Aknakereschő

Egy cli aknakereső játék fájlban kezelt history-val, ami 60fps képfrissítéssel a terminálban pótolja a gui-t. A játékszabályok ugyanazok, mint bárhol máshol: méri az időt, ha név van beállítva a személyes rekordokat, ha aknára nyom véget ér a játék, stb stb.. Természetesen lehet guest-ként is játszani.

Menüpontok
A főmenüben lehet elindítani a játékot, kilépni, nehézséget és méretet állítani. Emellett a játékos nevét is megadni, eddigi rekordokat megtekinteni, és törölni, ha véletlen rosszul lett elmentve a player figyelmetlensége miatt.
A játékban x-ek fogják jelölni a még fel nem fedezett területeket, amelyeket meg lehet jelölni, és ha meg vannak jelölve * lesz belőlük (mint a valódiban a flag). Sor x Oszlop logikán fog alapulni a játék, tehát úgy lehet “megnyitni” egy aknát, hogy terminálba beírjuk:

12:3 (ez a 12. sor 3. oszlopa)

Ezután vagy megnyitjuk, vagy beflageljük. Természetesen ha megnyitjuk és akna, véget ér a játék. Ha flageljük, akkor toggle-ként funkcionál. A már megnyitott aknával nem tudunk semmit csinálni.
A map generálását úgy fogom megoldani, hogy a beállításokban megadott érték alapján elhelyezek a pályán csomópontokat, amelyekben/körül random számú (ezt is a settings határozza meg) akna generálódik. Az aknákat struct-ban fogom tárolni, valószínűleg így:

```
int x;
int y;
int isBomb; // 0 vagy 1
int isOpen;
int isFlagged;
int around; // mennyi van körülötte
```

Az aknák listájának dinamikusan fogok allokálni memóriát, majd a játék végén felszabadítom a listát és az aknákat is, mivel azok is malloc-olva lesznek. Minden megnyitott aknán szerepelnie kell, hogy a körülötte lévő 8 aknából mennyi bomba, amelyet egy saját egyszerű algo segítségével oldok meg. A map generálásakor ki lesz számítva, és csak azoknál a mezőknél írja a számot, amelyek meg vannak nyitva. Ergo nem kell minden rendernél kiszámolni újra és újra.

A játék akkor fog végetérni, amikor:
– egy isBomb = 1 akna megnyitásra kerül
OR
– az összes mező ami isBomb = 0, meg van nyitva

Ilyenkor ha nem guestként játszunk, elmenti a nevünkkel a menetet a history.csv-be, ami így fog kinézni:

```
name,time,difficulty
sefsef,174345244,144
```

A time az epoch másodpercben, a difficulty a sor x oszlop; tehát ha 12 sor x 12 oszlopos nehézségen játszott, akkor 144 lenne. Ha már létezik a fájl, appendeli, ha meg nem akkor létrehoz egyet indításkor üresen, csak a fejléccel. Ha rossz a formátuma a fájlnak, kitörli, előzőhöz hasonlóan létrehoz üreset és berakja a fejlécet.

A játékot modulokra szedtem szét, mert így átláthatóbb lett és a későbbi bővítések is könnyebbek. Külön modul foglalkozik a pályagenerálással, külön a játék logikájával, külön a felhasználói felülettel (ami most ugye terminál), és külön a fájlkezeléssel is. A debugmallocot minden .c fájl elején include-olom, mert a memóriahibák kiszűrése miatt kötelező, de ezen kívül nem kell külön meghívni semmit belőle.

A felhasználói parancsok nincsenek túlbonyolítva: a "m" a megnyitás, az "f" a flag, és ezekhez mindig kell egy sor:oszlop formátumú hely. A hibás bemeneteket lekezelem, és nem törlöm olyan gyorsan a képernyőt, hogy a hibaüzenetek eltűnjenek, így azt is látja a játékos, ha valamit rosszul írt be. A "q" csak akkor lép ki, ha tényleg csak "q", és nem valami random hosszú szöveg, ami q-val kezdődik.

A pálya megjelenítését is megcsináltam úgy, hogy szépen legyen igazítva, mert különben 10 feletti sor- és oszlopértéknél hamar elcsúszik az egész.

Az eredmények fájlkezelése egy history.csv-ben megy, és minden indításkor ellenőrzi, hogy a fájl egyáltalán érvényes-e. Ha nem, újra megcsinálja. A fájlhoz új eredményeket mindig hozzáfűz, de csak győzelem esetén és akkor is csak akkor, ha nem guest a név.

A program a modularitás miatt elég átlátható lett: külön ui, külön játékmenet, külön beviteli rész, külön pályakezelés, külön history. Ez a gyakorlatvezető szerint is így szebb és kezelhetőbb.

A memóriakezelést is rendbe raktam: minden malloc-olt dologhoz van free, és a debugmalloc nem ír hibát, ha normálisan fut végig a játék. A tábla soronként malloc, soronként free, így nincs leak, debugmalloc is minden c fájlban include-olva.

A játékmenet egyszerű: a játék addig megy, amíg vagy bomba nyílik meg, vagy minden biztonságos mezo ki nincs nyitva. Ekkor ha kell, menti az eredményt, és visszadob a menübe

Program fordítása:

```
gcc main.c ui.c parancs.c board.c game.c history.c -o aknakereso
```