**Corrección Estrategis.md**

Justificacion de la estrcutura de datos utilizada.

* Estructura de datos: Pila Dinamica
* Uso: utilizamos una pila dinamica llamada pilaA para apilar el mazo de cartas. Luego utilizamos una pila dinamica llamada pilaB la cual apilaba las cartas descartas, es decir, las cartas desapiladas de la pilaA.

Justificacion de porque no compilaba el juego

* El problema fue que funcionesAPI.c no estaba incluida en el proyecto (No estaba en el gitHub).

Correcion URL harcodeada.

* El problema fue en las funciones verRanking y enviarResultadoAPI:
* Esta linea (perteneciente a enviarResultadoAPI) se ve como la URL esta hardcodeada, es decir, está directamente escrita a mano y pasada como paramentro. Lo que se deberia haber hecho es haber leido el archivo config.txt, extraer la URL, almacenarla en una variable y luego enviar esa variable a la funcion enviarResultadoAPI.



* Lo mismo con la funcion verRanking. Vemos como la url esta harcodeada pero el codigoGrupo no porque ese si fue leido del config.txt y enviado como paramentro a la funcion



Posible resolucion al problema de leer JSON crudo.

* Investigando (chatGPT) encontre una posible solucion que consiste en descargar una libreria llamada cJSON.h y cJSON.c, la cual ayuda la manipulación de datos JSON en C. Y lo que habria que hacer es:

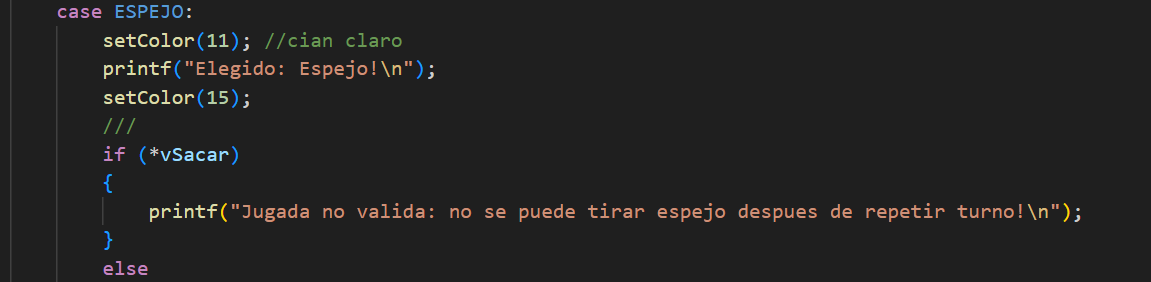
- Capturar la respuesta JSON como un texto en una cadena de caracteres.

* Parsear ese texto con cJSON\_Parse(). Esta funcion es como que transforma el contenido en un array o algo asi y nos permite recorrer y consultar
* Recorrer ese array extrayendo el campo del nombre del jugador y su puntaje
* Imprimir esa informacion con un formato agradable.

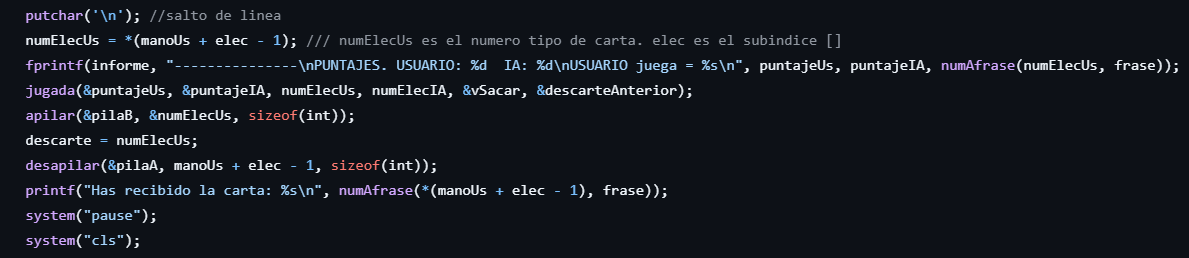
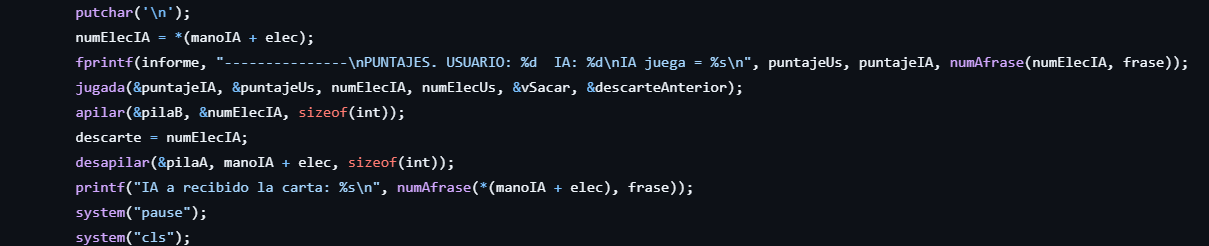
Posibles mejoras y como optimizarlo.

Creo que las posibles mejores ya son las mencionadas. Podria haber mas.

Como optimizarlo:

* Revisamos archivo por archivo.
* En menu.c: Creemos que no hay nada por optimizar, esta bien implementado el menu con el switch de los diferentes casos de la dificultad que se elegio.
* Ranking.c: Nada
* Funciones.c: Nada, son funciones de utilidad general
* FuncionesAPI.c: Aparte de la correcion de la URL hardcodeada nada mas.
* Jugadas.c: dentro de esta funcion, en el case de ESPEJO, este if no se ejecuta nunca debido a que en el archivo partida.c la variable vSacar se iguala a 0 en el comienzo de la funcion por lo tanto siempre va al else directo, una posible mejora seria tratar de este if se valide o sacarlo directamente ya que no aporta nada.
* Main.c: Nada para mejorar ni optimizar creemos nosotros. Se guarda correctamente en la variable tiempo el tiempo actual cuando se compilo y corrio el proyecto. Luego el switch valida correctamente la opcion de jugar, ver ranking o salir.
* Dificultades.c:
* En dificultad media:
* Demasiado if y else anidados. Esto se podria optimizar en separar el código en funciones más pequeñas y con nombres claros (por ejemplo, elegirCartaParaSumarPuntos, evitarRestadoras, tirarAleatoriaSalvoEspejo).
* Se repite mucho la validad de si es carta restadora o sumador. Una mejora/optimizacion seria crear una funcion auxiliar que haga la misma validacion.
* Se podria mejorar el nombre de las variables para que sean mas claras y descriptivas.
* Hay ciertos casos iguales donde se tiene que tirar una carta random que no sea espejo y se repite muchas veces, se podria optimizar con una funcion que reciba una lista de exclusiones y devuelva una carta aleatoria válida
* En dificultad dificil:
* Muy similar a la situacion de dificultad media. Se repite mucho la validad de si es una cierta carta en concreto, es decir, cuando es espejo o cartas sumadoras, restadoras, etc. Se podria mejorar con funciones auxiliares que validen esto
* La función tiene una lógica extensa y con muchos if-else anidados que dificultan su lectura. Se puede dividir en funciones más pequeñas y descriptivas. Como por ejemplo evaluarPresionAlJugador(), buscarCartaParaGanar(), responderConEspejo(), provecharRepetirTurno
* Mejorar en el nombre de las variables, que sean más descriptivas y claras.
* Partida.c:

Lineas de codigos repetidas: esto se puede simplificar incluyendolo en una funcion tal como:



Se podria incluir en una funcion como

Void procesarJugada(&pilaA,&pilaB,elec,manoIA,manoUs,...)

Esto podria ahorrarse unas cuantas lineas de codigo