

La utilidad del heap *pokemones_en_espera* la mencione previamente. Por otro lado, *pokemon_atendido* se usa para almacenar el Pokemón que esta siendo atendido en ese momento. Es necesario ya que una vez accedemos a la raíz del heap esta se

quita, por lo que debemos almacenar el Pokemón atendido en ese momento en algún lugar.

Para almacenar las dificultades, utilice una lista. Cada dificultad tiene asignado un *id*, comenzando desde el 0. De esta forma, acceder a sus elementos mediante su índice *(id)* resulta practico.

intento se usa para contar la cantidad de intentos que esta tomando adivinar el Pokemón actual. Una vez adivinado el nivel del Pokemón, *intentos* se iguala a 0.

en *estado* se almacena el estado actual del simulador. Tiene dos valores posibles: *EN_EJECUCION* y *FINALIZO.*

Eventos del Simulador

Voy a explicar el funcionamiento de algunos eventos

- `AtenderProximoEntrenador`

  Accede al próximo entrenador de la lista, recorre su lista de Pokemones y agrega todos a la sala de espera.

  De ser posible, siempre debe haber un Pokemón siendo atendido. Por lo que si no hay uno, lo quita del heap y lo almacena en *pokemon_atendido.*

- `AdivinarNivelPokemon`

  Accede al Pokemón siendo atendido y compara su nivel con el nivel del intento pasado por parámetro. Devuelve la información necesaria (dependiendo de la dificultad actual)

  Si el nivel es correcto, entonces calcula los puntos en función de los intentos e iguala la cantidad de intentos a 0. Por ultimo, libera el consultorio y atiende al próximo Pokemón (extrae el Pokemón del heap)

- `AgregarDificultad`

  Verifica si existe una dificultad con el nombre de la nueva dificultad. Si no existe una dificultad con el mismo nombre, entonces duplica los datos pasados y la inserta al final de la lista. (esto se hace por si el usuario es descuidado y rompe el simulador)

Funciones del Simulador

Voy a explicar el funcionamiento de algunas funciones.

- `simulador_destruir()`

  Destruye las estructuras creadas por el mismo, y las dificultades. Al destruir las dificultades, invoca el destructor en cada elemento de la lista (se debe liberar la memora reservada para el nombre de la dificultad).

  No destruye ningún Pokemón ni entrenador, de eso se encarga el destructor del hospital.

- `agregar_pokemones_a_espera()`

  Recibe un Pokemón y un heap, e inserta el Pokemón en el heap. Al usarse en conjunto con *lista_con_cada_elemento*, inserta todos los elementos de una lista en el heap.

- `existe_dificultad()`

  itera la lista de dificultades con un *lista_iterador* y compara sus nombres con el nombres pasado por parámetro.

Interfaz del Usuario

Este programa hace uso de los eventos del simulador para permitir a un usuario inexperimentado interactuar con el simulador.

La única función que vale la pena mencionar es *mostar_informacion_dificultad*. Esta función simula el evento *ObtenerInformacionDificultad* con cada *id* comenzando del 0, hasta que de error. De esta forma, imprime la información de cada dificultad disponible, permitiendo al usuario elegir la deseada.

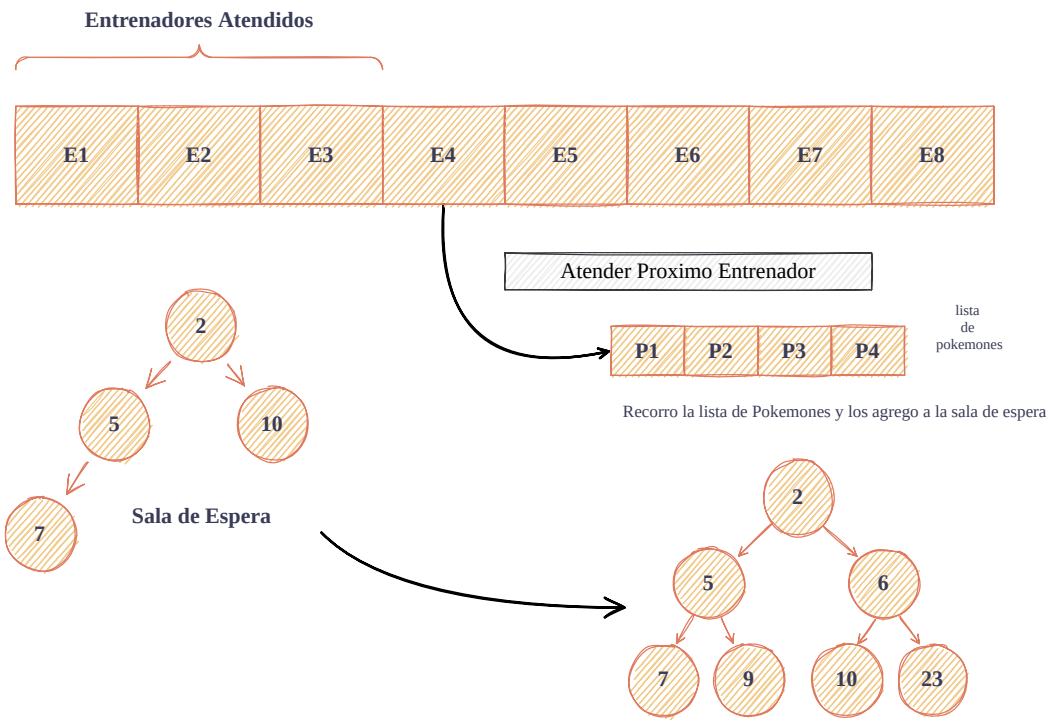
Las dificultades creadas en este archivo son bastante simples, ambas hacen lo mismo pero con distintos mensajes.

4. Diagramas

El funcionamiento interno de los TDAs no esta representado ya que ya se vio en los trabajos prácticos anteriores.

- `AtenderEntrenador`

  En este diagrama vemos como es la lógica de atender un entrenador. Accedemos a sus Pokemones y los agregamos a la sala de espera.


Entrenadores Atendidos
E1
E2
E3
E4
E5
E6
E7
E8
Atender Proximo Entrenador
P1
P2
P3
P4
lista
de
pokemones
Recorro la lista de Pokemones y los agrego a la sala de espera
2
5
10
7
Sala de Espera
2
5
6
7
9
10
23