Szakdolgozat

AKNAKERESŐ JÁTÉK ALGORITMUSOK VIZSÁGALATA

HUSOCZKIDANIEL@GMAIL.COM

1. Bevezetés

Bárki, akinek a gépén Windows 7 vagy az előtti Windows operációs rendszer volt/van legalább egyszer találkozhatott már az aknakereső játékkal.

De mi is tulajdonképpen az aknakereső?

Az aknakereső egy eléggé elterjedt egyszemélyes számítógépes játék, amely tulajdonképpen egy panelekkel teli táblával állít be a játékos elé. A megjelent panelek bármelyikére rá lehet kattintani, viszont vigyázni kell, mivel a panelek vagy egy számot rejtenek nullától-nyolcig vagy pedig egy bombát.

1.ábra: Példa a megjelenített panelek alatt található számokra és bombákra

Ha a játékos egy olyan panelre kattint, ami alatt bomba van akkor véget ér a játék és kezdheti az egészet előről. Ha pedig a panel egy számot rejt, akkor a szám alapján a játékos láthatja, hogy a jelenlegi mező körüli mezőkben mennyi bomba van elrejtve. Azonban, ha a panel nullát rejt, akkor a játék megjeleníti az összes szomszédos mezőt, addig ameddig nem lesz mindegyik legalább egynél nagyobb szám. A mezőket zászlókkal is meg lehet jelölni, amellyel a játékos saját magának ki tudja jelölni azokat a mezőket, amelyek alatt szerinte bomba lehet.

A játék célja megtalálni az összes bombát, anélkül, hogy bármelyik is felrobbanna.

2.ábra: Példa a játéktérre. A szürke mezők nullákat jelentenek.

A játékban alapvetően nehézségi fokozatok is vannak, be lehet állítani a tábla méretét, vagy a bombák számát. A nehézségi fokozatot a játékos állítja be, például egy nagyobb táblán kevesebb bombával sokkal könnyebb játszani, mint egy kisebb táblán több bombával.

2. A játék technikai háttere

2.1 Kihívások

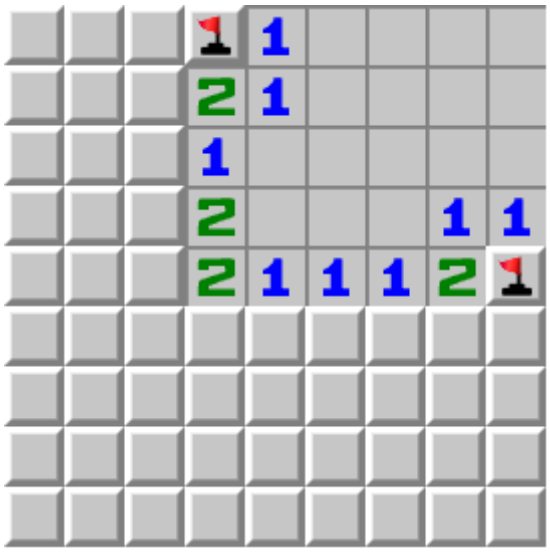
Az aknakereső megoldásának két fő kihívása van, és mindkettő magával a játék felépítésével kapcsolatos alapvető problémákkal foglalkozik.

Az aknakereső egy NP-teljes probléma

Először is be kell látnunk azt, hogy nem létezik olyan automatikus aknakereső megoldó algoritmus, ami bármikor is meg tudná oldani az összes lehetséges játéktáblát. Ez azért lehetséges, mivel az aknakereső egy tudományosan bizonyított NP-teljességi probléma, ami azt jelenti, hogy az összes lehetséges tábla kiszámítása lehetséges, viszont az rendkívül sok időt venne igénybe

Vannak esetek amikor találgatni kell

Léteznek olyan esetek a játékban, amelyek rá erőltetik a játékosra azt, hogy tippeljen, például amikor nem teljesen világos a játékos számára, hogy melyik mezőre kattintson a következő lépésben, vagy melyik mezőt jelölje meg bombának.



3.ábra: Ebben az esetben lehetetlen algoritmikusan eldönteni, hogy mi legyen a következő lépés. Egy lehetőség van, a tippelés.

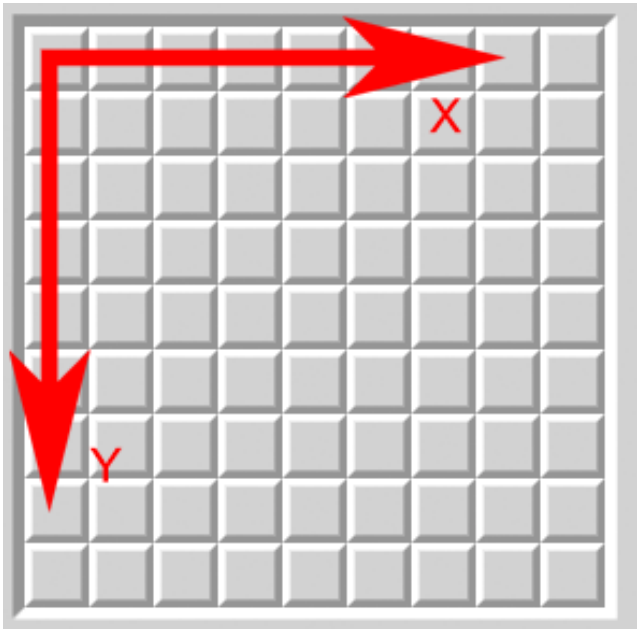
2.2 A játék megalkotása

Mielőtt használnánk az Aknakereső megoldó algoritmust, azelőtt még meg kell terveznünk, és el kell készítenünk magát az Aknakereső játékot.

Először is szükségünk lesz az alapból látható objektumok osztályaira.

Mezők / Panelek

A panelek a játéktáblán lévő kis mezők, amelyekre a játékos kattintani tud. A panel osztálynak több tulajdonsággal is rendelkeznie kell, elsősorban egy ID-vel, amely segít megkülönböztetni a panelokat. Mivel egy kettő dimenziós játékot készítünk, ezért a panelek tulajdonságai közt szerepelnie kell X és Y koordinátáknak is, amellyel meg tudjuk határozni, hogy melyik panel, hol szerepel a játéktáblán.



4.ábra: A játéktábla X és Y tengelye.

Szükség van egy Bomba logikai tulajdonságra is, amivel meg lehet adni, hogy a mező bomba-e vagy sem, és egy Panel körüli bombák tulajdonságra, ami, hogy ha nem bomba a mező, akkor megadja azt a számot, amennyi bomba van a mező körül. És végül kell még egy „helyzet” jelző enum, amely megadja, hogy a mező rejtett, felfedett vagy zászlózott helyzetben van.

A panel konstruktorában, csak az ID-t az X és az Y koordinátákat kell beállítani, akkor amikor le generáljuk őket, majd a játék során változik a többi tulajdonság is.

Játéktér / Játéktábla

A játéktér vagy játéktábla, az a hely, ahol a panelek fel vannak sorakoztatva egymás mellett, a játéktérnek viszont van még egy funkciója, mégpedig az, hogy nyomon kövesse a játék állapotát.

A játéktábla megalkotásakor elsősorban azt szeretnénk, hogy bármekkora játékteret létre lehessen hozni, így szükségünk lesz a méretekre, amelyhez szükségünk van Magasság és Szélesség tulajdonságokra, ezekkel az oszlopokban és a sorokban elhelyezkedő panelek számát adjuk meg. Viszont meg kell adnunk még a bombák számát is, ezért ennek is kell egy BombaSzám tulajdonság. Ezeken kívül még szükség lesz egy Paneleket tartalmazó listára, amely Magasság * Szélesség darab panelt fog eltárolni és végül a játék állapotát tartalmazó változóra, ami egy enum lesz.

A játék állapota lehet, Folyamatban, Befejezett, vagy Vesztett.

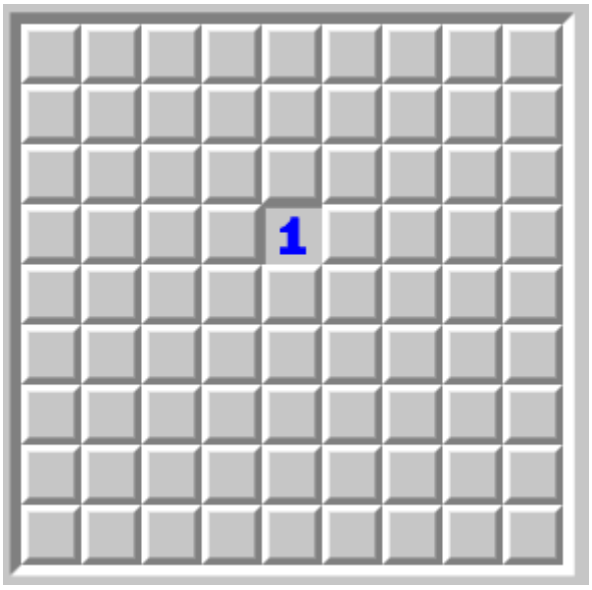
Az első lépés

A játékos első lépése teljesen véletlenszerű, hisz a játék kezdetekor egy teret lát rengeteg mezővel, így nincs semmilyen támpont arról, hogy melyik mezőre kellene kattintania.

Elég idegesítő lenne, hogy ha rögtön az első lépéskor a játékos bombára kattintana és elvesztené a játékot, ezért erre egy megoldást kell találni.

Ha a bombák, akkor generálódnának le, amikor elindítanánk a játékot, akkor ez a probléma valós lenne, és elég nagy lenne az esély arra, hogy bekövetkezzen, viszont erre a megoldás az, hogy a bombákat nem akkor generáljuk le amikor elindítjuk a játékot, hanem akkor, amikor a játékos megtette az első lépését.

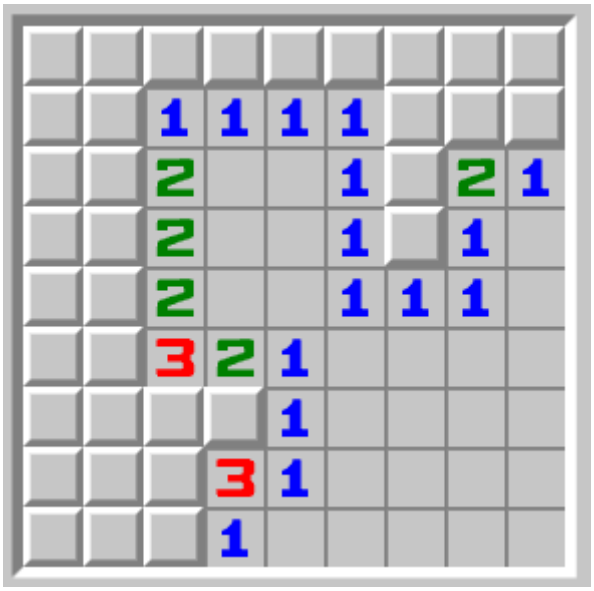
Azonban úgy is elég nehéz lenne elkezdeni a játékot, ha az első kattintás után, csak egy mezőnek az értéke jelenne meg.



5.ábra: Nehéz kezdés.

Ha a játékos az első lépése után egy ilyen táblával találná szembe megát, akkor szinte biztos, hogy nem tudná eldönteni, hogy melyik lenne a következő legbiztonságosabb lépés, és ismét véletlenszerűen kellene kattintania egy mezőre, ami azonban most már lehet bomba is.

Erre a problémára a megoldás pedig az, hogy az első lépéssel, nem egy, hanem több mezőt is felfedne a játékos. Ez úgy lehetne lehetséges, hogy az elősször kiválasztott mező körül véletlenszerűen nullák lennének, és mivel a nullák körül nem helyezkedhet el bomba, ezért az azok körüli mezőket is felfedné a játék, ezzel egy támpontot ad a játékosnak a kezdéshez.



6.ábra: Könnyű kezdés.

3. Már elkészült megoldó algoritmusok

3.1 Egypontos algoritmus (Single Point Algorithm)

Az egyik leggyakrabban használt algoritmus, amit komplex feladatok megoldására használnak. Az egypontos stratégia (Simple Point Strategy – SPS) egy kényszer elégedettségi problémát old meg, úgy, hogy nem ellenőrzi az ívkonzisztenciát, ezzel viszont a legkevésbé CPU igényes megoldás lesz, viszont ez azt is jelenti, hogy ha a táblának van megoldása, akkor az nem garantált, hogy az SPS megtalálja azt.

Az algoritmus két lépésből áll, amit minden iterációban elvégez:

- Az algoritmus minden még nem felfedett mezőn állva összehasonlítja a mező közelében lévő aknák számát, a mező közelében lévő megjelölt „zászlózott” aknák számával. Ha a két szám egyenlő, és a szomszédok között vannak még nem felfedett mezők, akkor azokat biztonságosként jelöli meg.
- Minden biztonságos mező esetén, az algoritmus összehasonlítja a közelben lévő aknák számát a még fel nem fedett szomszédos mezők számával, ha a két szám megegyezik, akkor az összes fel nem fedett szomszédos négyzetet megjelöli bombának.

Az algoritmus akkor ér véget, ha a játéktér egy iteráció után már nem változik meg, ami azt jelenti, hogy nincs biztonságos mező. Ekkor egy komplexebb megoldó algoritmusra kell átváltani, viszont, ha az az algoritmus talál egy biztonságos mezőt, akkor visszatérhetünk az egypontos megoldóhoz.

3.2 Kényszer elégedettségi probléma stratégia (Constraint satisfaction problem Strategy)

C. Studholme egy olyan algoritmust készített, amely 7 lépésből áll. A játéktérben minden állapot kényszer-elégedettségi problémaként van implementálva a megszorítások pedig egyenletek halmazaként vannak ábrázolva. Ezek az egyenletek le vannak egyszerűsítve, és azonos változókat tartalmazó egyenletekre vannak osztva. Az egyenleteket egy visszakövető algoritmus oldja meg. Minden változóhoz egy érték kerül hozzárendelésre, minden ilyen hozzárendelés után pedig ellenőrizni kell a megszorításokat. Ha az egyenletet még meg lehet oldani, akkor ismét egy hozzárendelésre kerül sor, viszont, ha nem lehet megoldani, akkor visszalépünk az előző konfigurációra. Ha bármely változó tartomány egy lesz, akkor új lépés következik, és új megszorítások jönnek létre.

A kényszer elégedettségi stratégia algoritmus képes sokkal több játéktáblát megoldani, sokkal gyorsabban, mint bármelyik algoritmus ez előtt.

Fontos megjegyezni azt, hogy Studholme algoritmusában léteznek esetek, amikor az algoritmus egy nem teljesen biztonságos mezőt választ következő lépésnek, mielőtt egy teljesen biztonságos mezőt találna. Ez olyan helyzetekben fordul elő, amikor a játéktábla adott területén nem képes más információt találni, viszont lépnie kell.

A dokumentum elkészítéséhez használt anyagok:

- Solving Minesweeper with C# and LINQ
- Algorithmic Approaches to Playing Minesweeper