FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico n2 — Hospital Pokémon

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	APELLIDO, Nombre
Número de padrón:	107924
Email:	matiavallejo@gmail.com
Email institucional:	mvallejos@fi.uba.ar

1. Introducción

El objetivo de este trabajo practico es crear un videojuego basado en un hospital pokemon que recibe entrenadores con pokemones y sus diferentes niveles mediante el cual el jugador debe intentar adivinar el nivel del pokemon, al jugador se le proveerá informaciones que lo ayudaran a llegar a ese nivel para poder terminar el juego y obtener el mayor puntaje posible. Uno de los objetivos también es poder aplicar los TDA vistos durante el año en la catedra y por lo tanto no se va a utilizar ni vectores dinámicos ni estáticos y así de esta manera obtener un TP más eficiente mediante usos de TDA. En este TP va a estar presente el desarrollo de Listas y Heap.

2. Teoría

1. Explicación de las estructuras

   Hospital.c : La estructura del hospital fue modificada respecto al primer TP ya que en esta contábamos con conocimientos de implementaciones avanzadas como son las listas o los árboles binarios, en este caso se utilizó una lista de entrenadores con sus respectivos nombres e id y cada entrenador a su vez contaba con una lista de sus propios pokemones, teniendo los pokemones un nombre un nivel y una referencia exacta hacia su entrenador. Se planteo de esta manera debido a la **facilidad para manejar la información** tanto de los pokemones y los entrenadores y su facilidad a la hora de recorrer la lista.

Simulador.c : Para la estructura del simulador utilizamos varias cosas, primero utilizamos el hospital para poder operar sobre el obteniéndose así datos de los pokemones almacenados y sus entrenadores, una "cola de espera" la cual si bien se podía plantear como una lista en formato de cola preferí plantearla como un heap minimal esto debido a la **facilidad que nos da el heap para obtener siempre el pokemon de menor nivel** ya que este siempre estará en la raíz, luego tenemos las estadísticas y la información del pokemon ya que en todo momento necesitaremos poder acceder a ellas para modificarlas, una lista de dificultades esto se debe a que cada dificultad posee tanto información como datos siendo la información la parte básica de la dificultad como el nombre si está en uso el id, y los datos la parte en donde se consulta como obtener un puntaje y chequear si está bien el nivel que se intentó adivinar, decidí poner tantos los datos como la información en una lista ya que **estos están relacionados entre sí** por lo tanto si tuviera que analizarlos por separado seria ineficiente de este modo es fácil recorrer la lista para buscar un id o incluso buscar la dificultad actual. Finalmente tenemos la cantidad de intentos que es el contador de la cantidad de intentos que el usuario lleva actualmente para adivinar el nivel del pokemon, dificultad_siguiente_id_disponible el cual nos sirve para cuando necesitamos agregar una nueva dificultad saber cuál es el id que le corresponde y simulacion_finalizada que nos indica si la simulación ya termino y por lo tanto no debe poderse ejecutar más funciones.

3. Detalles de implementación

Compilación y ejecución:

- El juego: el juego puede ser ejecutado y compilado mediante un comando sencillo en la terminal utilizando el makefile provisto en los archivos, este puede ser ejecutado de dos maneras, una simple mediante el comando `make juego` y otra mediante la cual obtenemos detalles sobre la memoria la cual es `make valgrind-juego` .
- Pruebas de lista: `make pruebas-lista` o con valgrind `make valgrind-lista`
- Pruebas de hospital: `make pruebas-hospital` o con valgrind `make valgrind-hospital`
- Pruebas de heap: `make pruebas-heap` o con valgrind `make valgrind-heap`
- Pruebas de simulador: `make pruebas-simulador` o con valgrind `make valgrind-simulador`

Lista

1. `lista_ordenar`

   Esta es la única función añadida a la lista para este TP, lo que hace es como dice su nombre ordenar los elementos de la lista mediante un comparador que recibe por parámetro, ese comparador va a devolver 0,1,-1 dependiendo de cómo tenemos que ordenar ese elemento en la lista.

Hospital

1. `hospital_leer_archivo`

   Esta función es una de las más importantes de este archivo, mediante el uso del split desarrollado va a leer el archivo enviado por parámetro y cargar en la estructura los valores correspondientes para los entrenadores y sus pokemones, se cambió respecto a cómo fue desarrollado en el primer TP ya que ahora contamos con una lista a diferencia de un vector dinámico.

2. `cargar_pokemones_en_lista`

   Esta función recibe un hospital y una lista y carga en esa lista todos los pokemones que hay en el hospital sin importar su entrenador o nivel, tampoco toma importancia el orden en que la ordena en la lista.

3. `comparador_pokemones`

   Es el comparador que se utiliza para enviar a la función lista_ordenar y poder ordenar así la lista, en este caso se ordenara alfabéticamente para poder recorrer los pokemones en `hospital_a_cada_pokemon` .

4. `hospital_guardar_archivo`

   Esta es una función nueva desarrollada para este TP, guarda el archivo llenando los datos del hospital en el mismo formato en que se lee.

Heap

1. `heap_insertar`

   Inserta un elemento en el heap, le aplica sift up el cual lo va acomodando en el heap mediante lo que devuelve el comparador con el cual fue creado el heap.

2. `heap_extraer_raiz`

   Extrae la raíz del heap, y le aplica sift down para que el heap quede acomododado dependiendo del comparador con el cual fue creado.

Simulador

1. `comparador_niveles`

   Es el comparador que se va a utilizar para crear el heap y así lograr un heap minimal teniendo en la raíz siempre al pokemon de menor nivel.

2. `crear_dificultades_estandar`

   Crea las dificultades iniciales con las que se puede jugar en el simulador y las añade a la lista de dificultades.

3. `atender_proximo_pokemon`

   Extrae el pokemon de la cola de espera(heap) y lo pone en tratamiento.

4. `buscar_nivel_pokemon`

   Recibe el nombre de un pokemon y el nombre de su entrenador y itera la lista de pokemones para buscar el nivel de ese pokemon y lo devuelve.

5. `atender_proximo_pokemon`

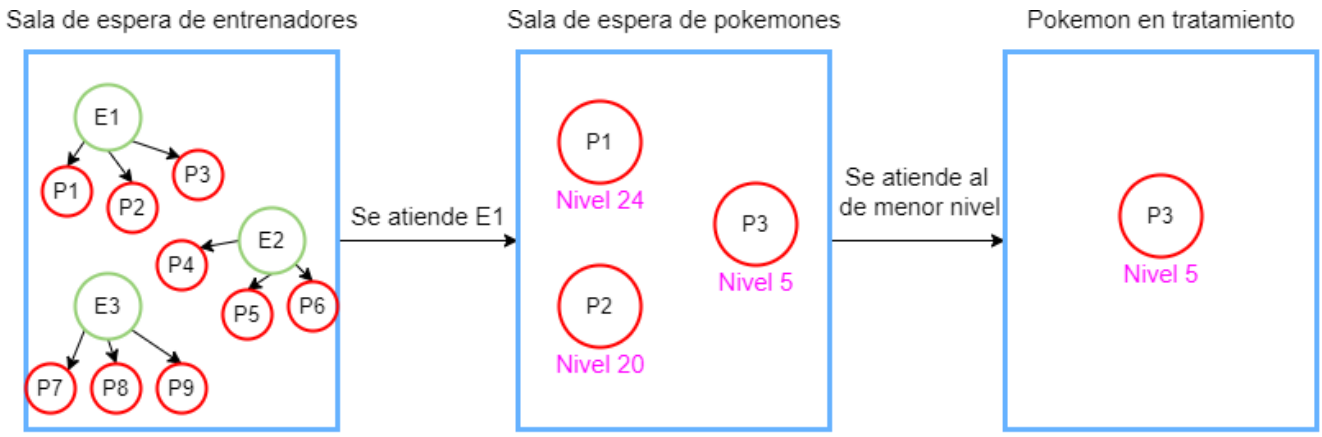
   Extrae el pokemon de la cola de espera(heap) y lo pone en tratamiento. Se utiliza cuando se atiende a un nuevo entrenador si no hay pokemon en tratamiento y cuando se adivina un pokemon para tratar al siguiente si es que hay uno.

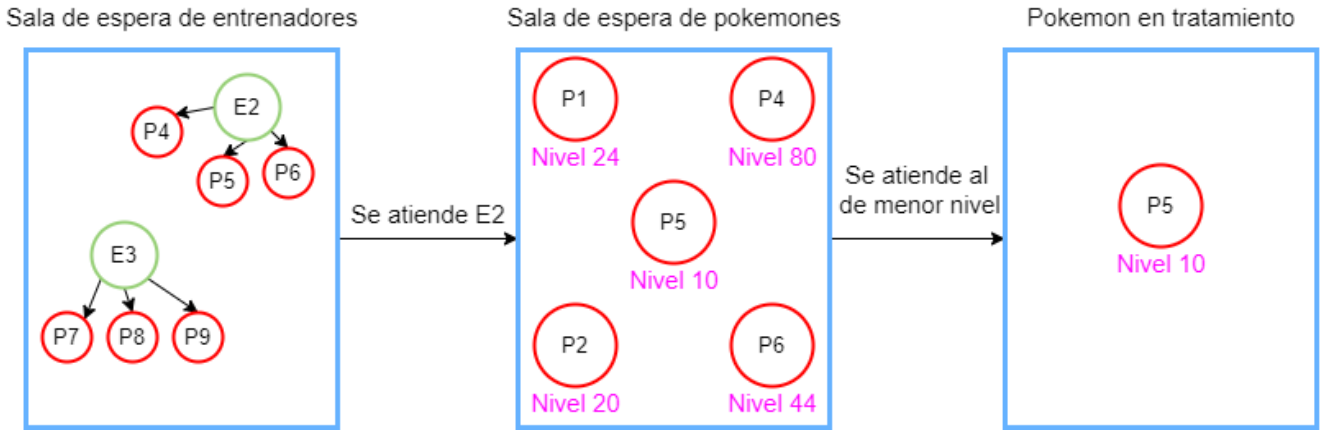
6. `agregar_pokemones_al_heap`

   Chequea cuantos entrenadores se atendieron para buscar el siguiente entrenador para atender y agrega todos sus pokemon a la cola de espera(heap) utilizando el comparador que se envió al heap, logrando así tener un heap minimal ordenado por el nivel del pokemon desde menor a mayor.

4. Diagramas

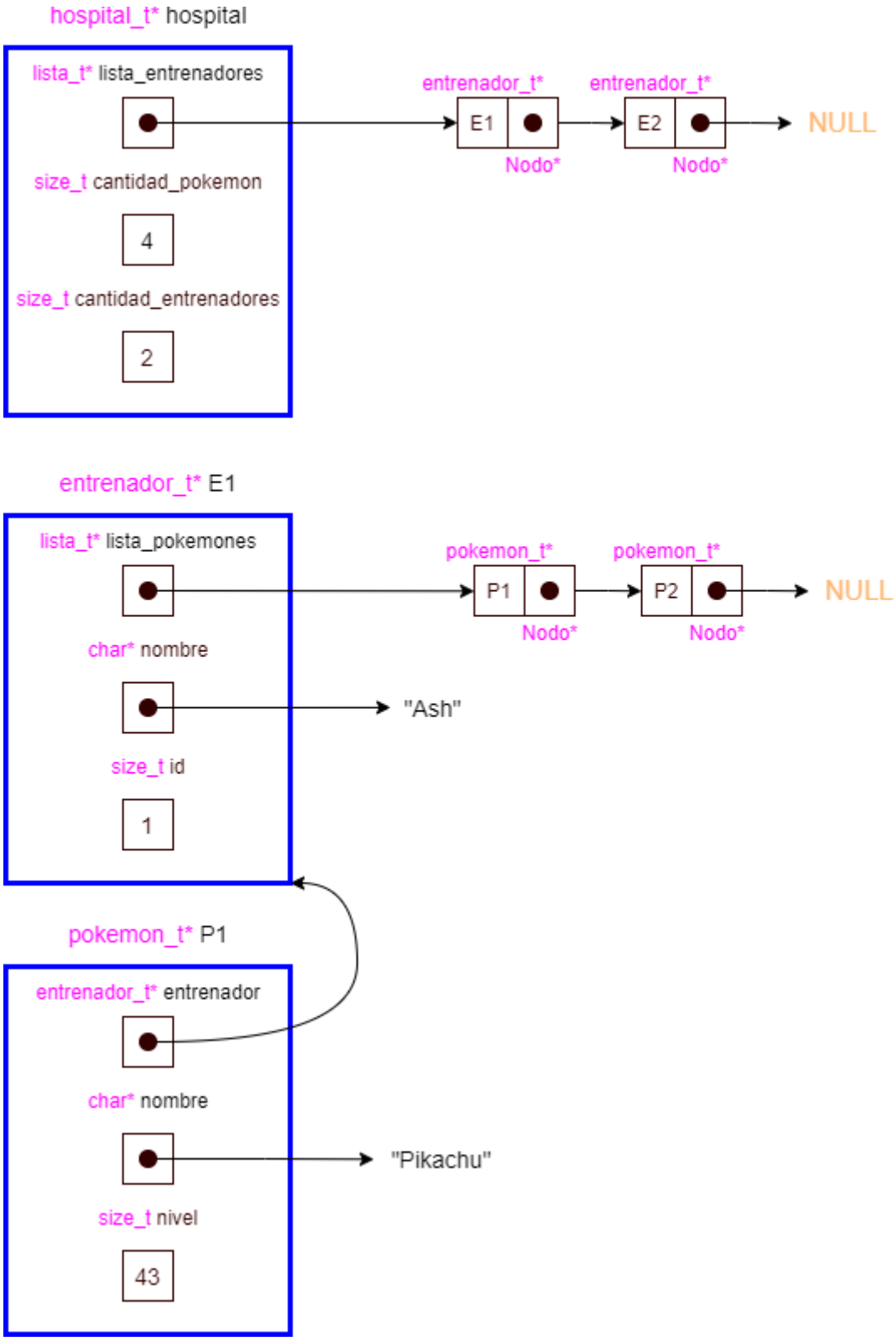
1. Explicación breve del hospital





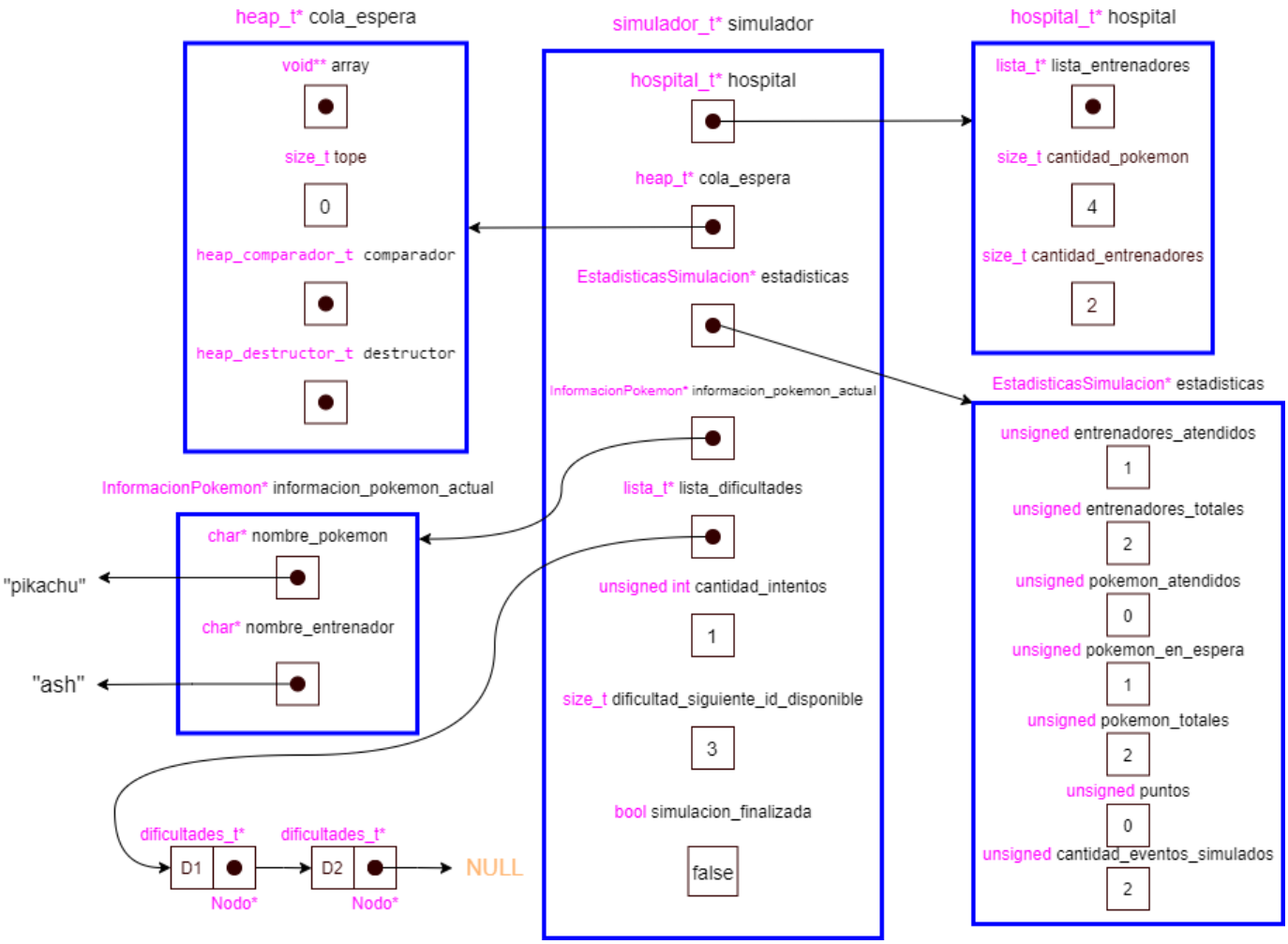
Este diagrama explica brevemente lo que sucede en el hospital, inicialmente tenemos una "sala de espera de entrenadores" donde están todos los entrenadores del hospital los cuales tienen sus respectivos pokemones, cuando se atiende al próximo entrenador se atiende los pokemones en forma de llegada del entrenador en este caso siendo E1 el primero que llego por lo tanto se atienden sus pokemones, entre los pokemones que están la sala de espera se elige el de menor nivel. En la parte de abajo como ya se atendió al E1 ya no está, ahora se atiende al E2 y en la sala de espera quedaron los pokemones sin atender del E1 sin embargo como ahora hay pokemones nuevos de menor nivel (Aunque sean de otro entrenador) se atienden primero.

2. Creación del hospital



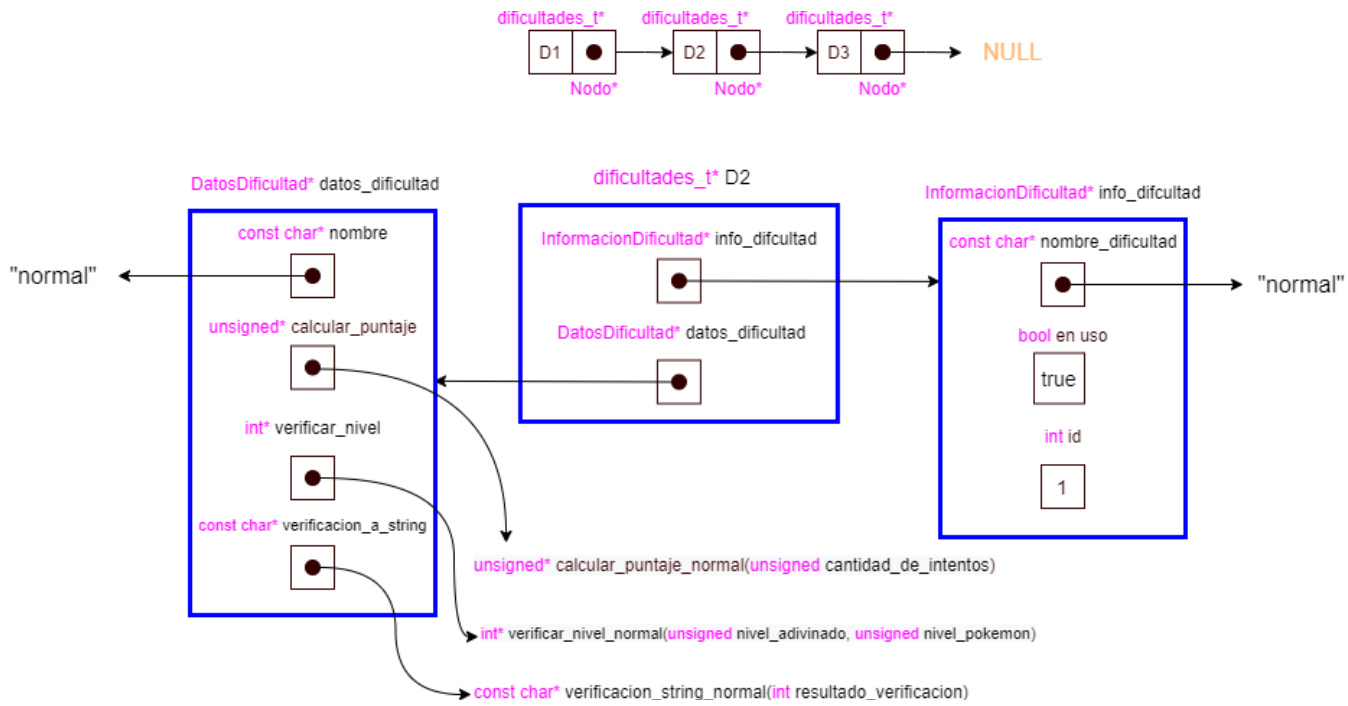
En este diagrama podemos ver lo que sucede en el archivo hospital.c y que luego se toma como el hospital en el simulador.c. En este caso podemos observar si tuviéramos 2 entrenadores con 2 pokemones cada entrenador, viendo como en el hospital_t se guarda una lista de estos dos entrenadores que a su vez estos entrenadores tienen una lista de pokemones los cuales tienen un nombre y un nivel y el entrenador hace referencia a su propio struct de entrenadores.

3. Creación del simulador



En este caso podemos ver cómo es de manera ilustrativa el simulador cuando se crea, en este caso se guarda el hospital explicado en el diagrama anterior, una cola de espera que es un heap minimal el cual se explicara en los siguientes diagramas, las estadísticas ya que debemos acceder a ellas en todo momento para modificarlas, estas tienen 2 entrenadores, 1 atendido y como cada entrenador tiene 3 pokemones 1 va a estar en espera hasta que sea atendido. Luego tenemos informacion_pokemon_actual la cual siempre tendrá el nombre del pokemon y entrenador que este siendo atendido, la cantidad_intentos las cuales empiezan como 1 y van aumentando en base a los intentos de adivinar el pokemon fallidos del usuario, la lista de dificultad la cual esta desarrollada en el siguiente diagrama, dificultad_siguiente_id_disponible sirve para, al momento de crear una nueva dificultad saber cuál es el id que le corresponde, se inicializa en 3 ya que por defecto el simulador ya cuenta con otras 3 dificultades(fácil(0),normal(1),difícil(2)) y finalmente simulación finalizada que es un booleano e indica si se pueden seguir realizando operaciones en el simulador o no.

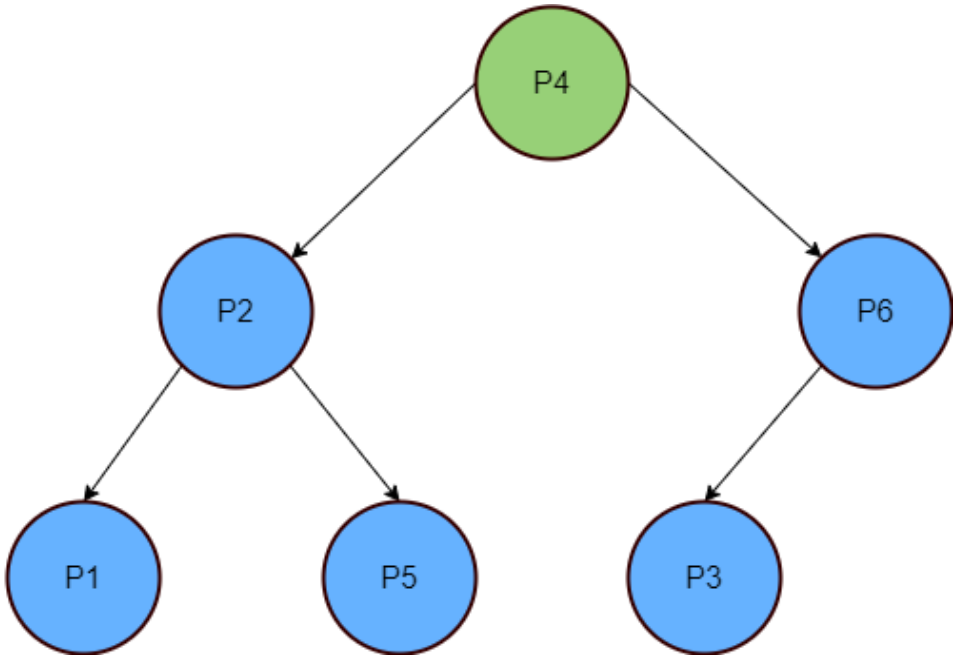
4. Lista de dificultades



En este caso observamos la lista de dificultades al momento de la inicialización del simulador, en donde la dificultad con el ID 1 es la dificultad normal la cual es la que por defecto está en uso esto lo podemos obtener de info_dificultad, por la parte de datos_dificultad contamos con punteros hacia las funciones correspondientes para probar si el intento del usuario fue correcto, que puntaje le corresponde por esa dificultad y el string que debe mostrar por pantalla dependiendo si se equivocó o no.

5. Cola de espera

pokemon_t* pokemon	size_t nivel
P1	51
P2	20
P3	90
P4	11
P5	36
P6	43



Finalmente tenemos la cola de espera del simulador que este caso se implementó como un heapminimal en donde le pasamos un comparador de niveles de pokemon logrando así que siempre este en la raíz el pokemon de menor nivel, esto nos facilita mucho el trabajo a la hora de tener que pasar un pokemon al tratamiento ya que solo tenemos que extraer la raíz del heap paraobtener el pokemon de menor niv