Fájlkezelés

Modulok: FileHandler.h FileHandler.c

Feladatuk: Felelősek a kérdések dinamikus beolvasásáért.

Question: Dinamikus Láncolt lista elem, melyben a kérdések összes adatát tároljuk. Benne:
int – a játék nehézsége, azaz, hogy hányadik körben teszi fel a program.
char* qst – maga a kérdés
char* answ_A, answ_B, answ_C, answ_D – a kérdésre adható válaszok
char correct – a kérdéshez tartozó helyes válasz betűjele
char* category – a kérdés kategóriája
struct Question* next - a következő elemre mutató pointer
Tervezési megfontolások: A bemenet formájára nézve (több különöző típusú adat sorokban pontosvesszővel elválasztva) egyértelmű volt, hogy a kérdéseket struktúrában a legérdemesebb tárolni. Struktúrákat pedig láncolt listában a legérdemesebb tárolni.

Get_File(): Egy fájl tartalmát írja ki egy Question típusú listába. Visszatér: Eleje **Question***.

Get_String(FILE* fp):: Fájlból kiolvas egy stringet ';' jelig. Visszatér: **char***

Resize(char string, int size):** Segítő algoritmus stringek beolvasásához. Átméretez egy dinamikus stringet.

Kérdések adatkezelése:

Modulok: QuestionHandler.h QuestionHandler.c

Tartalmuk: Felelősek a kérdésekkel végzett feladatok elvégzéért játékon belül.

Free_Them_All (Question* first): felaszabadítja a megadott **Question** típusú láncolt listát.

next_question(Question* first, int diff, Question invalid, Question * inv_out):**
Kiválaszt egy adott nehézségi szintű nem invalid kérdést majd kiadja kimenetként. Visszatér: Question

ask_question(int round, Question *Q_pt, int *price_pool) feltesz egy kérdést a körnek megfelelő nehézségi szint és pénzösszeg szerint.

Játék:

Modulok: gameplay.h gameplay.c

Feladatuk: A játékban vezérelt felhasználó által nem befolyásolható folyamatok kezelése.

Bools struktúra a játék eseményeit dokumentáló logikai változókkal benne:
bool **quit** – kilépési feltétel vizsgálata igaz esetén kilépünk
used_vote – a szavazás funkció használatának viszgálata
used_phone – a telefonos segítség funkció használattát viszgája
used_halving – a felezés funkció használatának vizsgálata
won – igaz ha a játékos nyert
Tervezési megfontolások: A játék alatt több különböző logikai érték működik melyek a játékban történt eseményeket tárolják. Ezeket könnyebb egy strukturában átadni mint egyenként.

voting(Question *Curr_Q): A közönség véleményének a szimulálása és a válaszok kiírása a képernyőre.

halving(Question *Curr_Q): A helytelen válaszok felét elveszi, és kírja a maradék két választ.

ex_machina(Question *Curr_Q): Kiírja a helyes választ.

check(Question* Curr_Q, char answ): Egy megadott kérdésre adott válasz helyességét nézi meg a **answ** változó alapján. Visszatér: **bool**

Modulok: MenuHandler.h MenuHandler.c

Feladatuk: A játék különböző menüinek irányítása és a bementek kezelése.

New_Game: Magas szintű, a teljes játék vezérléséért felelős alprogram.
Feladatai: Nyert pénzösszeg eltárolása. Nehézség meghatározása.
Bemenetek: A, B, C, D (válaszlehetőségek), ‘h’ (**help** meghívása), ‘k’ (**voting** meghívása), ‘f’ (**halving** meghívása), ‘t’ (**ex_machina** hívása) ‘q’ (vissza a menübe)
Bemenet: Eleje **Question***, Jelenlegi felhasználó **User** struktúra **Kimenet:** Jelenlegi felhasználó.

Command_Menu(char user_command, Question *Q_pt, Bools *Game_State): Minden lehetséges bemenetet megviszgál és végrehajtja az utasításokat.

Main_Menu() – magas szintű eljárás ami a főmenüért felel. A ranglista, és maga a játék is innen érhető el. Parancsok: ‘r’ – (**Ranklist** meghívása) ‘h’ – (**help** meghívása) ‘n’ – (**New_Game** meghívása) ‘k’ – kilépés.

Ranklist() – Ranglistán szereplő profilok kiírása. A megadott parancs alapján.

Exception(char user_command): Addig idegesíti a felhasználót amíg nem ad meg helyes bemenetet. Visszatér: **char**

Menu_Exception(char user_command) – hasonló az **Exception** függvényhez, de a főmenühöz használatos. Visszatér: **char**

help(): Kiírja az összes lehetséges parancsot a játékban és a főmenüben.

Felhasználó management:

Modulok: UserHandler.h UserHandler.c

Feladatuk: A játékban használatos felhasználói profilok kezelése. Azokat fájlbol olvassa, fájlba írja illetve rendezi.

User struktúra amely egy felhasználó összes adatát tárolja. Benne:
char* **name -** felhasználónév
int **money** – a felhasználó által összesen nyert pénzösszeg
last_money – a felhásználó utolsó játékban szerzett pénznyereménye.
result – a felhasználó által összesen megválaszolt kérdések száma.
last_result – a felhasználó utolsó játékban összesen megválaszolt kérdéseinek száma.
struct User* next - a lista következő elemére mutate pointer.
Tervezési megfontolások: A **Question** strukturához hasonlóan a felhasználóknak is több különböző adata van így a struktúra alkalmazása és a láncolt lista kézenfekvő volt. A számértékek a könnyebb kezelhetőség és az átláthatóbb kód miatt külön változók és nem tomb elemek.

Put_User(User *first) – a felhasználók adatainak kiírása a **"users.txt"** fájlba.

Free_User(User *first) – a beolvasott felhasználók láncolt listájának felszabadítása.

Load_User(User **first) – új felhasználó megadása. Vagy meglévő felhasználó betöltése név alapján játék megkezdése előtt. Visszatér: **User ***

List_Sort(User *first, char user_command) - user_command alapján rendezi a listát.

Compare_By(User *first, User* second, char user_command) – a láncolt lista rendezésének feltételét határozza meg a **user_command** alapján. Visszatér: TRUE ha **first** – nek hátrébb kellene Lennie mint **second** – nek.

swap_data(User *first, User *second) – a láncolt lista két adott elemének adatait cseréli meg. Rendezéshez használatos.

All_Users – Minden felhasználó nevének beolvasása fájlból egy **User** típusú láncolt listába. Visszatér: **User ***

Din_Copy (char *copy) – dinamikus stringmásolás. Visszatér: **char ***

Fordítás lépései: Minden .c és .h fájl egy új projekthez adása, majd fordítás (GCC fordítóval tesztelve)

Szükséges külső könyvtárak: <stdbool.h> <stddef.h> <stdlib.h> <stdio.h> <ctype.h> <string.h> <stdarg.h> <time.h> "debugmalloc.h"