

**A programozás alapjai 3.**

**Házi feladat dokumentáció**

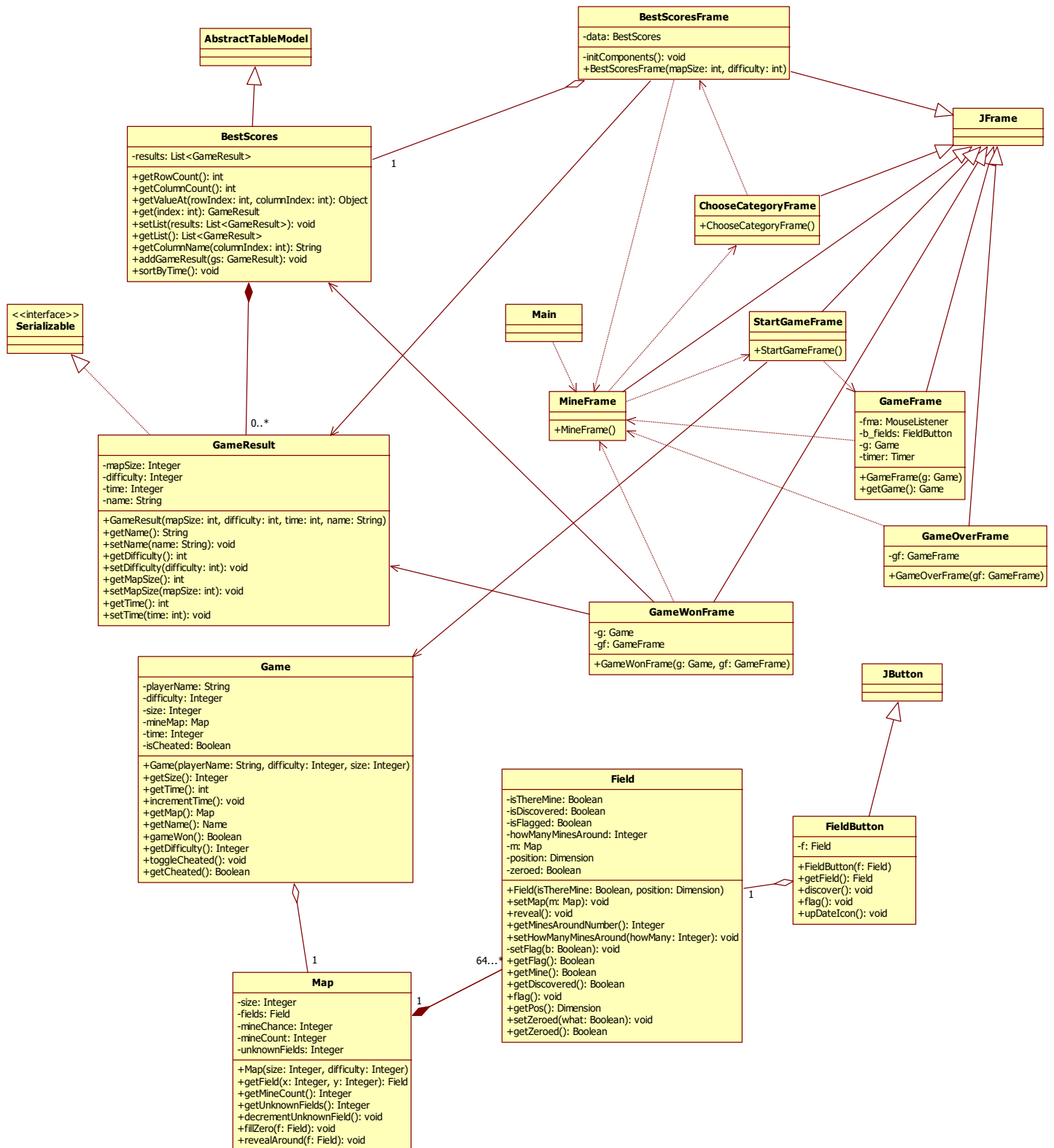
**Kálmán Bendegúz Bence**

**PTW6BD**

**Aknakereső**

# 1. Osztálydiagram

A diagramon a JFrame-ken elhelyezkedő elemeket és belső (listener) osztályokat nem jelöltem külön, mert áttekinthetetlen lett volna a diagram.



## 2. Osztályok, metódusok megvalósítása

A program írásakor törekedtem arra, hogy a játék logikáját ne a grafikus felületért és felhasználói interakciókért felelős osztályok valósítsák meg. Külön package-be szerveztem a felületért felelős, a játék logikájáért felelős, és a serializálásért felelős részeket.

### 2.1. A játék logikája

Ezek az osztályok egy teljesen funkcionális akna kereső-játékot valósítanak meg, csak éppen a felhasználóval való kommunikációt nem valósítják meg.

#### 2.1.1. Map osztály

Ez az osztály egy aknamezőt reprezentál. Létrehoz egy aknamezőt, aminek aztán nyilvántartja a mezőit, amiken különböző műveleteket végezhetünk.

##### Tagváltozók

`private Integer size` : az aknamező méretét adja meg.  
`private Field[][] fields` : a mezők tömbje.  
`private Integer mineChance` : egy mezőn mekkora valószínűséggel van akna (0-100).  
`private Integer mineCount` : összesen hány akna van a pályán.  
`private Integer unknownFields` : még hány felderítetlen mező van a pályán.

##### Metódusok

`public Map(Integer size, Integer difficulty)` : konstruktor, a megadott méret és nehézség alapján létrehoz egy pályát, amin véletlenszerűen helyezkednek el az aknák.  
`public Field getField(Integer x, Integer y)` : visszaad egy mezőt a megadott pozícióról.  
`public Integer getMineCount()` : visszaadja, hogy hány akna van az aknamezőn.  
`public Integer getUnknownFields()` : visszaadja, hogy hány felderítetlen mező maradt.  
`public Integer getMineCount()` : visszaadja, hogy hány akna van az aknamezőn.  
`public void decrementUnknownFields()` : csökkenti a felderítetlen mezők számát.  
  
`public void fillZero(Field f)` : rekurzívan felderíti az összes f-fel egybefüggő olyan mezőt, ami körül nincs egy akna sem. A pálya széleit és sarkait feltételekkel ellenőrzi.  
  
`public void revealAround(Field f)` : felderíti f körül az összes mezőt (f-et nem). A pálya széleit és sarkait feltételekkel ellenőrzi.

## 2.1.2. Field osztály

Egy mezőt reprezentál.

### Tagváltozók

`private Boolean isThereMine` : megmondja, hogy van-e a mezőn akna.  
`private Boolean isDiscovered` : megmondja, hogy a mezőt felfedeztük-e már.  
`private Boolean isFlagged` : megmondja, hogy a mezőn van-e zászló.  
`private Integer howManyMinesAround` : megmondja, hogy hány akna van a mező körül.  
`private Dimension position` : a mező koordinátái a pályán. (pl. bal felső sarok: (0;0)).  
`private Boolean zeroed` : a rekurzív felderítő függvénynek ezzel tudjuk jelezni, hogy meg lett-e már hívva a mezőre, így nem kerülünk végtelen ciklusba.

### Metódusok

`public Field(Boolean isThereMine, Dimension position)` : konstruktor, létrehoz egy mezőt, amihez argumentumban adjuk meg a pozícióját, és hogy tartozik-e hozzá akna.  
  
`public void setMap(Map m)` : beállítjuk a mezőhöz tartozó Map-et.  
  
`public void reveal()` : felfedi a mezőt.  
  
`public Integer getMinesAroundNumber()` : visszaadja a mező körülöttei aknák számát.  
  
`public void setHowManyMinesAround(Integer howMany)` : ezzel állíthatjuk be a mező körül lévő aknák számát.  
  
`private void setFlag(Boolean b)` : ezzel tehetünk zászlót egy mezőre, de csak akkor, ha a mező nincs felfedezve.  
  
`public Boolean getFlag()` : visszaadja, hogy a mezőn van-e zászló.  
  
`public Boolean getMine()` : visszaadja, hogy a mezőn van-e akna.  
  
`public Boolean getDiscovered()` : visszaadja, hogy a mező fel lett-e már fedezve.  
  
`public void flag()` : zászlót tehetünk egy mezőre vagy vehetünk el onnan, de nem foglalkozunk az objektum belső állapotával, azaz hogy fel van-e derítve, és/vagy van-e már ott zászló.  
  
`public Dimension getPos()` : visszaadja a mező koordinátáit.  
  
`public void setZeroed(Boolean what)` : beállítja, hogy az adott mezőn meghívtuk-e már a rekurzív felderítést.  
  
`public Boolean getZeroed()` : visszaadja, hogy a mezőn meg lett-e már hívva a rekurzív felderítés.

### 2.1.3. Game osztály

Ez az osztály foglalkozik, kezel egy játékot.

#### Tagváltozók

`private String playerName` : a játékos neve.  
`private Integer difficulty` : a játék nehézsége (1-5).  
`private Integer size` : a játék aknamezőjének mérete.  
`private Map mineMap` : a játék aknamezője.  
`private Integer time` : amennyi ideje tart a játék.  
`private Boolean isCheated` : a játék éppen „csaló” módban van-e.

#### Metódusok

`public Game (String playerName, Integer difficulty, Integer size)` : konstruktor, a megadott név, nehézség és méret alapján létrehoz egy új játékot.

`public Integer getSize()` : visszaadja a játék pályájának méretét.

`public int getTime()` : visszatér, hogy hány másodperc óta tart a játék.

`public void incrementTime()` : a játék idejét növeli egy másodperccel.

`public Map getMap()` : visszaadja a játék pályájának méretét.

`public String getName()` : visszaadja a játékos nevét.

`public Boolean gameWon()` : igaz, ha megnyertük a játékot.

`public Integer getDifficulty()` : visszaadja a játék nehézségét.

`public void toggleCheated()` : átkapcsolja az ellentettjére a játék „csaló” állapotát (ha igaz volt hamis lesz, és fordítva).

`public Boolean getCheated()` : visszaadja, hogy a játék „csaló” állapotban van-e.

## 2.3. A grafikai felületért felelős osztályok

Ezek az osztályok felelnek a grafikai felületért. A nevében „Button” végződésű a JButton-ból származik le, a „Frame” végződésűek a JFrame-ből.

### 2.3.1. MineFrame

Ez az osztály valósítja meg a menü frame-jét.

#### Metódusok

`public MineFrame()` : konstruktor, felépíti az ablakot.

#### Belső osztályok

`private class ExitButtonActionListener implements ActionListener` : a kilépő gombot kezelő listener osztály.

`private class StartButtonActionListener implements ActionListener` : a Start gombot kezelő listener osztály

`private class ScoresButtonActionListener implements ActionListener` : a „Best Scores” gombot kezelő listener osztály.

### 2.3.2. StartGameFrame

Amikor új játékot szeretnénk kezdeni, ez a frame nyílik meg. Itt beírhatjuk a nevünket, kiválaszthatjuk a pálya méretét és a játék nehézségét, majd elindíthatjuk a játékot.

#### Metódusok

`public StartGameFrame()` : konstruktor, felépíti az ablakot.

#### Belső osztályok

`private class SubmitButtonActionListener implements ActionListener` : a Submit gombot kezelő osztály. Ha rákattintunk a gombra, elindul a játék.

### 2.3.3. GameFrame

A játék frame-je. Az összes FieldButton-t kirajzolja, és mindegyikhez MouseListener-t tart nyilván. Méri az időt, amit az ablakon megjelenít. Az ablak felső részén egy meü található, ahol gombbal feladhatjuk a játékot, egy másikkal pedig csalhatunk. Ha nyerünk vagy veszítünk, azt is lekezele.

#### Metódusok

`public GameFrame(Game g) : konstruktor, a paraméterben megadott játék alapján felépíti az ablakot, rárakja a gombokat, és elindít egy időzítőt.`

`public Game getGame() : visszaadja a frame-hez tartozó játékot.`

#### Belső osztályok

`private class GiveUpButtonListener implements ActionListener :` a „Give Up” gombhoz tartozó listener. Ha megnyomjuk a gombot, feladjuk a játékot, visszakerülünk a menübe.

`private class FieldMouseAction implements MouseListener :` egy mező gombját kezelő osztály. Annyi példány jön létre belőle, ahány mező van. Attól függően, hogy jobb vagy bal gombbal kattintunk a mezőre, mást csinál: bal gombra felderíti a mezőt, jobb gombra zászlót tesz rá. Minden mezőfelderítés után leellenőrzi, hogy megnyertük, vagy elvesztettük-e a játékot. Ha a mező körül, amire kattintottunk, nincs akna, akkor felderíti az egész, a mezővel egybefüggő ilyen mezőkből álló területet.

`private class TimerListener implements ActionListener :` Ez az osztály a GameFrame konstruktorában elindított Timer-t figyeli, és mindig amikor az aktiválódik (másodpercenként), a játék idejét inkrementálja.

`private class CheatButtonListener implements ActionListener :` A „Cheat!” (=csalás) gombhoz tartozó listener. Ha megnyomjuk, megmutatja, hogy a pályán hol vannak aknák. Ha ismét megnyomjuk, kikapcsolja a csalást, és eltűnnek az aknák.

### 2.3.4. GameWonFrame

Ha megnyerjük a játékot (tehát minden aknamentes mezőt felderítettünk), ez az ablak jelenik meg. Ez felel az eredménynek a ranglistára írásáért.

#### Metódusok

`public GameWonFrame(Game g, GameFrame gf) :` konstruktor, felépíti az ablakot, rárakja az elemeket. Átadjuk neki a játékot és a játékhoz tartozó frame-t, hogy tudjon menteni, és be tudja zárni a játékot.

#### Belső osztályok

`private class OkButtonActionListener implements ActionListener :` Az „Ok” gombot lekezelő listener osztály. Ha rákattintunk, akkor a program beolvassa

a háttértárról a ranglistát a „bestscores.dat” fájlból, hozzáadja az eredményünket, majd visszairja a listát a háttértárra. Végül bezárja a játék ablakát, és megnyitja a menüt.

### 2.3.5. GameOverFrame

Egyszerű frame osztály, amely közli a játékoskal, hogy elvesztette a játékot. Az „Ok” gomb hatására visszakerülünk a menübe.

#### Metódusok

`public GameOverFrame(GameFrame gf)` : konstruktor, ami megkapja a GameFrame-t is, hogy be tudja zárni.

#### Belső osztályok

`private class OkButtonActionListener implements ActionListener`: Az „Ok” gombhoz tartozó listener. Ha rákattintunk a gombra, bezáródnak az ablakok, és megnyílik a menü.

### 2.3.6. FieldButton

Ez az osztály egy-egy mezőhöz tartozó gombot reprezentál a grafikus felületen. Minden példánya nyilvántartja a játék egy-egy mezőjét. Ha a játékos interakcióba lép ezzel, tehát a mezőt reprezentáló gombbal, akkor az adott interakciót az osztály végrehajtja a hozzá tartozó mezőn is.

#### Metódusok

`public FieldButton(Field f)` : konstruktor, felépít egy gombot.

`public Field getField()` : visszaadja a gombhoz tartozó mezőt.

`public void discover()` : felderíti a gombhoz tartozó mezőt.

`public void flag()` : megjelöli a gombhoz tartozó mezőt zászlóval.

`public void upDateIcon()` : a gombhoz tartozó mező állapota alapján frissíti a gomb ikonját.

### 2.3.7. ChosseCategoryFrame

Mielőtt megnyitnánk a ranglistát, kiválasztjuk, hogy melyik pályaméterhez és melyik nehézséghez akarjuk megnézni az eredményeket. Ezek kiválasztását végzi ez az ablak.

#### Metódusok

`public ChooseCategoryFrame()` : konstruktor, ami felépíti az ablakot.

#### Belső osztályok



```
private class SubmitButtonActionListener implements  
ActionListener : A „Submit” gombot kezelő listener osztály. Bezárja az ablakot,  
majd megnyitja a rangistát.
```

### 2.3.8. BestScoresFrame

A ranglista frame-jét megvalósító osztály. Egy táblázatot (JTable) jelenít meg.

#### Metódusok

```
public BestScoresFrame(Integer mapSize, Integer difficulty) :  
létrehoz egy ablakot, ami egy táblázatban kilistázza a paraméterként megadott Map  
mérethez és nehézséghez sorrendben a legjobb eredményeket. Ezt úgy teszi, hogy  
beolvassa a „bestscores.dat” fájlból az összes eredményt egy BestScores objektumba,  
majd ebből kiválogatja a követelménynek megfelelőeket, rendezi idő szerint, és így  
adja át az ablak BestScores adattagjának, ami alapján felépül az ablak táblázata.
```

```
private void initComponents() : beállítja az ablak komponenseit, a  
konstruktor hívja meg.
```

#### Belső osztályok

```
private class MenuButtonActionListener implements  
ActionListener : A táblázat alatt van egy „Back to menu” gomb, az ehhez tartozó  
listener ez az osztály. Ha a gombra kattintunk, visszakerülünk a menübe.
```

## 2.4. A szerializálásért felelős osztályok

Ezek az osztályok végzik a ranglista szerializálását.

### 2.4.1. GameResult

Egy sikeres játék eredményének adatait tároló egyszerű osztály. Delegálja a Serializable interfészt.

#### Tagváltozók

private Integer mapSize : A játékhoz tartozó aknamező mérete.  
private Integer difficulty : A játék nehézsége.  
private Integer time : A játék teljesítésének ideje.  
private String name : A játékos neve.

#### Metódusok

public GameResult(Integer mapSize, Integer difficulty, Integer time, String name) : konstruktor, a megadott paraméterek alapján létrehoz egy objektumot.

public String getName() : visszaadja a játékos nevét.

public void setName(String name) : beállítja a játékos nevét.

public Integer getDifficulty() : visszaadja a játék nehézségét.

public void setDifficulty(Integer difficulty) : beállítja a játék nehézségét.

public Integer getMapSize() : visszaadja a játékhoz tartozó aknamező méretét.

public void setMapSize(Integer mapSize) : beállítja a játékhoz tartozó aknamező méretét.

public Integer getTime() : visszaadja a játékhoz tartozó időt.

public void setTime(Integer time) : beállítja a játékhoz tartozó időt.

## 2.4.2. BestScores

A ranglistát tároló és kezelő osztály, az AbstractTableModel leszármazottja. Képes idő szerint rendezni a tartalmát.

### Tagváltozók

`private List<GameResult> results` : Egy GameResult-okat tároló ArrayList, itt tárolódnak a táblázat sorai.

### Metódusok

`public int getRowCount()` : visszaadja, hogy hány GameResult-ot tárolunk. A táblázat működéséhez szükséges (örökölt metódus).

`public int getColumnCount()` : visszaadja, hogy hány megjelenítendő adattag van egy sorban. A táblázat működéséhez szükséges (örökölt metódus).

`public Object getValueAt(int rowIndex, int columnIndex)` : A megadott pozíción visszaadja az értéket a táblázatból. Az eredményt olyanná kasztolhatjuk, amilyen típust várunk. A táblázat működéséhez szükséges (örökölt metódus).

`public GameResult get(int index)` : visszaadja a táblázat egy sorát, azaz a tárolt ArrayList egy elemét.

`public void setList(List<GameResult> results)` : ennek segítségével kívülről beállíthatunk egy listát az objektumnak.

`public List<GameResult> getList()` : visszaadja az objektumhoz tartozó tárolt listát.

`public String getColumnName(int columnIndex)` : visszaadja a táblázat oszlopainak nevét, ezt a függvényben direktben állítjuk be. Ahhoz szükséges, hogy a táblázat fejléceit el tudjuk nevezni.

`public void addGameResult(GameResult gs)` : ezzel a metódussal a listához hozzáadunk egy GameResult-ot.

`public void sortByTime()` : a lista elemeit az elemek „time” adattagjai szerint rendezi növekvő sorrendbe.

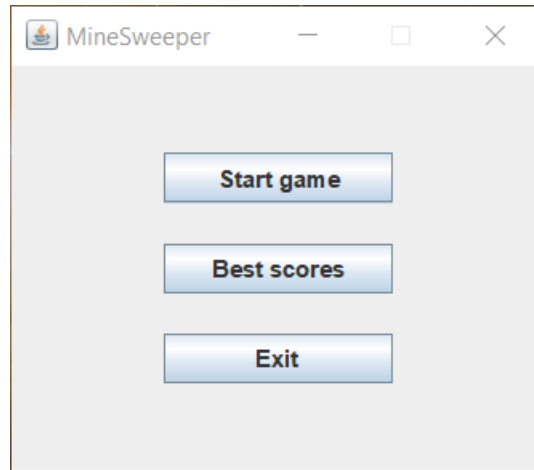
### Belső osztályok

`private class SortByTime implements Comparator<GameResult>` : Két GameResult-ot idő szerint összehasonlító komparátor.

### 3. Felhasználói kézikönyv

#### 3.1. Menü

A programot megnyitva egy menü fogad minket:

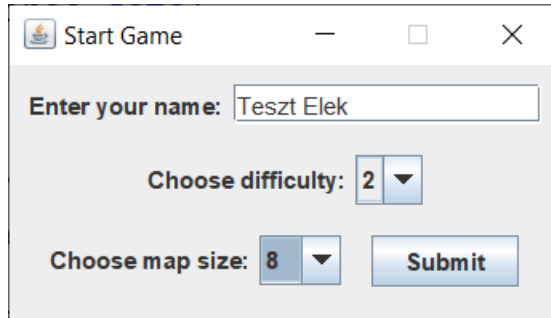


Itt ha a „Start game” gombra kattintunk, új játékot kezdhethetünk. Ha a „Best scores” gombra kattintunk, megnézhetjük a ranglistát. Az „Exit” gombra kilépünk a programból.

## 3.2. Játék

### 3.2.1. Új játék kezdése

Az „Start game” gombra kattintva megjelenik egy ablak, ahol beírhatjuk a nevünket, és kiválaszthatjuk, hogy mekkora pályán, milyen nehézségen szeretnénk játszani. Ha nem változtatjuk meg a nevet, „Anonymous”-ok leszünk. A „Submit” gombra kattintva indul a játék.



Start Game

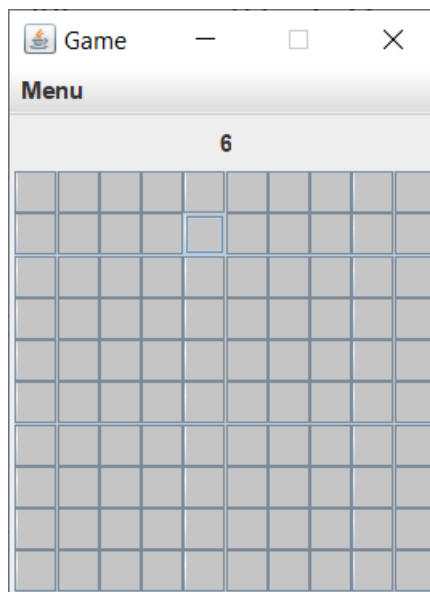
Enter your name:

Choose difficulty:  ▼

Choose map size:  ▼

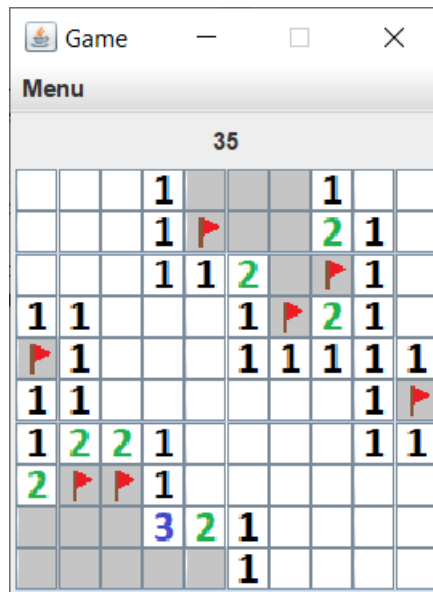
### 3.2.2. Játék

A játék ablakában egy akkora aknamező fogad minket, amekkorát beállítottunk. Felül középen látszik, hogy mennyi ideje játszunk. Kezdetben egy teljesen felderítetlen pálya fogad.



Ha egy mezőre bal gombbal kattintunk, felfedjük a mezőt. Egy mezőn vagy akna van, vagy egy szám, hogy a mező körül hány akna van, kivéve ha 0, ekkor a mező üres. Ha a mező üres, akkor a program automatikusan felderíti az ezzel egybefüggő üres mezőkből álló területet. A mezőkön megjelenő számoknak a segítségével tudunk következtetni a mező körül lévő mezőkön lévő aknákra. Ha a felfedett mezőn akna volt, elveszítjük a játékot. Jobb gombbal egy felfedetlen mezőre kattintva zászlót tehetünk a mezőre. Ezzel magunknak jelezhetjük, ha úgy gondoljuk, hogy ott akna

van. Egy zászlós mezőre ismét jobb gombbal kattintva eltűnik róla a zászló. A játék célja, hogy az összes aknamentes mezőt felderítsük.



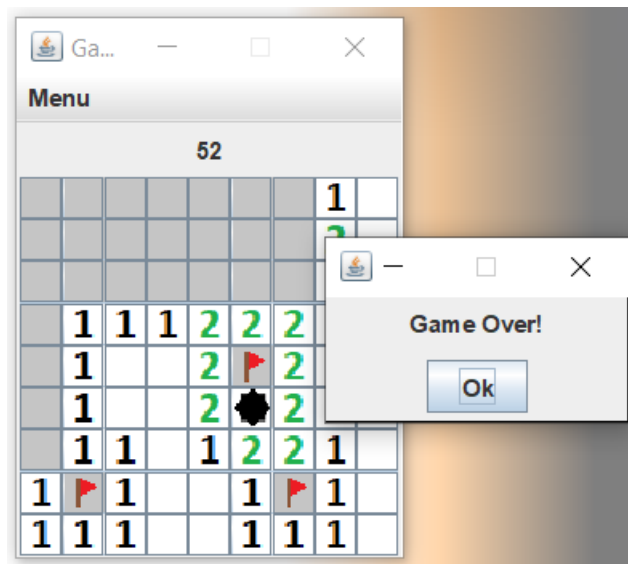
### 3.2.3. Játék megnyerése

Ha az összes aknamentes mezőt felderítettük, megnyertük a játékot, erről egy ablak is tájékoztat minket. Ekkor felkerülünk a ranglistára. Az „Ok” gomb hatására visszakerülünk a menübe.



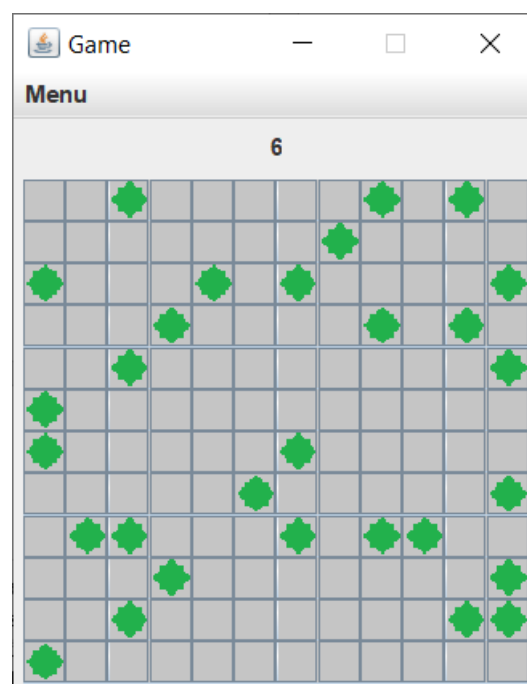
### 3.2.4. Játék elvesztése

Ha aknára léptünk, elveszítjük a játékot, erről is egy ablak tájékoztat. Az „Ok” gomb hatására visszakerülünk a menübe.



### 3.2.5. Csalás

Ha csalni akarunk, a menüben a „Cheat!” gombra kattintva megjelennek a pályán az aknák zöld színnel. Ha ezek után ismét a „Cheat!” gombra kattintunk, megszűnik a csalás, és eltűnnek az aknák.



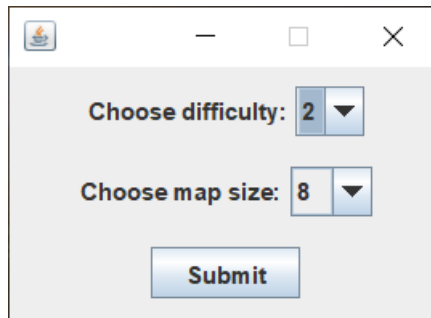
### 3.2.6. Játék feladása

Ha már nincs kedvünk folytatni a játékot, a legördülő menüben a „Give up ☹️” gombra kattintva feladhatjuk a játékot. Ekkor visszakerülünk a menübe.

### 3.3. Ranglista

#### 3.3.1. Kategória kiválasztása

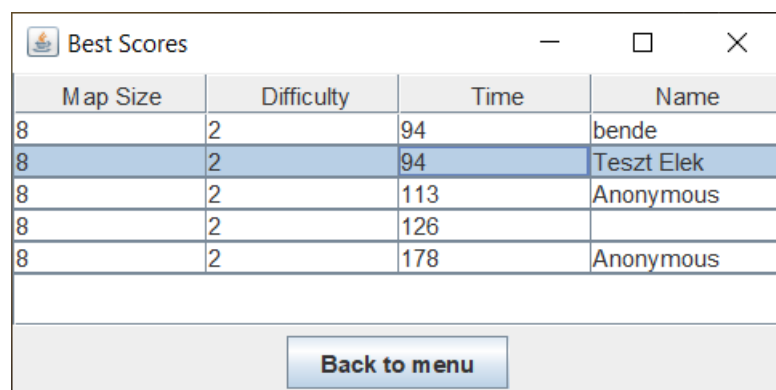
Ha a menüben a „Best scores” gombra kattintunk, akkor megjelenik egy ablak, ahol kiválaszthatjuk, hogy milyen kategóriához szeretnénk megnézni a ranglistát. Egy kategóriát egy pályaméret – nehézség páros definiál. A „Submit” gomb nyitja meg a ranglistát.



A screenshot of a small application window titled "Best Scores". It contains two labels with corresponding dropdown menus: "Choose difficulty:" with a value of "2" and "Choose map size:" with a value of "8". Below these is a "Submit" button.

#### 3.3.2. Ranglista megtekintése

A ranglistát ezek után táblázatos formában, idő szerint rendezve tekinthetjük meg, egy átméretezhető ablakban. A „Back to menu” gomb hatására visszakerülünk a menübe.



A screenshot of the "Best Scores" window. It displays a table with four columns: "Map Size", "Difficulty", "Time", and "Name". The table contains five rows of data. The second row is highlighted. Below the table is a "Back to menu" button.

Map Size	Difficulty	Time	Name
8	2	94	bende
8	2	94	Teszt Elek
8	2	113	Anonymous
8	2	126	
8	2	178	Anonymous