

# **AKNAKERESŐ**

## **Dokumentáció és leírás a teljes feladathoz**

# Tartalom

|   |    |
|---|----|
| 1. A program rövid ismertetése .....                    | 4  |
| 2. Használati utasítás .....                            | 6  |
| 2.1 A menüsáv .....                                     | 6  |
| 2.2 Játék indítása.....                                 | 7  |
| 2.2.1 A kezdőképernyő .....                             | 7  |
| 2.2.2 Egyedi nehézségi szint .....                      | 7  |
| 2.3 Játék .....   | 8  |
| 2.4 Új játék indítása, miután a játék véget ért .....   | 12 |
| 2.5 Eredménytábla megtekintése .....                    | 13 |
| 2.6 Kilépés a programból.....                           | 13 |
| 3. Tervezési döntések .....                             | 14 |
| 3.1 Adatszerkezetek.....                                | 14 |
| Az aknamező tárolása .....                              | 14 |
| Megjelenítendő mezők .....                              | 14 |
| Gombok tárolása .....                                   | 14 |
| Pontok tárolása egy játékmenetből .....                 | 14 |
| Pontlista.....  | 14 |
| 3.2 A grafikus interfész felépítése, sajátosságai ..... | 15 |
| GameFrame .....   | 15 |
| StartScreen .....                                       | 15 |
| GameInstance.....                                       | 15 |
| StatusBar .....   | 15 |
| GameOverBar .....                                       | 15 |
| Editor.....   | 15 |
| ScoreBoard.....   | 15 |
| 3.3 Forrásfájlok és tartalmuk .....                     | 16 |
| 4. Osztályok.....                                       | 17 |
| Osztálydiagram.....                                     | 17 |
| 4.1 Minefield .....                                     | 18 |
| 4.2 GameInstance .....                                  | 19 |
| 4.3 GameFrame .....                                     | 22 |
| 4.4 StartScreen.....                                    | 23 |
| 4.5 Editor .....  | 24 |
| 4.6 StatusBar.....                                      | 25 |

|   |    |
|---|----|
| 4.7 GameOverBar .....                               | 26 |
| 4.8 ScoreBoard .....                                | 26 |
| 4.9 FieldListener .....                             | 27 |
| 4.10 ScoreInstance .....                            | 28 |
| 4.11 ScoresData .....                               | 30 |
| 4.12 ScoreComp .....                                | 31 |
| 4.13 MenuListener .....                             | 31 |
| 4.14 StartScrButtonListener .....                   | 32 |
| 4.15 EditorButtonListener .....                     | 32 |
| 4.16 SliderListener .....                           | 32 |
| 4.17 InstantRestartButtonListener .....             | 33 |
| 4.18 ToggleButtonListener .....                     | 33 |
| 4.19 TimerListener .....                            | 34 |
| 4.20 Main .....                                     | 34 |
| 5. Use-casek, interakciók .....                     | 35 |
| 6. Fájlkézelés .....                                | 35 |
| 7. Tesztesetek leírása .....                        | 36 |
| Beadott tesztállományok listája .....               | 37 |
| 7.1 CheckIndexTest .....                            | 38 |
| 7.2 MarkerTest .....                                | 38 |
| 7.3 MinesLeftStatusBarTest .....                    | 38 |
| 7.4 MinesMarkedTest .....                           | 38 |
| 7.5 PlaceMinesTest .....                            | 38 |
| 7.6 TestButtonBoardDimensions .....                 | 38 |
| 7.7 TestScoreSorting .....                          | 39 |
| 7.8 ToggleMarkerTest .....                          | 39 |
| 7.9 UncoverTest .....                               | 39 |
| 7.10 UpdateBoardValuesTest .....                    | 39 |
| 8. A futáskor elvárt könyvtárszerkezet .....        | 40 |
| 9. Munkanapló .....                                 | 41 |
| 10. Saját gondolatok a feladat elvégzése után ..... | 41 |

# 1. A program rövid ismertetése

Az Aknakereső nem más, mint egy ismert logikai játék. Azért választottam ezt a feladatot, mivel egyrészt kedvelem az eredeti játékot, másrészt kíváncsi voltam milyen lenne a saját ötleteim alapján elkészíteni egy hasonmását.

Egy aknamezőről két fontos tulajdonságot szeretnék előre definiálni, ugyanis ezekre többször is hivatkozni fogok ebben a dokumentumban:

-n: az aknamező méretét adja meg, mégpedig olyan módon, hogy az aknamezőt egy  $n \times n$ -es táblaként képezzük le

-k: az aknamezőn véletlenszerűen elhelyezett aknák számát adja meg

A játék menetét a korábban beadott specifikációban már egyszer megfogalmaztam:

„Ez gyakorlatilag egy logikai játék, viszont előfordulhat olyan helyzet, ahol a játékosnak tippelnie kell. A lényege az, hogy egy aknamezőn kell megkeresni a véletlenszerűen elhelyezett robbanószerkezeteket. A játéknak 3 különböző, kezdéskor választható nehézségi szintje van. Az aknamező egy  $N \times N$  rublikával rendelkező táblaként van mevalósítva. Ezekre a rublikákra a továbbiakban a „mező” névvel fogok hivatkozni. A mezők a játék kezdetekor lefedett, ismeretlen állapotban vannak a játékos számára. A mezők felderítése az egér használatával, a mezőkre való klikkeléssel történik. Új játék indításánál az első klikk után a program a véletlenszerűen elhelyez a mezőkön a nehézségi szinttől függő fix számú aknát. Az első választott mező sosem lehet bomba. Amennyiben a játékos olyan mezőre kattint, amire a játék aknát helyezett el, a játék véget ér. Amennyiben nem, a mező attól függően, hogy hány bomba van az azt körülvevő mezőkön (8 mező, oldalra és átlóban) a következőket fedheti fel: üres mező, ami körül nincs bomba, vagy egy szám, ami a körülvevő bombák számát jelzi. A játékos jobb egér klikkeléssel megjelölheti azokat a mezőket, ahol szerinte bomba van. Ha a játékos felfedte az összes olyan mezőt amin nem volt bomba, a játéknak vége, nyert. A grafikus felületen látszódik továbbá a játék kezdete óta eltelt idő másodpercben, illetve az elhelyezett bombák számának és a játékos által megjelölt mezők számának a különbsége. A program menüjéből elérhető az eddigi játékok eredménye, amely tartalmazza a játékos által megadott nevet, a nehézségi szintet, és a játék hosszát másodpercben.”

**A beépített nehézségi szintek:**

- Könnyű: 6x6 pálya, 10 akna
- Közepes 8x8 pálya, 18 akna
- Nehéz 10x10 pálya, 25 akna

A laborvezetőmmel való konzultálás után a következő változtatásokkal szeretném kiegészíteni a fenti leírást:

-a mezők megjelölése nem jobb egérklikkel történik, hanem a játékterület feletti státuszsávban elhelyezett gomb megnyomásával lehet váltani, hogy egy mezőre való kattintás megjelölje vagy felfedje a mezőt (tehát a jobb egérklikkhez a programban nem tartozik semmilyen funkció)

-a 3 beépített nehézségi szint mellett lehet egyedi (n,k) paraméterekkel is aknamezőt létrehozni

Ezekén kívül megnőveltem az **eredménytáblában eltárolt adatok** mennyiségét, így a tárolt adatok:

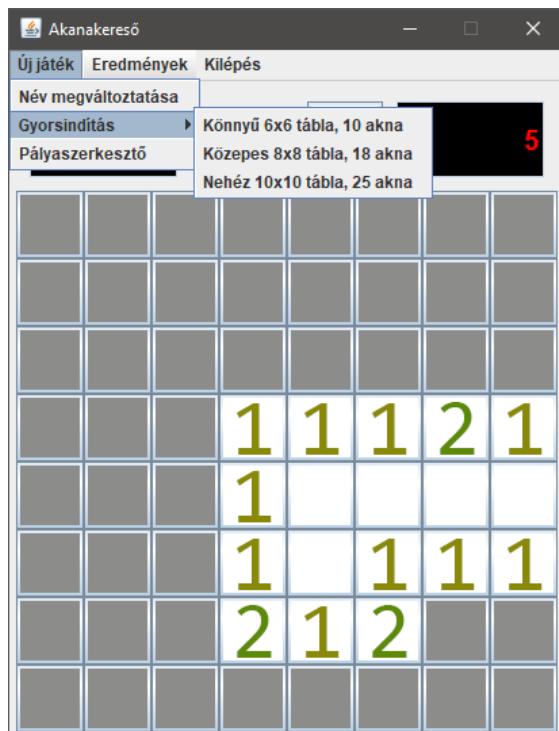
- játékosnév
- sikeres volt-e a játék
- nehézségi szint
- játékidő
- pálya mérete
- aknák száma
- játékos által helyesen megjelölt aknák száma

A program vezérlését megpróbáltam lehetőleg egyértelművé, valamint a felületet átláthatóvá tenni. A következő pontban a programmal való interaktálást szeretném részletesen kifejteni.

## 2. Használati utasítás

A program használata közben különböző felületek jelennek meg a grafikus interfészen. Az alábbi leírás segíthet eligazodni rajtuk.

### 2.1 A menüsáv



A program futása közben a menüsáv folyamatosan látható és használható, csak ebben a pontban részletezem. A menüpontok a következők:

-Új játék

-**Név megváltoztatása:** Megjeleníti a kezdőképernyőt

-Gyorsindítás

-**Könnyű:** Elindít egy játékot

-**Közepes:** Elindít egy játékot

-**Nehéz:** Elindít egy játékot

-**Pályaszerkesztő:** Megjeleníti a pályaszerkesztő képernyőt

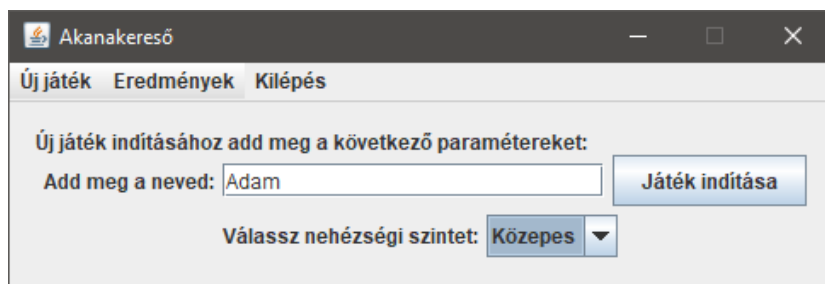
-Eredménytábla

-**Eredménytábla megjelenítése:** Megjeleníti az eredménytábla képernyőt

-**Kilépés:** Bezárja a programot

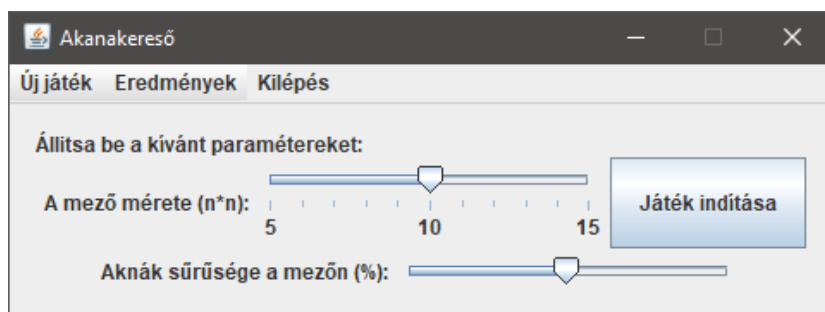
## 2.2 Játék indítása

### 2.2.1 A kezdőképernyő



A felület tartalmaz egy szövegmezőt, ahova a felhasználó beírhatja a nevét. Ha a szövegmező üres, a játékos nem tud továbblépni a képernyőről. A program megjegyzi a nevet, ameddig a kezdőképernyőre visszatérve meg nem változtatják azt, vagy be nem zárják a játékot. Indításkor az értéke „Player”. A legördülő menüből a játékos nehézségi szintet választhat. A választható szintek a feljebb definiált szintek, valamint az „Egyedi” szint, amit a következő alpontban ismertetek. A név megadása és a nehézségi szint megadása után elindítható egy játék menet a „Játék indítása” gombbal.

### 2.2.2 Egyedi nehézségi szint

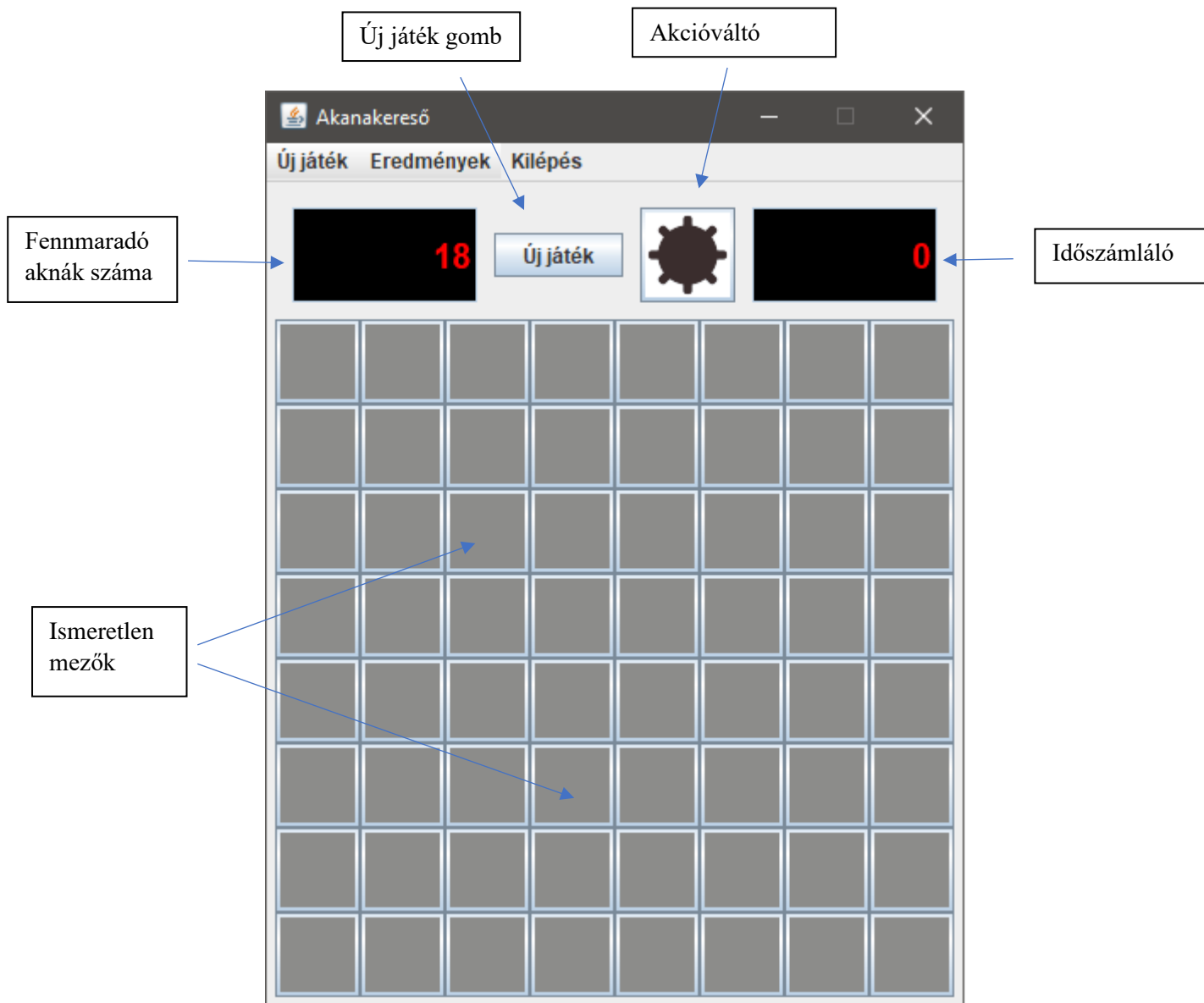


Ha a játékos a beépített nehézségi szintektől eltérő paraméterű pályát szeretne generáltatni, használhatja a pályaszerkesztő funkciót. Ez elérhető a menüből, valamint a kezdőképernyőről is ide kerülünk, ha „Egyedi” nehézséget választunk. Ezen a felületen a játékos egy csúszkával beállíthatja a pálya kívánt méretét. A választható tartomány: 5x5 – 15x15. A másik csúszkával azt lehet állítani, hogy a mezők hány százaléka legyen akna. ez legalább 15%, a felső korlátja viszont a méret függvényében változik. A legkisebb méretnél ez 60%, ugyanis ennél több akna nem helyezhető el minden kiindulási helyzetben. Ahogy nő a méret, a maximum százalék is nő. A „Játék indítása” gombbal elindul egy játékmenet a megadott paraméterekkel.

A játék ugyan engedi, de nem ajánlott szélsőséges k paraméterrel játszani, mivel a játék túl könnyűvé/lehetetlenné válik. Nagy méretű pálya esetén az ablak kilóghat a képernyőről, ha a felbontás kisebb mint 1920x1080, ugyanis a mezők fix mérettel jelennek meg.

## 2.3 Játék

Ha új játékot indítunk, a következő képernyő fogad minket:



A játék akkor indul el, ha rákattintunk bármelyik ismeretlen mezőre. A program ekkor generálja le a pályát, ügyelve arra, hogy a választott mezőre és szomszédjaira ne rakjon aknát. A számláló minden másodpercben eggyel nő. Ha olyan mezőre kattintottunk ami üres, tehát nincs a szomszédjain akna, akkor a program az összes szomszédos üres mezőt, illetve azok üres szomszédjait (... , rekurzívan) felfedi. Az egyes mezőkre kattintva felfedhetjük őket.



A játékos a mezőkön a következő képeket láthatja:

| Kép | Helyettesítő szöveg | Jelentés          |
|-----|---------------------|-------------------|
|     | ?                   | Ismeretlen mező   |
|     | F                   | Megjelölt mező    |
|     | 0                   | Üres mező         |
|     | ☒                   | Akna              |
|     | 1                   | 1 szomszédja akna |
|     | 2                   | 2 szomszédja akna |
|     | 3                   | 3 szomszédja akna |
|     | 4                   | 4 szomszédja akna |
|     | 5                   | 5 szomszédja akna |
|     | 6                   | 6 szomszédja akna |
|     | 7                   | 7 szomszédja akna |
|     | 8                   | 8 szomszédja akna |

Ha a program valamilyen okból nem tud betölteni egy képet, akkor a helyettesítő szöveget jeleníti meg.

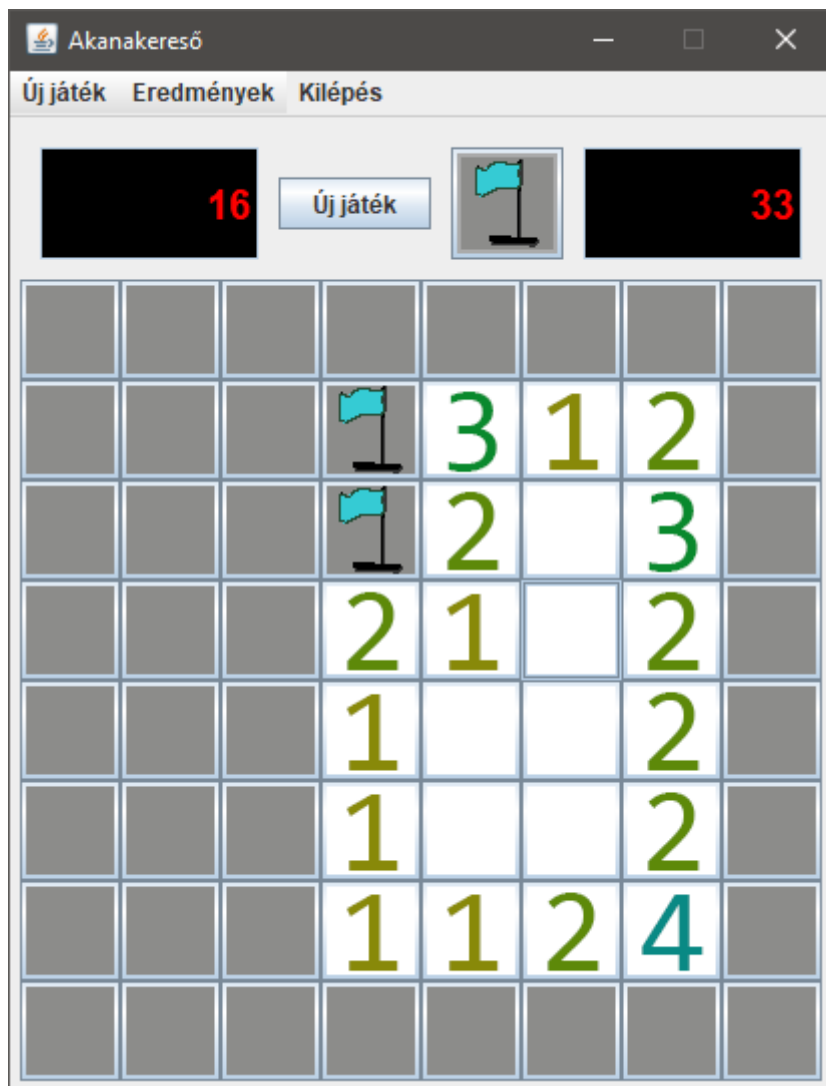
A játék célja felderíteni az összes olyan mezőt, amin nem akna van.

Ha a játékos felderít egy mezőt amin akna van, a játéknak vége, a játékos veszett. A program ilyenkor megmutatja az összes akna helyét, valamint a játéktól jobbra rövid statisztikát jelenít meg.



Az akcióváltó gombbal lehet váltani a jelölés és a felderítés között. Ha jelölő módban van a program, akkor egy zászló képe van a gombon, ha felderítő módban, akkor pedig egy aknát látunk rajta.

A megjelölt mezőkre kattintva (jelölő módban) visszavonhatjuk a jelölést.



Azokat a mezőket célszerű megjelölni, ahol szertintünk akna van.

Ha megjelölünk egy mezőt, akkor csökken a „Fennmaradó aknák” számát jelző érték.

A játékos bármikor instant új játékot indíthat az „Új játék” gombbal, ilyenkor az (n,k) paraméterek ugyanazok lesznek, mint az előző játékban.

Ha a játékosnak sikerült felfednie az összes biztonságos mezőt, a játéknak vége, a játékos nyert. Ha egy játék véget ér, a program automatikusan hozzáadja az elért pontokat az eddigiek listájához és elmenti azt egy fájlba. A játék végén a játékot jellemző adatok kerülnek megjelenítésre.



Nem kötelező az összes aknát megjelölni, a játék akkor ér véget, ha a „biztonságos” mezőket feloldották.

## 2.4 Új játék indítása, miután a játék véget ért



Ha véget ért a játék, megjelenik a JÁTÉK VÉGE panel. Ha vesztettünk, a program megmutatja az összes akna helyét. A számláló a játék végén megáll. Innen, ha szeretnénk akkor új játékot indíthatunk, akár instant, az „Új játék” gombbal, vagy a menüből testre szabhatjuk a következő játékunk paramétereit, esetleg módosíthatjuk a játékosnevet. Továbbá lehetőségünk van megtekinteni az eredménytáblát, vagy kiléphetünk a programból.

Ezek mikéntjét a következő pontok ismertetik.

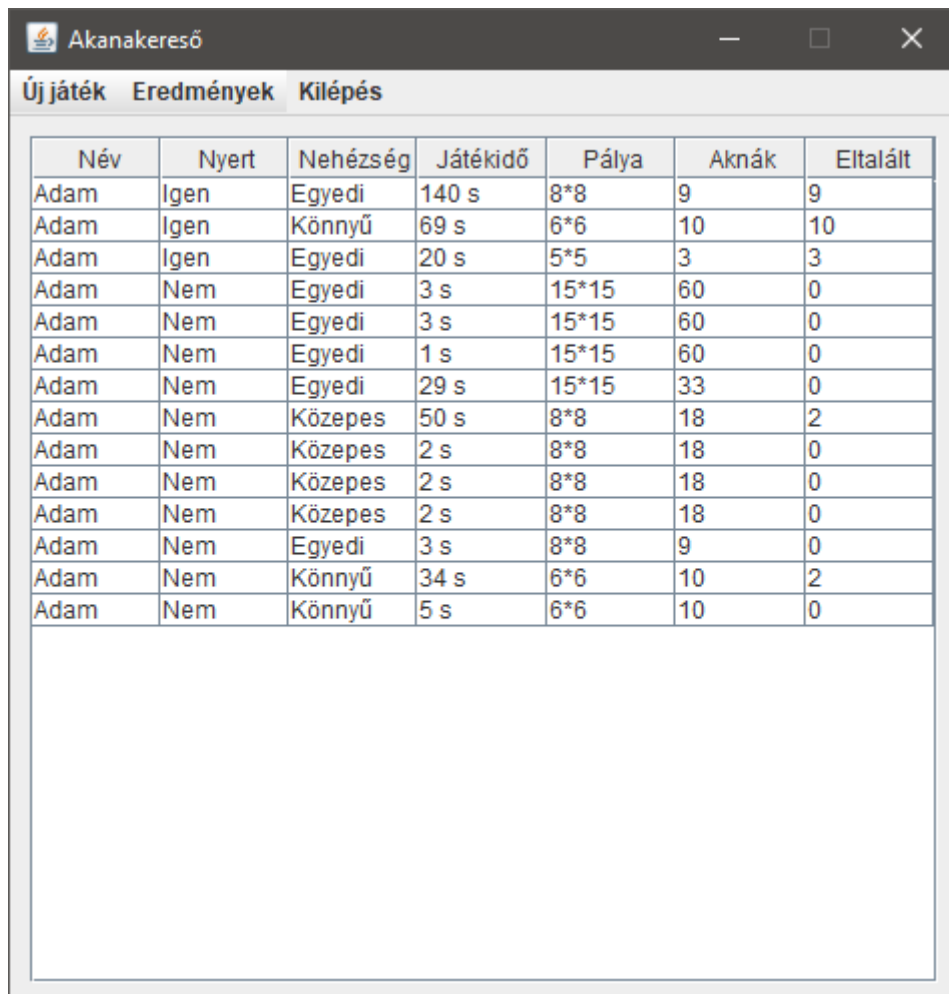
## 2.5 Eredménytábla megtekintése

A program futásának bármelyik pontján megtekinthetjük az eddig játszott játékok eredményét. Ehhez a menüben az

**Eredménytábla -> Eredménytábla megtekintése**  
pontra kell kattintanunk.

Előfordulhat, hogy még nincsenek eredmények elmentve, vagy a program valamiért nem képes megnyitni az azokat tároló fájlt, ekkor az eredménytábla helyett szöveges üzenet jelenik meg.

Ha a program talált korábbi eredményeket, akkor a következőhöz hasonló táblázatot láthatunk:



| Név  | Nyert | Nehézség | Játékidő | Pálya | Aknák | Eltalált |
|------|-------|----------|----------|-------|-------|----------|
| Adam | Igen  | Egyedi   | 140 s    | 8*8   | 9     | 9        |
| Adam | Igen  | Könnyű   | 69 s     | 6*6   | 10    | 10       |
| Adam | Igen  | Egyedi   | 20 s     | 5*5   | 3     | 3        |
| Adam | Nem   | Egyedi   | 3 s      | 15*15 | 60    | 0        |
| Adam | Nem   | Egyedi   | 3 s      | 15*15 | 60    | 0        |
| Adam | Nem   | Egyedi   | 1 s      | 15*15 | 60    | 0        |
| Adam | Nem   | Egyedi   | 29 s     | 15*15 | 33    | 0        |
| Adam | Nem   | Közepes  | 50 s     | 8*8   | 18    | 2        |
| Adam | Nem   | Közepes  | 2 s      | 8*8   | 18    | 0        |
| Adam | Nem   | Közepes  | 2 s      | 8*8   | 18    | 0        |
| Adam | Nem   | Közepes  | 2 s      | 8*8   | 18    | 0        |
| Adam | Nem   | Egyedi   | 3 s      | 8*8   | 9     | 0        |
| Adam | Nem   | Könnyű   | 34 s     | 6*6   | 10    | 2        |
| Adam | Nem   | Könnyű   | 5 s      | 6*6   | 10    | 0        |

Ebben a pontok csökkenő sorrendbe rendezve jelennek meg. Mivel a program nem számít egy konkrét számszerű pontértéket, ezért a megjelenített adatok hierarchikus rendezésével alakítja ki a sorrendet.

## 2.6 Kilépés a programból

A menüsáv „Kilépés” gombja vagy az ablak szabványos bezáró gombja a program futása során bármikor, azonnal kilép a programból.

## 3. Tervezési döntések

### 3.1 Adatszerkezetek

Az adatok tárolásához felhasználtam a Java-beli `Collections` segítségét.

**Az aknamező tárolása:** *field* attribútum a *Minefield* osztályban

Az aknamező tárolásához szükségem volt egy 2 dimenziós, integer-eket tároló adatstruktúrára. Erre a célra egy `List<List<Integer>>` típust használtam fel a laborvezetőm ajánlására. Az adatokat így könnyedén kezelni tudtam. Az adatszerkezetben tárolt értékek:

**Megjelenítendő mezők:** *displayField* attribútum a *GameInstance* osztályban

A megjelenítéshez kellett egy ugyanolyan struktúra, mint amiben az aknamezőt is tárolom. Ennek megfelelően itt is a `List<List<Integer>>` struktúrát használtam. Benne tárolt értékek:

| Tárolt érték | Jelentés - field  |
|--------------|-------------------|
| -1           | Akna              |
| 0            | Üres mező         |
| 1            | 1 szomszédon akna |
| 2            | 2 szomszédon akna |
| 3            | 3 szomszédon akna |
| 4            | 4 szomszédon akna |
| 5            | 5 szomszédon akna |
| 6            | 6 szomszédon akna |
| 7            | 7 szomszédon akna |
| 8            | 8 szomszédon akna |

| Tárolt érték | Jelentés - displayField |
|--------------|-------------------------|
| -1           | Akna                    |
| 0            | Üres mező               |
| 1            | 1 szomszédon akna       |
| 2            | 2 szomszédon akna       |
| 3            | 3 szomszédon akna       |
| 4            | 4 szomszédon akna       |
| 5            | 5 szomszédon akna       |
| 6            | 6 szomszédon akna       |
| 7            | 7 szomszédon akna       |
| 8            | 8 szomszédon akna       |
| 9            | Megjelölt mező          |
| 10           | Ismeretlen mező         |

**Gombok tárolása:** *board* attribútum a *GameInstance* osztályban

Mivel az tárolandó objektumok elrendezése követte az előző két adatszerkezet elrendezését, itt is kétszintű `List`-et használtam: `List<List<JButton>>`, a gombok koordináta alapján elérhetőek.

**Pontok tárolása egy játékmenetből:** *ScoreInstance* osztály

Ezt az osztályt dedikált módon arra hoztam létre, hogy egy adott játékmenet statisztikáját rögzítsék. Megvalósítja a `Serializable` interface-t, így könnyedén kiírható bármilyen adatfolyamba, konkrét esetünkben pedig egy erre a célra készülő fájlba. Kap egy játékmenet példányr, és abból kiolvassa az adatokat amik eltárolásra kerülnek benne.

**Pontlista:** *ScoresData* osztály

Ez az osztály az *AbstractTableModel* osztály leszármazottja. Benne az egyes játékok pontjai egy `ArrayList<ScoreInstance>` típusú, *results* nevű változóban kerülnek tárolásra.

## 3.2 A grafikus interfész felépítése, sajátosságai

A megoldás során több osztályomat is a Swing egyes komponenseiből származtattam, ezzel leegyszerűsítve a felület építését/átépítését:

**GameFrame** – a JFrame leszármazottja

Ez az osztály biztosítja a program ablakát, az összes többi komponens ebbe kerül bele.

### *JPanel-ből származtatott osztályaim:*

**StartScreen** – a játék kezdőképernyője (lásd feljebb)

**GameInstance** – egy adott játékmenetet, táblát megjelenítő és kezelő osztály

**StatusBar** – a tábla felett megjelenő sáv

**GameOverBar** – a játék végén statisztikát megjelenítő sáv

**Editor** – a pályaszerkesztő ablakot megtestesítő osztály

**ScoreBoard** – ez az osztály pedig az eredménytáblát jeleníti meg

## 3.3 Forrásfájlok és tartalmuk

A program minden osztálya az *aknakereso* package része.

### **Editor.java**

Editor osztály, azon belül EditorButtonListener és SliderListener osztályok

### **FieldListener.java**

FieldListener osztály

### **GameFrame.java**

GameFrame osztály, azon belül MenuListener osztály

### **GameInstance.java**

GameInstance osztály

### **GameOverBar.java**

GameOverBar osztály

### **Main.java**

Main osztály

### **Minefield.java**

Minefield osztály

### **ScoreBoard.java**

ScoreBoard osztály

### **ScoreComp.java**

ScoreComp osztály

### **ScoreInstance.java**

ScoreInstance osztály

### **ScoresData.java**

ScoreData osztály

### **StartScreen.java**

StartScreen osztály, azon belül StartScrButtonListener osztály

### **StatusBar.java**

StatusBar osztály, azon belül InstantRestartButtonListener, ToggleButtonListener és TimerListener

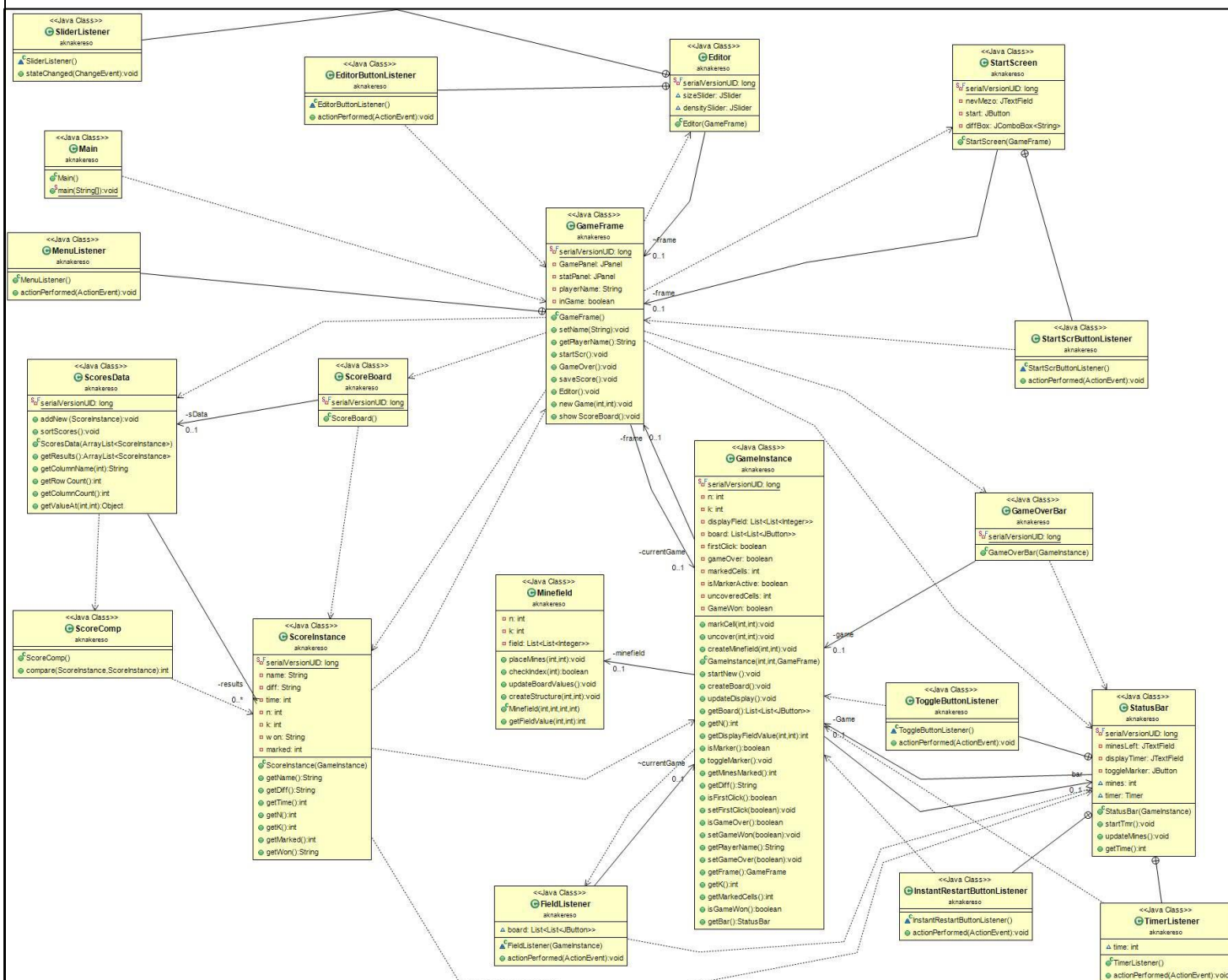


## 4. Osztályok

Azokat a metódusokat és egyéb részleteket, amik úgy éreztem, hogy nem egyértelműek, a forráskódban is megjegyzésekkel láttam el.

## Osztálydiagram

Itt látható a program összes osztálya, azok attribútumai és metódusai, a közöttük lévő kapcsolatok és függőségek ábrázolásával.



## 4.1 Minefield

### Felelősségek

Aknamező logikai szerkezetének és tartalmának generálása és tárolása.

### Attribútumok

*n : int*

A pálya méretét tárolja.

*k : int*

Az aknák számát tárolja.

*field : List<List<Integer>>*

Az aknamezőt tárolja.

### Metódusok

*placeMines(int, int): void*

Elhelyez k db aknát a field struktúrában. A paraméterként kapott koordinátára és szomszédjaira nem tehet le aknát.

*checkIndex(int): boolean*

Ellenőrzi, hogy a paraméterként kapott szám lehet-e koordináta része. (Legalább 0, kisebb min n)

*updateBoardValues(): void*

Végiglépked a field struktúrán, szomszédok alapján kiszámítja a mezők számértékét.

*createStructure(int, int): void*

Létrehozza a field struktúrát, (n,k) paraméterekkel.

*Minefield(int, int, int, int)*

Létrehozza a szerkezetet és elhelyezi a bombákat. Paraméterek: méret, aknák száma, koordináta (x,y).

*getFieldValue(int, int):int*

Visszadja a field struktúra paraméterként kapott koordinátáján tárolt elemét.

## 4.2 GameInstance

### JPanel leszármazottja

#### Felelősségek

Ez az osztály felelős egy adott játékmenet vezérléséért, ideértve a megjelenített pálya tárolását, a gombtábla tárolását, egy aknamező példány tárolását, a pálya grafikus megejelenítését, a pálya változásainak kezelését, és természetesen a játékmenet befejezését.

#### Attribútumok

***n : int***

Tárolja a pálya méretét.

***k : int***

Tárolja az aknák számát.

***minefield : Minefield***

Tárolja a játékmenethez tartozó aknamezőt.

***displayField : List<List<Integer>>***

Tárolja a megjelenített pálya értékeit.

***board : List<List<JButton>>***

Tárolja a tábla gombjait.

***firstClick : boolean***

Tárolja, hogy elindult-e már a játék.

***gameOver : boolean***

Tárolja, hogy véget ért-e már a játék.

***frame : GameFrame***

Tárolja a játékmenetet tartalmazó ablakot.

***markedCells : int***

Tárolja a megjelölt mezők számát.

***isMarkerActive : boolean***

Tárolja, hogy jelölő módban van-e a játék.

***bar : StatusBar***

Tárolja a játékmenethez tartozó státuszbart.

**uncoveredCells : int**

Tárolja a feloldott cellák számát.

**GameWon : boolean**

Tárolja, hogy a játék sikeres volt-e.

**Metódusok****markCell(int, int): void**

Megejelöli a kapott koordinátájú mezőt, ha már meg volt jelölve, akkor visszavonja.

**uncover(int, int): void**

Feloldja a kapott koordinátájú mezőt. Ha üres volt, rekurzívan meghívódik a szomszédokra is. Ha akna volt, feloldja az összes aknát, befejezettnak minősíti a játékot.

**createMinefield(int, int): void**

A kapott koordináták ismeretében létrehozhatja az aknamezőt.

**GameInstance(int, int, GameFrame)**

Konstruktor, inicializálja a változókat, felépíti a felületet. Paraméterei (n,k), valamint az anyaablak.

**startNew(): void**

Megkéri az anyaablakot, hogy indítson új játékot, az aktuális (n,k) paraméterekkel.

**createBoard(): void**

Létrehozza a gombokat a játéktáblán.

**updateDisplay(): void**

A displayField értékeinek megfelelő képeket jelenít meg a gombokon. (Ha nem sikerül, helyettesítő szöveget ír rá)

**getBoard(): List<List<JButton>>**

Visszaadja a gombtáblát.

**getN(): int**

Visszaadja n értékét.

**getDisplayFieldValue(int, int): int**

Visszaadja a displayField értékét a paraméterként kapott koordinátán.

**isMarker(): boolean**

Megmondja, jelölő módban van-e a játék.

**toggleMarker(): void**

Vált Felderítő/Jelölő mód között.

**getMinesMarked(): int**

Megmondja hány akna van megjelölve a játékpályán.

**getDiff(): String**

Megmondja milyen nehézségi szinten folyik a játék.

**isFirstClick(): boolean**

Megmondja elindult-e már a játék.

**setFirstClick(boolean): void**

Beállítja a firstClick változó értékét.

**isGameOver(): boolean**

Megmondja tart-e még a játék.

**setGameWon(boolean): void**

Beállítja a gameWon változó értékét.

**getPlayerName(): String**

Elkéri a játékos nevét az anyaablaktól és visszaszadja.

**setGameOver(boolean): void**

Beállítja a gameOver változó értékét.

**getFrame(): GameFrame**

Visszaszadja az anyaablakot.

**getK(): int**

Visszaszadja az akna számát.

**getMarkedCells(): int**

Megmondja, hány mező van megjelölve.

**isGameWon(): boolean**

Megmondja, sikeres volt-e a játék.

**getBar(): StatusBar**

Visszaszadja a játékhoz tartozó státuszbárt.

## 4.3 GameFrame

### JFrame leszármazottja

#### Felelősségek

A program összes grafikus komponensét tartalmazza, megjeleníti a különböző felületeket és a menüt. Ha egy játék véget ér, elmenti az eredményét. Tárolja az adott játékmenetet. Tárolja a játékos nevét.

#### Attribútumok

**currentGame : GameInstance**

Tárolja az aktuális játékmenetet.

**GamePanel : JPanel**

Ebbe a panelba kerülnek az aktuális játékmenetek.

**statPanel : JPanel**

Ebbe a panelba kerülnek a statisztikát mutató GameOverBar példányok.

**playerName : String**

Játékosnév eltárolása.

**inGame : boolean**

Játékmenet állapotának eltárolása.

#### Metódusok

**GameFrame()**

Konstruktor, inicializálja a változókat, felépíti az ablakot, megjeleníti a kezdőképernyőt.

**setName(String): void**

Beállítja a játékos nevét a paraméterként kapott értékre.

**getPlayerName(): String**

Visszaadja a játékos nevét.

**startScr(): void**

Eltávolítja az aktuális paneleket az ablakból, megjeleníti a kezdőképernyőt.

**GameOver(): void**

Ha aktív a játékmenet, elmenti az eredményt. Megjeleníti a JÁTÉK VÉGE panelt (statisztika). Inaktívnak minősíti az aktuális játékot.

**saveScore(): void**

Beolvassa az eredményfájlt, hozzáadja a struktúrához az újonnan létrehozott eredményt, és újraírja az eredményfájlt.

### **Editor(): void**

Eltávolítja az aktuális paneleket az ablakból, megjeleníti a pályaszerkesztőt.

### **newGame(int, int): void**

Eltávolítja az aktuális paneleket az ablakból, megjeleníti az újonnan elindított játékmenetet, amit a paraméterként kapott (n,k) párossal hoz létre.

### **showScoreBoard(): void**

Eltávolítja az aktuális paneleket az ablakból, megjeleníti a eredménytáblát, amit létrehoz.

## 4.4 StartScreen

### JPanel leszármazottja

#### Felelősségek

Megtestesíti a kezdőképernyőt.

#### Attribútumok

### **nevMezo : JTextField**

A mező, ahol a játékos megadhatja a nevét.

### **start : JButton**

A játékot elindító gomb.

### **diffBox : JComboBox<String>**

Legördülő menü, innen lehet nehézséget választani.

### **frame : GameFrame**

Tárolja a panelt tartalmazó ablakot.

#### Metódusok

### **StartScreen(GameFrame)**

A konstruktor regisztrálja a paraméterként kapott ablakot, és felépíti a kezdőképernyő paneljét.

## 4.5 Editor

### JPanel leszármazottja

#### Felelősségek

Megtestesíti a pályaszerkesztőt.

#### Attribútumok

##### *sizeSlider : JSlider*

Tárolja a csúszkát, amivel a pálya kívánt méretét lehet állítani.

##### *densitySlider : JSlider*

Tárolja a csúszkát, amivel az aknák kívánt arányát lehet megadni.

##### *frame : JFrame*

Tárolja a panelt tartalmazó ablakot.

#### Metódusok

##### *Editor(JFrame)*

A konstruktor regisztrálja a paraméterként kapott ablakot, és felépíti a pályaszerkesztő paneljét.



## 4.6 StatusBar

### JPanel leszármazottja

### Felelősségek

Megtestesíti a státuszsávot, számolja az időt és a fennmaradó aknák számát.

### Attribútumok

#### minesLeft : JTextField

Tárolja a szövegmezőt, ami a fennmaradó aknák számát jelzi.

#### displayTimer : JTextField

Tárolja a szövegmezőt, ami az időszámlálót valósítja meg.

#### toggleMarker : JButton

Tárolja az akcióváltó gombot.

#### Game : GameInstance

Tárolja az aktuális játékmenetet, amihez a státuszsáv tartozik.

#### mines : int

Tárolja a fennmaradó aknák számát.

#### timer : Timer

Tárolja a játékmenethez tartozó időzítőt.

### Metódusok

#### StatusBar(GameInstance)

A konstruktor regisztrálja a paraméterként kapott játékmenetet és felépíti a státuszsáv paneljét.

#### startTmr(): void

Elindítja az időzítőt.

#### updateMines(): void

Frissíti a fennmaradó aknák számát.

#### getTime(): int

Megmondja, hogy hány másodpercnél tart a számláló.

## 4.7 GameOverBar

**JPanel leszármazottja**

### Felelősségek

Megtestesíti a statisztikapanelt.

### Attribútumok

**game : GameInstance**

Tárolja a játékmenetet, amihez a panel tartozik.

### Metódusok

**GameOverBar(GameInstance)**

A paraméterként kapott játékmenet alapján kiszámolja a mezőinek értékét, majd felőíti a statisztikapanelt.

## 4.8 ScoreBoard

**JPanel leszármazottja**

### Felelősségek

Megtestesíti az eredménytáblát. Beolvassa az eredményfájlt és megjeleníti a tartalmát.

### Attribútumok

**sData : ScoresData**

Eredmények halmazát tárolja.

### Metódusok

**ScoreBoard()**

A konstruktor eredményfájlt keres és próbál meg beolvasni, ha sikerrel jár, panelt épít az eredménytáblázattal. Ha nem jár sikerrel, a panelen hibaüzenet lesz látható.

## 4.9 FieldListener

implementálja az **ActionListener** interface-t

### Felelősségek

Kezeli a játéktáblán történő történő kattintásokat.

### Attribútumok

**currentGame : GameInstance**

Tárolja a játékmenetet, amihez tartozik.

**board : List<List<JButton>>**

Tárolja a gombtáblát, amit kezelnie kell.

### Metódusok

**FieldListener(GameInstance)**

Regisztrálja a kapott játékmenetet. Elkéri a játékmenettől a gombtáblát, eltárolja, végigmegy rajta, minden gombnak beállítja az eseménykezelőjét saját magára.

**actionPerformed(ActionEvent)**

Kezeli a játék eseményeit. Egy gombra kattintáskor meghatározza annak a koordinátáit, aztán attól függően, hogy a játék Jelölő/Felderítő módban van-e, megjelöli/feloldja az adott gombhoz tartozó mezőt. Kivételek, ha a játék meg nem indult el, ekkor elindítja azt, legeneráltatja az aknamezőt, valamint ha a játék véget ért, ilyenkor figyelmen kívül hagyja a kattintásokat.

## 4.10 ScoreInstance

implementálja a **Serializable** interface-t

### Felelősségek

Tárolja az egy adott játékmenethez tartozó eredményeket.

### Attribútumok

**name : String**

Tárolja a játékos nevét.

**diff : String**

Tárolja a nehézséget.

**time : int**

Tárolja a játékidőt.

**n : int**

Tárolja a pályaméretet.

**k : int**

Tárolja az aknák számát.

**won : String**

Tárolja szövegesen, hogy nyert-e a játékos.

**marked : int**

Tárolja, hogy hány aknát jelölt meg sikeresen a játékos.

### Metódusok

**ScoreInstance(GameInstance)**

A kapott játékmenetből lekérdezi az adatokat amiket eltárol.

**getName(): String**

Visszadaja a játékosnevet.

**getDiff(): String**

Visszaadja a nehézséget.

**getTime(): int**

Visszaadja a játékidőt.

**getN(): int**

Visszaadja a pályaméretet.

**getK(): int**

Visszaadja az aknák számát.

**getMarked(): int**

Visszaadja az eltalált jelölések számát.

**getWon(): int**

Visszaadj szövegesen, hogy nyert-e a játékos.

## 4.11 ScoresData

**AbstractTableModel** leszármazottja

### Felelősségek

Eredmények egy halmazát tárolja és kezeli, képes a sorbarendezésre.

### Attribútumok

**results : ArrayList<ScoreInstance>**

Tárolja az eredmények listáját.

### Metódusok

**addNew(ScoreInstance): void**

Új eredményt ad a listához.

**sortScores(): void**

Sorbarendezi a listát. (lásd ScoreComp osztály)

**ScoresData(ArrayList<ScoreInstance>)**

Egyszerű eredménylistából csinál egy ScoresData példányt. (táblázatmodell)

**getResults(): ArrayList<ScoreInstance>**

Visszaadja a tárolt eredménylistát.

**getColumnName(int): String**

Visszaadja az egyes oszlopok nevét. (táblázatmodell)

**getRowCount(): int**

Visszaadja a lista hosszát.

**getColumnCount(): int**

Visszaadja a tárolt tulajdonságok számát.

**getValueAt(int, int): Object**

Visszaad egy táblázatértéket, a koordinátával jelölt helyen. Az értékek a megjelenítéshez vannak formázva.

## 4.12 ScoreComp

### implementálja a Comparator interface-t

(ScoreInstance objektumként paraméterrel)

#### Felelősségek

Összehasonlít két eredmény objektumot, eldönti melyik a jobb.

Összehasonlítási hierarchia:

- nyert-e a játékos
- nagyobb-e a pálya
- több akna volt-e
- több aknát jelölt-e meg
- gyorsabban végzett-e

#### Metódusok

*compare(ScoreInstance, ScoreInstance): int*

A fenti prioritás alapján visszaadja a jobbik eredményt a kapott 2 közül.

## 4.13 MenuListener

### implementálja az ActionListener interface-t

A GameFrame osztályon belül található.

#### Felelősségek

A menügombok megnyomására végrehajtja a megfelelő akciót.

#### Metódusok

*actionPerformed(ActionEvent): void*

Végrehajtja a kiválasztott menüelemnek megfelelő akciót.

## 4.14 StartScrButtonListener

**implementálja az ActionListener interface-t**

A StartScreen osztályon belül található.

### Felelősségek

Gombnyomásra elindítja a játékot a kiolvasott paraméterekkel.

### Metódusok

***actionPerformed(ActionEvent): void***

Kiolvassa a paramétereket, elindítja a játékot.

## 4.15 EditorButtonListener

**implementálja az ActionListener interface-t**

Az Editor osztályon belül található.

### Felelősségek

Gombnyomásra elindítja a játékot a Sliderekől olvasott paraméterekkel.

### Metódusok

***actionPerformed(ActionEvent): void***

Leolvassa a paramétereket, elindítja a játékot.

## 4.16 SliderListener

**implementálja a ChangeListener interface-t**

Az Editor osztályon belül található.

### Felelősségek

A méretet állító csúszka változásakor annak értékéhez képest megváltoztatja az aknaszázalékot állító csúszka értékének a felső korlátját.

### Metódusok

***stateChanged(ChangeEvent): void***

Kiszámítja az új felső korlátot a csúszka értékének, és beállítja azt.



## 4.17 InstantRestartButtonListener

**implementálja az ActionListener interface-t**

A StatusBar osztályon belül található.

### Felelősségek

Az aktuális (n,k) paraméterekkel indít új játékot.

### Metódusok

***actionPerformed(ActionEvent): void***

Megkéri a játékmenetet, hogy kérje meg az ablakot, hogy indítson új játékot.

## 4.18 ToggleButtonListener

**implementálja az ActionListener interface-t**

A StatusBar osztályon belül található.

### Felelősségek

Gombnyomásra vált Felderítő és Jelölő mód között.

### Metódusok

***actionPerformed(ActionEvent): void***

Ha aktív a játékmenet, vált a két akció között, megváltoztatja a gombon megjelenő képet.

## 4.19 TimerListener

### implementálja az ActionListener interface-t

A StatusBar osztályon belül található.

#### Felelősségek

Növeli az időt jelző mező értékét, ha lejárt az időzítő.

#### Attribútumok

*time : int*

Tárolja a kezdés óta eltelt másodperceket.

#### Metódusok

*TimerListener()*

Nullára állítja a számlálót.

*actionPerformed(ActionEvent): void*

Eggyel növeli a számlálót.

## 4.20 Main

#### Felelősségek

A program indításakor létrehoz egy ablakot.

#### Metódusok

*main(String[]): void*

Létrehoz egy új ablakot.

## 5. Use-casek, interakciók

A use-case-eket olyan formában szeretném ismertetni, mint a felhasználó lehetséges interakciói a programmal.

Lehetséges interakciók:

### **Váltás a program különböző felületei között:**

- kezdőképernyő megjelenítése
- pályaszerkesztő megjelenítése
- Eredménytábla megjelenítése
- Játék indítása

### **Játék indítása / játék:**

- Nehézség választása vagy Méret és sűrűség választása
- Mezők felfedése
- Mezők megjelölése
- Új játék indítása

### **Kilépés a programból**

## 6. Fájlkezelés

### **scoreBoard.data** – Az elért eredmények gyűjteménye



A program egyedül az eredményeket képes fájlba menteni, illetve onnan beolvasni. Ezek a műveletek a felhasználó beavatkozása nélkül, automatikusan történnek. A program olyankor írja ki az adatokat fájlba, ha egy játék véget ért. Ilyenkor először megpróbál beolvasni egy már létező eredményfájlt aminek a tartalmához hozzáfűzheti az új eredményt, ha ez nem sikerül neki akkor újat hoz létre. Beolvasásra az imént említett eseten kívül olyankor kerül sor, ha a felhasználó megnyitja az eredménytáblát a programban.


A kezelt fájl a ScoreInstance (Serializable) osztály példányaiból álló listát tartalmaz. Az adatszerkesztés nem támogatott.





















## 7. Tesztesetek leírása

A programhoz 10 tesztesetet készítettem, ezek ellenőrzik az olyan metódusok működését, amik bonyolultabb műveleteket végeznek. A tesztelés során minden eset hibátlanul futott, az elvárt eredményeket hozta:

Finished after 1.445 seconds

Runs: 10/10    Errors: 0    Failures: 0



- ✓  aknakereso.MarkerTest [Runner: JUnit 4] (1.093 s)
  - ✓  testMarking (1.093 s)
- ✓  aknakereso.MinesMarkedTest [Runner: JUnit 4] (0.033 s)
  - ✓  minesMarkedTest (0.033 s)
- ✓  aknakereso.CheckIndexTest [Runner: JUnit 4] (0.000 s)
  - ✓  testIndexRestrains (0.000 s)
- ✓  aknakereso.TestScoreSorting [Runner: JUnit 4] (0.072 s)
  - ✓  testSortment (0.072 s)
- ✓  aknakereso.UncoverTest [Runner: JUnit 4] (0.028 s)
  - ✓  uncoverTest (0.028 s)
- ✓  aknakereso.TestButtonBoardDimensions [Runner: JUnit 4] (0.034 s)
  - ✓  testDimensions (0.034 s)
- ✓  aknakereso.PlaceMinesTest [Runner: JUnit 4] (0.001 s)
  - ✓  testPlacement (0.001 s)
- ✓  aknakereso.UpdateBoardValuesTest [Runner: JUnit 4] (0.000 s)
  - ✓  testCellValues (0.000 s)
- ✓  aknakereso.MinesLeftStatusBarTest [Runner: JUnit 4] (0.022 s)
  - ✓  minesLeftTest (0.022 s)
- ✓  aknakereso.ToggleMarkerTest [Runner: JUnit 4] (0.016 s)
  - ✓  toggleTest (0.016 s)

Ezek a tesztek azon alapulnak, hogy olyan pályát hozok bennük létre, aminek ismerem az elemeit, azokat ellenőrzöm le. Teljes mértékben függetlenek és determinisztikus a futásuk. A program jellegéből adódóan nehéz olyan tesztet írni, ami dinamikus olyan szempontból, hogy tetszőleges adatokkal működik, ezért is írtam meg ezeket a tesztek előre kiszámolt adatokkal.

## Beadott tesztállományok listája

A következő 10 tesztosztályt adtam be, ezek közül mind az *aknakereso* package része.

**-CheckIndexTest.java**

**-MarkerTest.java**

**-MinesLeftStatusBarTest.java**

**-MinesMarkedTest.java**

**-PlaceMinesTest.java**

**-TestButtonBoardDimensions.java**

**-TestScoreSorting.java**

**-ToggleMarkerTest.java**

**-UncoverTest.java**

**-UpdateBoardValuesTest.java**

## 7.1 CheckIndexTest

### Tesztelt funkció:

Minefield osztály CheckIndex metódusát teszteli, létrehoz két Minefield példányt, majd leellenőriztet különböző indexeket a függvénnyel, ami megmondja, hogy az adott aurnamezón érvényes-e az adott index.

## 7.2 MarkerTest

### Tesztelt funkció:

GameInstance osztály MarkCell metódusát hivatott tesztelni, létrehoz egy ablakot és egy játékot, majd jelöléssel és annak visszavonásával próbálkozik a pályán.

## 7.3 MinesLeftStatusBarTest

### Tesztelt funkció:

A StatusBar osztály UpdateMines metódusát teszteli. Létrehoz egy ablakot, egy játékot, azon belül egy aurnamezót. Elkezd jelölgetni rajta, közben ellenőrzi a fennmaradó aknák számát reprezentáló mezőt.

## 7.4 MinesMarkedTest

### Tesztelt funkció:

A GameInstance osztály getMinesMarked metódusát teszteli. Létrehoz egy ablakot és egy játékot, majd megjelöl néhány mezőt, ezután ellenőrzi a függvény által visszaadott számot.

## 7.5 PlaceMinesTest

### Tesztelt funkció:

A Minefield osztály PlaceMines metódusára koncentrálni. Létrehoz egy aurnamezót, elhelyeztet a függvénnyel az aknákat, aztán leellenőrzi a mezők értékét.

## 7.6 TestButtonBoardDimensions

### Tesztelt funkció:

A GameInstance osztály createBoard függvényét (a konstruktorban fut le) ellenőrzi. Létrehoz egy ablakot és egy játékot, majd elkéri a játék gombtábláját és ellenőrzi annak méretét.

## 7.7 TestScoreSorting

### Tesztelt funkció:

A ScoreComp komparátor működését ellenőrzi. Létrehoz egy ablakot, ahhoz 4 játékot, ezekből pontokat számol (el nem kezdett játékokra is értelmezhető az összehasonlítás, méret stb. alapján), amiket hozzáad egy listához. Ezt rendezi, és ellenőrzi a sorrendet.

## 7.8 ToggleMarkerTest

### Tesztelt funkció:

A GameInstance osztály toggleMarker metódusát ellenőrzi, létrehoz egy ablakot és egy játékot, majd többször vált az akciók között.

## 7.9 UncoverTest

### Tesztelt funkció:

A GameInstance osztály uncover metódusának validálása képpen létrehoz egy ablakot, egy játékot és egy aknamezőt, majd feloldja a pálya középső mezőjét, ami üres, mivel az volt a kezdőmező a pálya létrehozásakor (ez rekurzívan feloldja a szomszédokat). Ezután összeveti a kapott értékeket az elvárt értékekkel.

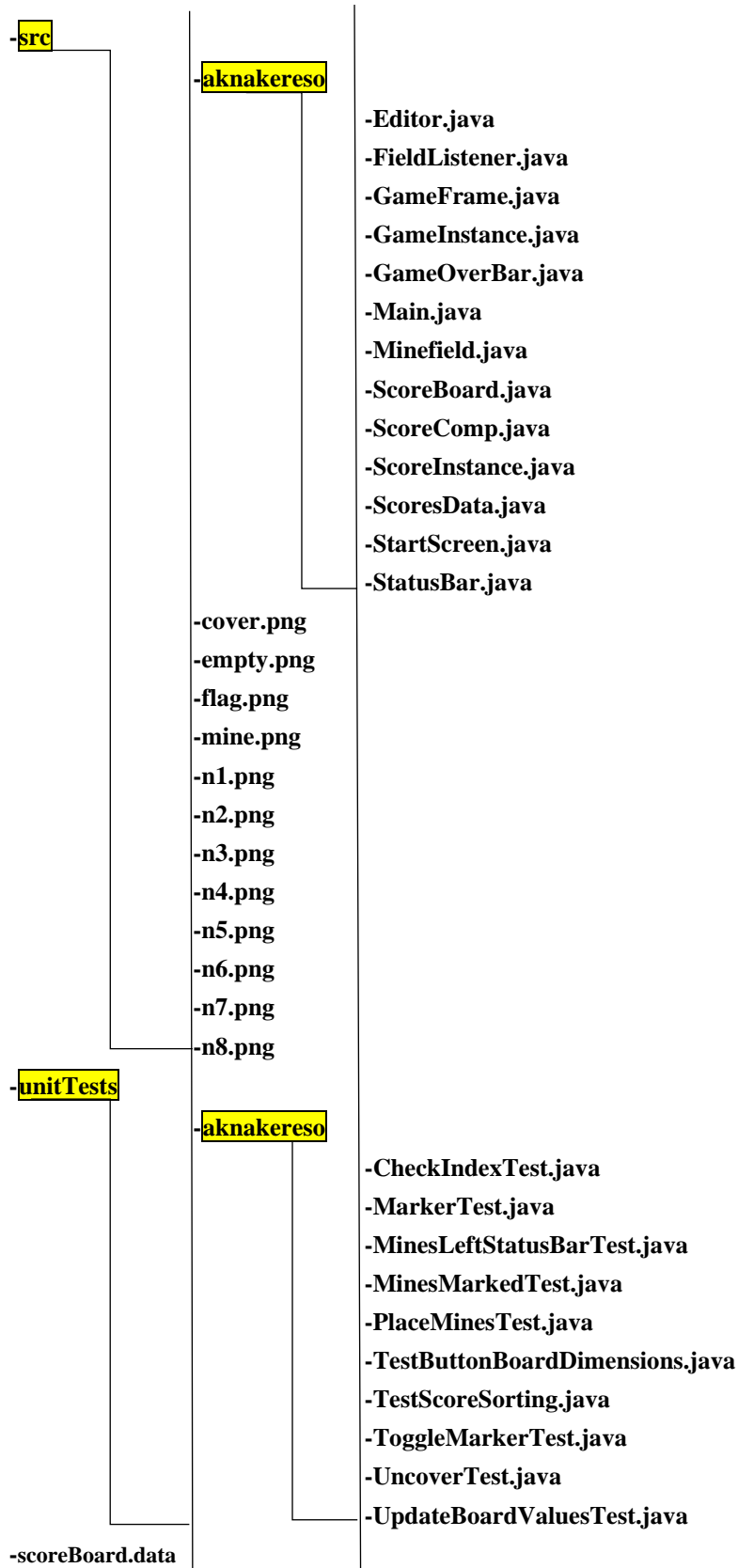
## 7.10 UpdateBoardValuesTest

### Tesztelt funkció:

A Minefield osztály updateBoardValues metódusát tesztel. Létrehoz egy aknamezőt, elhelyezi az aknákat, majd kiszámoltatja az üres mezők értékét, amit ezután leellenőriz.

## 8. A futáskor elvárt könyvtárszerkezet

A forrásfájlokat két mappába szerveztem. A program által megjelenített, általam készített PNG képek az src könyvtárban kell, hogy legyenek. Az eredményfájl helyét is jeleztem. A struktúra a következő:





## 9. Munkanapló

| Dátum      | Munkaórák | Elvégzett munka   |
|------------|-----------|---|
| 2020.11.23 | 7 óra     | „Projektterv”, Minefield osztály, GameInstance osztály, GameFrame osztály |
| 2020.11.24 | 6 óra     | FieldListener osztály, StartScreen osztály, javítások és bővítések        |
| 2020.11.25 | 7 óra     | Editor osztály, StatusBar osztály, sok javítás és bővítés                 |
| 2020.11.27 | 9 óra     | GameOverBar, eredménykezelés, javítások                                   |
| 2020.12.01 | 6 óra     | Dokumentáció elkezdése, ismertető, használati útmutató                    |
| 2020.12.02 | 5 óra     | Dokumentáció bővítése, osztályleírások                                    |
| 2020.12.04 | 9 óra     | Sok finomítás a kódban, rendezgetés, dokumentáció befejezése              |

**A feladattal töltött idő:** durván 50 óra

**Felhasznált eszközök:**

Eclipse

MS Word

ObjectAid (osztálydiagram)

Adobe Photoshop (PNG képek)

## 10. Saját gondolatok a feladat elvégzése után

A feladattal úgy érzem, hogy lényegesen több időt töltöttem, mint amennyi szükséges lett volna rá. A végeredménnyel működésében teljesen, képességeivel nagyrészt elégedett vagyok. Az utolsó időszakban sok pontatlanságot javítottam a kódban, kivettem felesleges részeket, viszont elképzelhető, hogy még így is maradt bent olyan rész, amit egy kis átgondolás után ki lehetne hagyni a programból. A játék jól működik, képes voltam belemélyülni néha, ami elnehezítette a munkával való haladást. Sok osztályt hoztam létre, de úgy gondolom mindegyiknek megvan a létjogosultsága, indokolt volt kitalálni őket. Sok olyan technikát használtam fel, amiket a laborgyakorlatok során tanultam. Összességében élveztem a feladat elkészítését.

Tóth Ádám László

2020. 12. 04.