Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0117: Programación Paralela y Concurrente

Tarea Programada 1

Profesor : José Andrés Mena Arias

Estudiantes :

B82957 : Marco Ferraro Rodríguez
B86524 : Gabriel Revillat Zeledón

Índice

1. Enunciado

El propósito de este proyecto es implementar un simulador de batallas entre hilos basándose en la dinámica de batallas PvP del juego Pokémon GO. En estas batallas dos jugadores se enfrentan utilizando 3 Pokémon cada uno, donde cada Pokémon tiene ataques y características específicas (ataques rápidos, ataques cargados, puntos de vida, entre otros). Un enfrentamiento entre dos Pokémon consiste en ambos atacando simultáneamente al otro hasta que uno de los dos se debilite (sus puntos de vida llegan a 0). El objetivo de cada jugador es lograr debilitar a los 3 Pokémon de su oponente antes de que el oponente debilite los suyos. Puede ver un ejemplo de cómo funcionan las batallas en este video.

Para efectos de este proyecto no se va a implementar el juego como tal, sino una simulación entre dos jugadores virtuales. De igual manera no todos los detalles del sistema de batallas estarán dentro del alcance de la simulación, únicamente las que se especifican en los requerimientos de las siguientes secciones.

2. Diseño de la Solución

Para implementar una solución a este problema se plantea usar un Modelo Vista Controlador. En la sección de **Controllers** se plantea tener 4 archivos **.c**.

- **mapper.c**

  Este archivo tiene todos los datos e información necesaria para identificar cada Pokemon, sus ataques y la información de sus tipos. El profesor nos proporciono esta matriz de datos.

- **pokemon.c**

  Se encarga de inicializar los datos para cada estructura de datos. Además, tendrá métodos para recuperar la información de los ataques rápidos y ataques cargados.

- **player.c**

  Se encargara de mantener la información de un jugador. A lo largo del proyecto solo van a haber 2 jugadores, sin embargo cada jugador tendrá su propio equipo de 3 pokemones. El equipo se definirá en este archivo.

- **game_master.c**

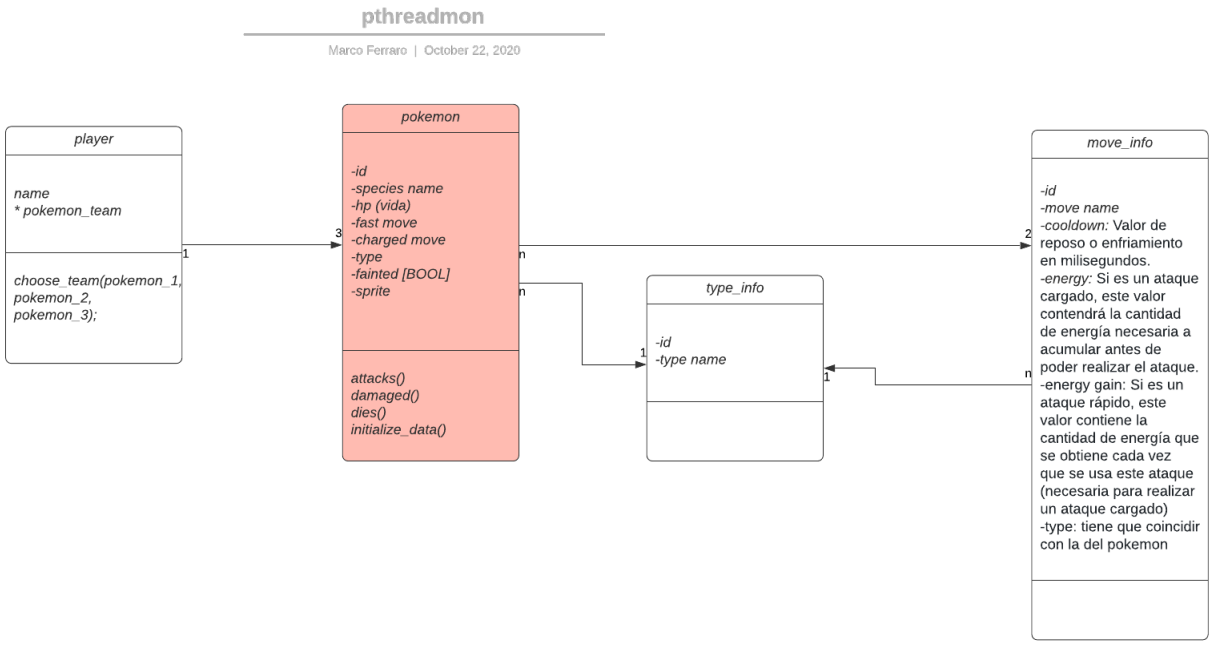
  Este archivo tendrá el labor de manejar la sincronización de hilos. Adicionalmente manejará la repartición de estructuras de datos para que la aplicación logre correr cumpliendo con los requerimientos. Será el archivo que controle el manejo de la pelea en sí.

Todas las firmas de los métodos y especificación de estas estructuras están bajo la carpeta de **Models** bajo sus **.h** respectivos.

Adicionalmente, bajo la carpeta de **Views** se encontrara **pthreadmon_ui.c** que se encargara de levantar la interfaz gráfica y el proyecto en si.

2.1. Diagrama UML

A continuación se presenta un diagrama para representar los requerimientos de datos para la implementación.



3. Implementación de Solución

3.1. Estructuras de Datos

3.1.1. pokemon_t

```
typedef struct
{
    int hp; // es 1500
    int efectivity; // por default es 1, pero depende
    int energy; // en la energia actual, cada vez que

    int is_attacking; // GABRIEL -> Para la interfaz
    int attacking_fast;
    int attacking_charged;

    pokemon_info_t * pokemon_info;
    move_info_t * fast_move_info;
    move_info_t * charged_move_info;
    type_info_t * type_info;
      You, 12 days ago • Ya crea pokemones
}pokemon_t;
```

Esta estructura tendrá variables que estará cambiando de valor como **hp, efectivity** y **energy**. Ademas algo más que cabe destacar que que cada estructura **pokemon_t** va a tener un grupo de punteros como se aprecia en la imagen. Este set de punteros contiene la información respectiva de cada atributo necesario del pokemon.

3.1.2. player_t

```
You, 4 days ago | 2 authors (gabrielrevillat
typedef struct player
{
    char* player_name;
    int pokemon_availible;
    pokemon_t** pokemon_team;

} player_t;
```

Lo mas importante de destacar de esta estructura de datos es que un jugador esta compuesto por su nombre, una variable que indica

3.1.3. game_master_t

```
typedef struct{
    pthread_barrier_t barrier;
    int battle_over;
    sem_t * pokemon_semaphore_array;

    pthread_mutex_t mutex;
    pthread_cond_t cond_var;
    pthread_cond_t sleep_cond_var;

    player_t ** players_array;
    int gotta_wait;
    player_t * winner;

    pokemon_t ** poke_p_array;
}battle_arena_t;      You, 5 days ago
```

Como se puede notar, **game_master_t** se utiliza como una estructura de datos compartidos para el trabajo de hilos. Más adelante se explica como se se utilizaras, pero para la sincronización de hilos se utiliza un mutex, 2 variables de condición, 1 barrera y un arreglo de semáforos. Además, cabe mencionar que la estructura de datos compartida esta utilizando un arreglo de jugadores llamado **players_array** que va a apuntar a los 2 jugadores existentes. Además, posee un puntero de punteros tipo **pokemon_t** para poder apuntar a estructuras de datos de los pokemones que están peleando actualmente. Las otras variables son para de uso para que se pueda correr la simulación satisfactoriamente. La GUI sacara toda la información necesaria para mostrar la información de esta estructura de datos.

3.1.4. private_data_t

```
You, 9 hours ago | 1 author (You)
typedef struct{
    int thread_id;
    size_t team_number;
    size_t team_id; // id dentro del equipo
    pokemon_t * pokemon;
    battle_arena_t * shared_data;
}private_data_t;
```

Finalmente, la última estructura de datos que se utiliza es **private_data_t**. Esta es la estructura para los datos privados para cada hilo. Como cada hilo es un pokemon, tiene un id de hilo, un id que permite identificar de que equipo es, y el id dentro de su equipo, (si es el primer, segundo o tercer pokemon)

3.2. Métodos de Sincronización

A continuación se mencionan el grupo de herramientas de sincronización que están en el programa y como se utilizan.

- **pthread_barrier_t barrier**

barrier se utiliza para sincronizar todos los hilos en el inicio de los métodos paralelos, para asegurar que se estén ejecutando al mismo tiempo.

- **sem_t * pokemon_semaphore_array**

  pokemon_semaphore_array se utiliza para dormir a todos los hilos que no están peleando actualmente. Cuando el pokemon de un equipo es derrotado, despierta por medio de este semáforo al siguiente pokemon de su equipo. Cada hilo esta mapeado a 1 entrada del arreglo.

- **pthread_mutex_t mutex**

  Este mutex se utiliza para las variables de condicion. No se vio con la necesidad de utilizar el mutex para otras variables compartidas, ya que por el diseño del programa, no hay condiciones de carrera. No va a haber 2 hilos que estén compitiendo o escribiendo sobre otra variable compartida.

- **pthread_cond_t cond_var**

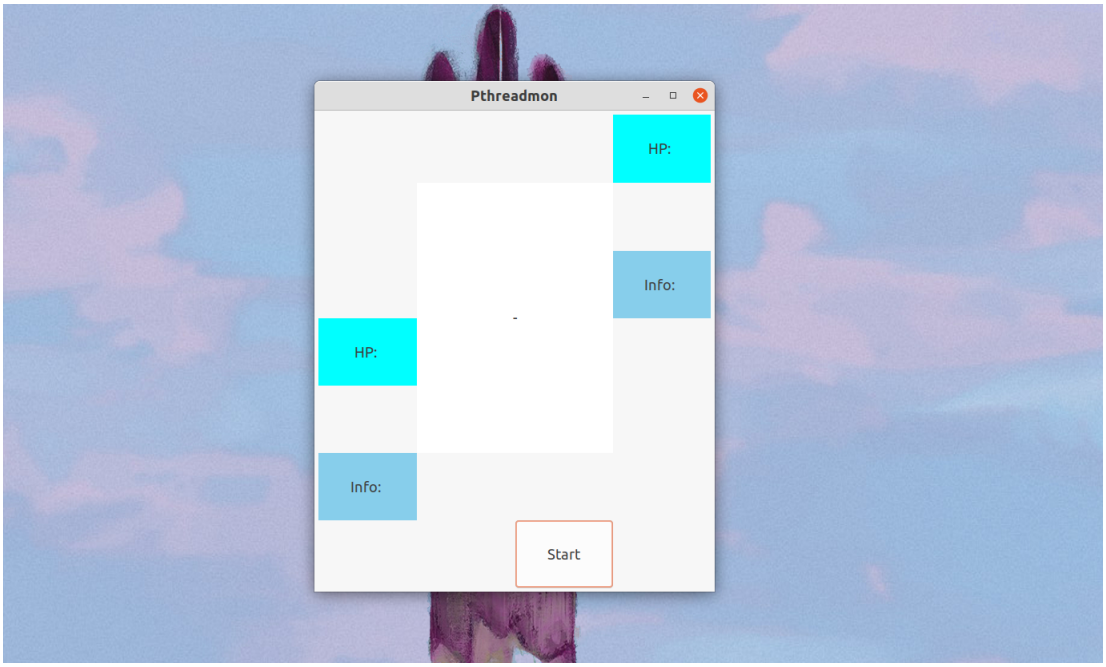
  Esta variable de condición es utilizada para que un hilo se espere mientras el otro pokemon esta atacando con un ataque cargado. Esto esta en los requerimientos del enunciado.

- **pthread_cond_t sleep_cond_var**

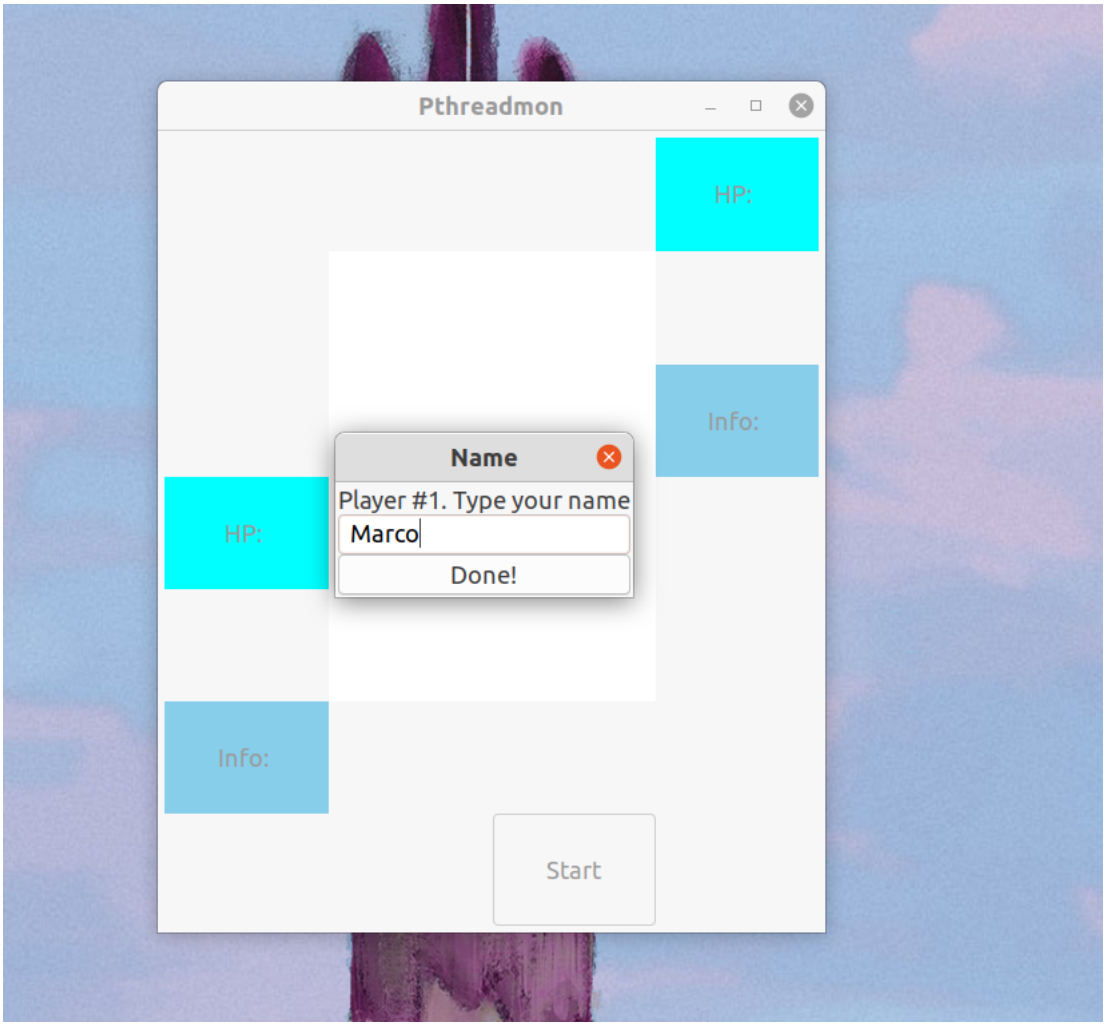
  Cuando un pokemon es derrotado, debe ser intercambiado. Esta variable de condición tiene la funcionalidad de que el otro pokemon o hilo que esta en el campo de batalla espere mientras el pokemon es intercambiado, para que así no ocurran errores de segmentation fault.

Cabe destacar, a como se menciono anteriormente, se trató de implementar esta solución bajo un problema de repartición de tareas, por lo tanto cada hilo tiene un grupo de variables a cuales acceder y a cuales no.

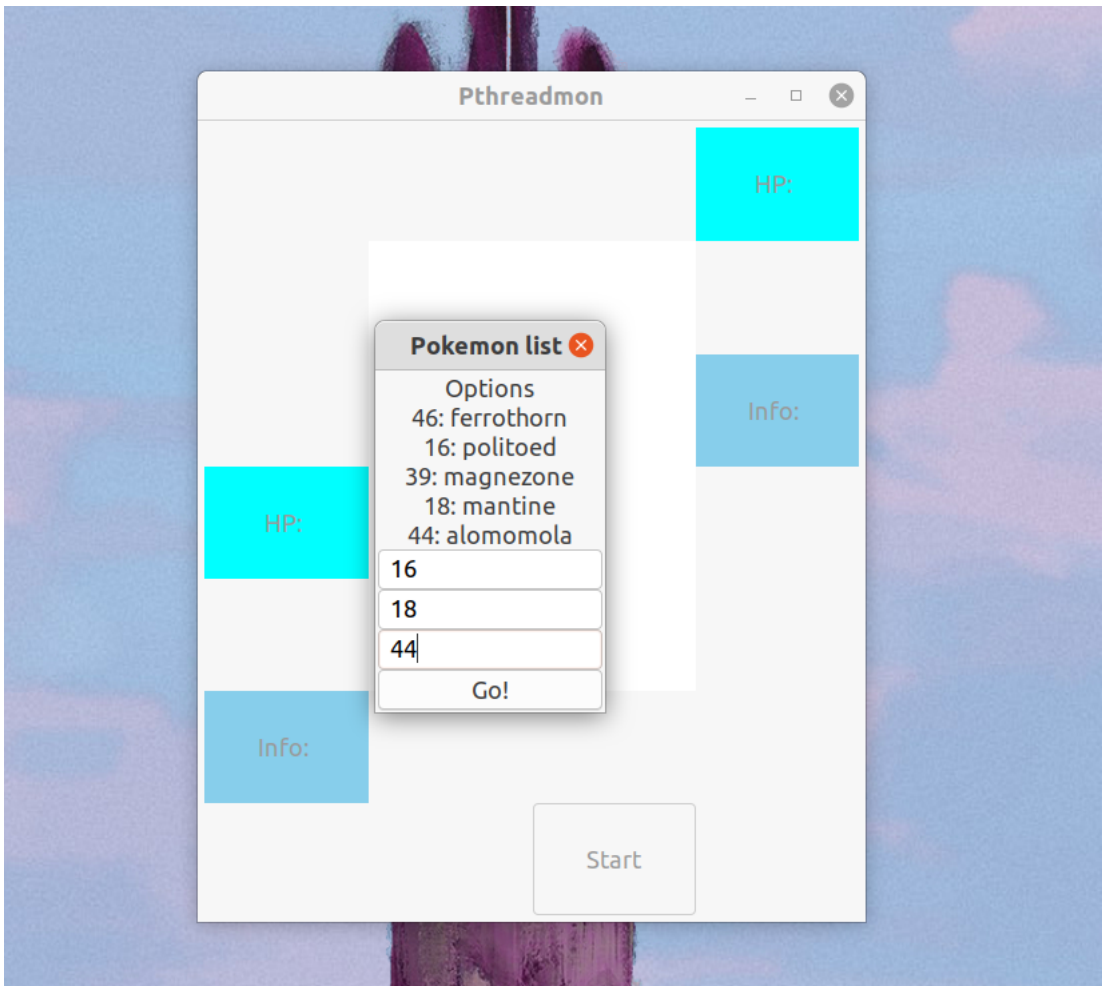
4. Manual de usuario



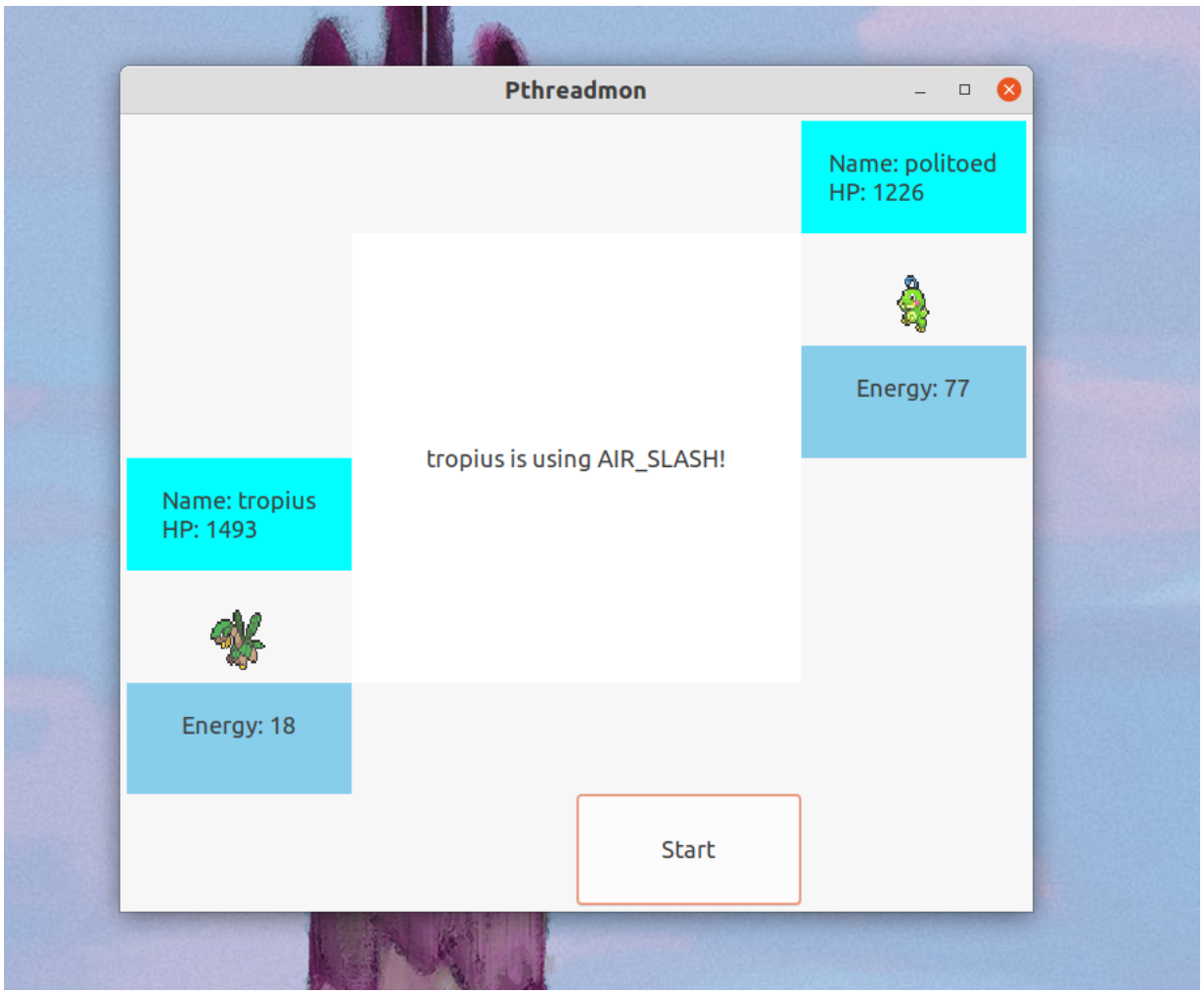
Al compilar y ejecutar el programa se le presentara al usuario la ventana principal de la simulación, para iniciar que **hacer click en el botón de Start**.



Subsequentente se le presentara una ventana para que el usuario escriba el nombre del Jugador # 1. Cuando se escriba se tiene que **hacer click en el botón de Done!**.



Después se le presentara al usuario 5 pokemones aleatorios. El usuario en cada entrada debe de digitar el id de 3 de estos pokemones. Este mismo proceso se repetirá para el Jugador # 2.



Automaticamente después de ingresar todos los datos empezará la simulación de la pelea, donde se muestra la vida y la energia de los pokemones peleando actualmente. El texto en el centro de la pantalla representa los ataques del Player 1.

Al final de la simulación el texto indicara el pokemon ganador de la pelea.

4.1. Compilación

En el proyecto se adjunta un archivo **MakeFile**. Para la compilación basta escribir desde la terminal **make** siempre cuando este corriendo un S.O. Linux.

5. Conclusiones

A medida de trabajo de grupo, se trabajo mediando github con ramas distintas, bajo el mismo repositorio.

Con respecto al tiempo y el desarrollo de la implementación, si se atraso un poco con la GUI, además de que se tuvo que hacer un refactoring a la clase **game_master.c** para evitar tener código duplicado, sin embargo si ayudo tener un diseño previo y un buen conocimiento de los métodos de sincronización.

5.1. Lo que no se logro

Primero, se estableció desde el inicio la intención de manejar la GUI como un hilo aparte, sin embargo no se hizo así por miedo y por tiempo.

No se logro hacer una validación de entrada, por lo tanto un usuario puede ingresar pokemones repetidos a su equipo.

Y de ultimo falto un poco lograr una interfaz más amigable, específicamente en el asunto de que si no se siguen las instrucciones del manual de usuario, y se trata de apretar enter después de ingresar los datos no va a pasar nada.

5.2. Distribución de Trabajo

Gabriel : player.c, pthreadmon_ui.c
Marco : pokemon.c, game_master.c