# Identificación del problema y análisis de requerimientos

Juan Felipe Correa Muñoz

## Caso de Estudio: Pokecollector (registro de cartas)

|  |  |
| --- | --- |
| Cliente | Empresa desarrolladora de videojuegos (Pokemon company). |
| Usuario | Jugadores de Pokémon que desean coleccionar cartas de manera virtual |
| Contexto del problema | *Se desea realizar un sistema en el cual se permita almacenar y gestionar diversas cartas Pokémon, garantizando que cada carta ingresada sea única, ordenando las cartas de manera descendente según los puntos de vida que tenga la misma, de igual manera se debe permitir que el usuario modifique algunas características de la carta o que si desea la elimine de el almacenamiento.* |
| Requerimientos funcionales | RF1: ALMACENAR INFORMACIÓN DE UNA CARTA.  RF2: ORDENAR CARTAS POR PUNTOS DE VIDA. |
| Requerimientos no funcionales | * Rendimiento (debe poder almacenar el numero de cartas solicitado sin que el programa se trabe). * Escalabilidad (Debe permitir que la colección de cartas crezca, sin comprometer ninguna de las otras funcionalidades). * Disponibilidad (Debe estar disponible el 99.9% del tiempo para que el usuario pueda realizar cualquier acción). * Seguridad (Se debe garantizar que no se pierda la información almacenada por el usuario). * Usabilidad (La interfaz del programa debe ser intuitiva y agradable para el usuario). * Mantenibilidad (El programa debe permitir actualizaciones futuras o agregar nuevas funcionalidades sin comprometer su funcionamiento). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador y nombre | RF1: ALMACENAR INFORMACIÓN DE UNA CARTA. | | | |
| Resumen | *El usuario debe poder ingresar la información de una carta, cada carta contiene un id, un nombre, una cantidad de puntos de vida, un tipo y una rareza determinada. Todo lo anterior se debe registrar correctamente al interior del programa. Además, cada carta debe ser única en el programa.* | | | |
| Entradas | **Nombre entrada** | **Tipo de dato** | | **Condición valores válidos** |
| Id | String | | *Debe ser distinto a los que ya se han ingresado (no se puede repetir),* |
| Nombre | String | | No debe contener espacios |
| Puntos de vida | Int | | Números positivos |
| Tipo | String | | Debe ser (Agua, Fuego, Planta, Oscura, metálica…) que estén en el juego. |
| Rareza | String | | (rara, común, legendaria…) |
| Resultado o Postcondición | La carta se almacena en el sistema, asegurando que no haya duplicados en los identificadores de carta, si se intenta registrar una carta con el mismo id, el programa debe volver a pedirlo. | | | |
| Salidas | **Nombre salida** | | **Tipo de dato** | **Formato** |
| Mensaje de confirmación | | String | *-La carta ha sido registrada correctamente (si hay espacio para almacenar la carta)*  *-Error, la memoria está llena (si no se puede almacenar más cartas)*  *- Error, la id que registrar ya ha sido registrada* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador y nombre | RF2: ORDENAR CARTAS POR PUNTOS DE VIDA. | | | |
| Resumen | *Al almacenar las cartas estas se deben organizar de mayor a menor según los puntos de vida, de manera que cuando el usuario quiera ver las cartas que tiene registradas estas se muestren desde la que tiene más puntos de vida hasta la que menos tiene (de la más fuerte a la más débil).* | | | |
| Entradas | **Nombre entrada** | **Tipo de dato** | | **Condición valores válidos** |
| Resultado o Postcondición | El sistema usa los puntos de vida ingresados por el usuario en el requerimiento 1 y ordena el arreglo que contiene las cartas de mayor a menor según esta característica. | | | |
| Salidas | **Nombre salida** | | **Tipo de dato** | **Formato** |
| Lista de cartas ordenadas | | String | (id, nombre, puntos de vida, tipo, rareza) |