Programozói dokumentáció aknakeresőhöz

Nehézség választása

Az első lépés a játék indítása után, hogy a játékos kiválasztja, mekkora táblán szeretne játszani. Ezt a főprogram tartalmazza, érvénytelen nezézség kód esetén új választást kér, egészen addig, amíg nem megfelelőt adnak meg. Van lehetőség a megadottaktól eltérő egyéni nehézség beállítására, itt csak arra kell figyelnie a programnak, hogy ne lehessen több akna, mint mező. Erre felhívja a felhasználó figyelmét hibás akna darabszám esetén.

A tábla generálásához tartozó függvény

A függvényeket tartalmazó modulban van a játék kezdetekor a mezők elkészítéséért felelős függvény. A függvény paraméterként a kiválasztott nehézséghez tartozó adatokat veszi át, és egy pointerrel tér vissza az egész kétdimenziós tömbre. A dinamikus tömb használata jó választás, mivel lépésenként módosulni fog valamennyit a tömbünk, így elég csak átírni a meglévő tömböt, nem kell minden lépésnél újra létrehozni azt, vagy kettő tömböt használni

Aknamentes terület 0-val van jelölve, az aknákat a random szám generátor helyezi el. Ha esetleg már aknát tartalmazó koordinátát generálna újra, amire kisebb nehézség esetén (pl 8x8-as táblán) elég nagy esély van, akkor újra generál koordinátát, amíg az nem üres mezőre mutat. Ezt oldja meg a feltétel, ami csak nullát tartalmazó mező esetén tesz le aknát és növeli a számlálót.

A tömb megjelenítése

Az aknák rejtettek, az őket jelölő (-1)-eseket tartalmazó mezők csak elvesztett játék esetén válnak láthatóvá, a „robban” függvény meghívására. Nehéz módban a tábla már egészen nagy ahhoz, hogy kisebb kijelzőkről kilógjon, ezért a két karakter hosszú szám ([-1]) helyett dobozba rakott csillag karakterrel ([*]) érdemes jelölni a bombák helyét. Az üres terület pedig egy dobozban lévő szóköz ([ ]). Így egyenlő szélességűek, négyzetes táblát fognak kirajzolni, nem lesz elcsúszás. Ha például teszteléskor szeretnénk látni az aknák helyét, a fuggvenyek.c fájlban a matrix_kiir() függvényben kell átírnunk a (-1) esetét például csillagra, vagy bármilyen egyjegyű karakterre.

Az aknák megkeresése

A legelső lépést teljes mértékben a szerencsére bízzuk, már ilyenkor is lehet aknára lépni, hiszen a tábla generálása már az első tipp előtt megtörténik. Ebben eltér az eredeti aknakereső szabályaitól, ott nem lehet veszíteni az első lépésben. Ezután kilogikázva lehet továbbhaladni, a szomszédos mezőkön lévő aknák darabszáma alapján. A feltárás koordinátánként történik, először a sor koordinátáját, majd az oszlopét kéri be a tipp() nevű függvény a billentyűzetről. A tipp elnevezés nem feltétlenül a legpontosabb, hiszen aki ismeri a játékszabályokat, az általában nem tippelve játszik, mégis ezt használja a program, az egyértelműség kedvéért. Tippelés után a tömbben eltárolt adatok kicsit módosulnak, és minden tippet követően kiírjuk, hogy a játékosnak minden alkalommal egyértelmű legyen, hol tart.

Fájlba mentés

A játék során, amikor a játékos úgy dönt, hogy fájlba szeretné menteni a játékállást, a sor koordinátája helyett (-1)-et ad meg. Ekkor létrejön 3 db .txt fájl. Az elsőben a nehézséghez tartozó sorok száma, a másodikban a nehézséghez tartozó oszlop száma, hiszen ezek változnak, és a tábla visszahívása előtt szükség van rájuk. A harmadik szövegfájlban pedig maga a játéktábla van elmentve, 0-val jelölve az üres és még fel nem tárt területeket, 'a' karakterrel a már feltárt üres, vagyis 0-ás területeket, továbbá 1-8-ig a már kiszámolt szomszédokat, és 9-essel az aknákat.

Lépés, nyerés vagy vesztés

A lépés a main()-ben egy while() loopban található, és kettő különböző esetben ér véget a játék:

(1) A játékos aknára lépett: ezt minden lépés után ellenőrzi egy feltételes elágazás. Ha így történt, meghívódik a game_over() függvény, és felfedi a táblát [*] és [ ] karakterekkel. Ebben az esetben vége a programnak, a függvény felszabadítja a lefoglalt memóriaterületeket is, hiszen már nincs rájuk szükség.
(2) A játékos megnyerte a játékot, ezt is minden lépés után ellenőrzi egy feltételes elágazásban lévő nyeres_check() függvény. Ez a függvény végigmegy a táblán, és megszámolja a nullákat, azaz a még ismeretlen területeket. Ha ezek száma nulla, a játékos minden területet feltárt, vége a játéknak és nyert. Ez a függvény szintén felszabadítja a lefoglalt területeket.

Amennyiben ezek nem teljesülnek, folytatódik a tippelés. Dinamikus tömbben tárolja a tippek koordinátáit, a végén ezért ezt is fel kell szabadítani.

Feltárás

Ha helyes lépést tett a játékos, akkor meghívódik a szomszedszamolo.c fájlban található loop() nevű rekurzív függvény. Ez a függvény felelős az adott terület feltárásáért. Minden 0-t tartalmazó területet, és annak határain lévő értékeket kell feltárnia. Ehhez mind a nyolc lehetséges irányba elindul, és megállapítja, hogyha a szomszéd terület 0 (vagy -2). Ezesetben ugyanis a szomszéd szomszédját is meg kell vizsgálni, itt használja a rekurziót, meghívja önmagát, ha a szomszéd cellának is 0 db szomszédja akna. A sarkok eseteit a szomszed_szamolo() függvény már helyesen kezeli, csak arra kell figyelnie a függvénynek, hogy ne menjünk ki a tábláról.

Fájlból beolvasás

A nehézség választás részben kell megadnia a játékosnak az 5-ös kódot, ekkor a program a saját mappájában megkeresi a save.txt, a nehez_oszlop.txt és a nehez_sor.txt fájlokat, ha léteznek. Először a sor és oszlop változókba eltárolja a megfelelő mennyiségeket. (Ilyenkor az aknák számára végülis nincsen szükség.) Ezután iterál végig a save.txt-ben lévő számok tömbjén, hiszen már tudja, milyen hosszan kell végigmenni. Ezek a fájlok mindig felülíródnak, tehát mindig a legutolsó mentést lehet csak meghívni. Ha még nincs mentés a mappában, NULL pointerrel visszatérve vége a programnak. A tábláról az adatokat átkonvertálja betűkké, mert persze a -2-t is például 'a'-val jelöltük, ezért kellett egy dekódoló, ami például az új sort sem veszi figyelembe karakterként, az 'a' betűt visszaalakítja (-2)-re, és a betűként értelmezett számokat újra int-ekké.