Halas feladat – megoldásrészletek

Feladat:

Az ábrán látható felület belső mérete: 850*460, a vezérlő rész 150 pixel széles.

Induláskor minden üres, csak a háttérkép látszódik.
Az „Adatbetöltés” gomb hatására kerülnek be a halak a rendszerbe. Ez azt jelenti, hogy létre is jönnek a hal példányok: mindnek lesz neve, egyértelműen megadható hozzájuk a bal- és jobboldali profilképük és a képméret. Esetünkben a neveket az *adatok.txt* fájl tartalmazza, a képek a kepek mappában találhatóak a nevek sorrendjében, a méretük pedig véletlenül generálható a megadott határok között. Az adatfájlt egy fájlválasztó segítségével keressük meg.
Adatbetöltés után megjelenik a nevük a listafelületen. A „Vízbedobás” gomb csak akkor válik aktívvá, ha sikerült beolvasni az adatokat.

A „Vízbedobás” gomb hatására a listából kiválasztott halak (egyszerre többet is lehet választani) bekerülnek a vízbe, és ott elkezdenek úszkálni. Egyúttal a listából kikerül a nevük.

Az úszás egyelőre ezt jelenti: véletlen sebességgel véletlenszerűen indulnak balra vagy jobbra, és ha az „akvárium” falához érnek, akkor visszafordulnak.

Oldjuk meg azt is, hogy a hangszóró gombra kattintva szólaljon meg egy háttérzene, majd ismét megnyomva hallgasson el.

2. A vízre való kattintáskor ez lehet:
a/ Ha eltaláltunk egy halat, akkor kikerül a vízből,
b/ ha nem, akkor viszont az összesnek megváltozik a mozgása (megáll, vagy újraindul).

Megoldásrészletek

A megoldásból hiányoznak az eredetileg kiadott részletek, hiányoznak a setterek/getterek, illetve a generált részek. Ezeket remélhetőleg önállóan is meg tudja oldani.

A részletek sorrendje:

1. Adatbevitel
   Ehhez szükség lesz a `Hal` osztály konstruktorára
   a képpárokat tartalmazó `KepPar` osztályra is

2. Vezérlőpanel
   az adatbevitel és a vízbedobás gomb hatása

3. Rajzpanel
    rajzolás és felületre kattintás

4. Vezérlő osztály

1. Adatbevitel:

Beolvasáskor a hal nevén kívül megadjuk még a hozzá tartozó két képet és a képek méretét. Vagyis a `Hal` osztály konstruktorának ezeket az adatokat kell paraméterként tartalmaznia. Mivel fontos, hogy ugyanazt a halat lássuk jobbra vagy balra haladva, ezért fontos, hogy a hozzá tartozó két kép mindig együtt szerepeljen. Emiatt ezeket egy külön osztály megfelelő példányaiban tároljuk majd.

A `Hal` osztály konstruktora és a hozzá tartozó adattag-deklarációk:

```
private String nev;
private int kepSzelesseg, kepMagassag;
private KepPar kepPar;

public Hal(String nev, int kepSzelesseg, int kepMagassag, KepPar kepPar) {
    this.nev = nev;
    this.kepSzelesseg = kepSzelesseg;
    this.kepMagassag = kepMagassag;
    this.kepPar = kepPar;
}
```

A konstruktorban hivatkozott `KepPar` osztály tartalmazza a halhoz tartozó kép-párokat:

```
public class KepPar {

    private Image balKep;
    private Image jobbKep;

    public KepPar(Image balKep, Image jobbKep) {
        this.balKep = balKep;
        this.jobbKep = jobbKep;
    }

    public Image getBalKep() {
        return balKep;
    }

    public Image getJobbKep() {
        return jobbKep;
    }
}
```

Az előkészületek megtétele után jöhet a beolvasás. Mivel kétféle módon is meg akarjuk írni (az adatbázisból való olvasás hf ☺), induljunk ki most is egy interfészből.

Ha csak szigorúan erre a feladatra koncentrálnánk, akkor akár azt is megtehetnénk, hogy a vezérlő-panelen írjuk meg a teljes beolvasást, vagy csak arra figyelnénk, hogy a beolvasott hal példányoknak egy listamodellbe kell kerülniük, és ezért közvetlenül modellbe olvasnánk az adatokat. Ha azonban megpróbáljuk magunkévá tenni azt a szemléletet, amely szerint mindig gondolunk a

könnyű módosíthatóságra, akkor célszerű ezt is függetleníteni a konkrét feladattól, vagyis az adatszerkezetet is és a vezérlést is próbáljuk minél általánosabban kezelni.

Ebből egyrészt az következik, hogy az adatokat egy `List` típusú eredményt visszaadó metódus hozza létre, vagyis gondolni kell arra, hogy esetleg konzolos felületen akarják használni ezeket a példányokat. Persze, a konzolos alkalmazásnak némileg ellentmond az, hogy képeket is akarunk használni ☺. Az viszont lehet érv a lista mellett, hogy esetleg egy másik panelen nem default-, hanem másfajta modellbe szeretnénk rakni őket. Úgyhogy úgy oldjuk meg, hogy a fájlbeolvasó osztályban listába olvassuk az adatokat.

Másrészt a vezérlést pedig úgy írjuk meg, hogy ha valakinek az jut eszébe, hogy mégsem a gombnyomás hatására olvassunk, hanem pl. a rajzpanelre kattintáskor, vagy egy újabb panelhez kapcsolt esemény bekövetkeztekor, vagy esetleg a beolvasott halak listáját kellene máshol megjeleníteni, akkor se kelljen nagyon sokat változtatnunk. Ezért most úgy oldjuk meg, hogy a vezérlő osztályra bízzuk az olvasást, és a gombnyomás hatására a vezérlőpanel csak megkéri a vezérlőt, hogy olvassa be az adatokat.

Előbb az adatbeolvasó interfészt és osztályt mutatom meg:

```
public interface AdatInput {
    public List<Hal> halLista() throws Exception;
}
```

Olvasáskor egy adatfájlból beolvassuk a neveket, majd minden egyes névhez hozzárendelünk egy-egy képpárt. Logikusabb lenne az adatfájlban a névhez tartozó képneveket is megadni (ekkor ugyanis garantáltan ugyanaz a kép tartozik egy adott névhez), de most (főleg azért, hogy ilyenre is lásson példát), a képeket sorszám szerint különböztetjük meg, és a beolvasás sorrendjében rendeljük a halakhoz. Ehhez nyilván kell majd egy változó, amelyben a sorszámot tároljuk, és evvel együtt hivatkozunk a fájlnévre.
Vitatható az a megoldás, amelyet most választok, vagyis az, hogy a képfájlok elérési útvonalát tartalmazó stringbe a fájlnév közös részét is belevesszük, de most mégis így csináljuk. A megoldás hátránya: a változónév nem, vagy csak „kacifántosan” elnevezve takarja pontosan a tartalmat.
Előnye: bármely olyan esetben használható, amikor kezdőszöveg+sorszám+BAL/JOBB szerkezetű a fájlnév.
A képméreteket pedig adott határok között véletlenszerűen adjuk meg.

```
public class FajlBevitel implements AdatInput{

    private File fajl;
    private String CHAR_SET = "UTF-8";
    private double felsoMeret;
    private double alsoMeret;
    private String kepfajlEleres;

    public FajlBevitel(File fajl, double felsoMeret, double alsoMeret,
                       String kepfajlEleres) {
        this.fajl = fajl;
        this.felsoMeret = felsoMeret;
        this.alsoMeret = alsoMeret;
        this.kepfajlEleres = kepfajlEleres;
    }
```

```
    @Override
    public List<Hal> halLista() throws Exception {
        List<Hal> halak = new ArrayList<>();
        try(Scanner fajlScanner = new Scanner(fajl, CHAR_SET)){
            int sorszam = 0;
            String nev;
            int kepSzelesseg, kepMagassag;
            Image balKep, jobbKep;
            KepPar kepPar;
            while (fajlScanner.hasNextLine()) {
                nev = fajlScanner.nextLine();
                kepSzelesseg =
                     (int) (Math.random()*(felsoMeret - alsoMeret) + alsoMeret);
                kepMagassag =
                     (int) (Math.random()*(felsoMeret - alsoMeret) + alsoMeret);
                balKep =
                    new ImageIcon(this.getClass().
                    getResource(kepfajlEleres + sorszam + "_BAL.png")).getImage();
                jobbKep =
                    new ImageIcon(this.getClass().
                    getResource(kepfajlEleres + sorszam + "_JOBB.png")).getImage();
                kepPar = new KepPar(balKep, jobbKep);
                halak.add(new Hal(nev, kepSzelesseg, kepMagassag, kepPar));
                sorszam++;
            }
        }
        return halak;
    }
}
```

Gyakorlaton kicsit felemás módon oldottuk meg a feladatot, ugyanis a vezérlőpanel döntötte el, hogy egy dialógusablakból kiválasztott fájlból olvassunk, vagyis ha adatbázisból szeretnénk, akkor mégiscsak hozzá kellene nyúlni a vezérlőpanel osztály kódjához is. Most még általánosabb megoldást mutatok, mégpedig olyat, hogy valóban csak a vezérlő osztály kódját kelljen kicsit módosítani, ha mégis adatbázisból szeretnénk olvasni.

Vezérlőpanel:

A vezérlőpanelnek három funkciója van: az adatbevitel, a halak vízbe dobása és a zene ki- be-kapcsolása. Pontosabban: ezek egyike sem az ő dolga, az ő dolga csak annyi, hogy érzékelje azokat az eseményeket, amelyek hatására az előbb említett funkciókat szeretnénk megvalósítani, és jelezze a vezérlőnek, hogy bekövetkeztek ezek az események, vagyis, hogy oldja meg a kért funkciókat.

Az adatbevitelt úgy oldjuk meg, hogy közben figyelünk az MVC/MVP szemléletre.
(A teljesen „tisztességes” változatot csak a *tovabbi_halas_variaciok.pdf* fájlban mutatom majd meg.)

A szemlélet szerint a panelek kizárólag megjelenítésre valók, vagyis úgy kellene megoldani a feladatot, hogy abban pillanatok alatt ki lehessen cserélni a paneleket, vagyis tényleg csak a megjelenítést és a vezérlő osztállyal való kapcsolatot tartalmazzák. Ezért nem itt, hanem a vezérlő osztályban írjuk meg az `adatBeolvasas()` metódust. Ez azonban további kérdéseket és módosítandó részleteket vet fel. Egyrészt, ha igazán általánosan szeretnénk megoldani a feladatot,

akkor a fájlválasztást is a vezérlőre kellene bízni. A másik probléma az, hogy valahogy meg kell oldani, hogy a gombok aktivitása csak akkor változzon meg, ha sikeres az olvasás. A probléma megoldása az, hogy a vezérlőben `boolean` metódusként írjuk meg az adatbeolvasást, és a panel majd akkor hívja meg a gombváltást, ha sikeresen futott le a beolvasás.

A harmadik megoldandó részlet az, hogy most az adatok a vezérlő osztályban vannak, valahogyan át kellene kerülniük a panel listafelületére – ezt meg lehet oldani az `adatBeolvasas()` metóduson belül, itt kell átadni őket a vezérlőpanelnek. Mivel beolvasás után azonnal feldolgozzuk a beolvasott lista adatait, ezért azt elég csak lokálisan deklarálni – a vezérlő osztálynak nincs szüksége a teljes listára, csak a vízben lubickolókra.

A negyedik: a listafelületre való kiíratás miatt a vezérlő osztálynak ismernie kell a vezérlő panelt, vagyis deklarálni kell egy ilyen típusú példányt, és lehetővé tenni, hogy értéket kapjon (vagy a konstruktorban kell szerepeltetni, vagy settert kell írni hozzá).

A vezérlőpanelen az adatbevitelért felelős gombnyomás a vezérlő `adatBeolvasas()` metódusát hívja meg. Ehhez deklarálni kell egy `Vezerlo` típusú `vezerlo` változót, és persze, a hozzá tartozó settert is meg kell írni. Miután a vezérlő beolvassa az adatokat, átadja őket a panelnek. Ezt dolgozza fel a panel a `listabaIr()` metódusban. A `VezerloPanel` beállításai és az adatbeolvasás- és megjelenítés lépései:

```
public class VezerloPanel extends javax.swing.JPanel {

    private final Icon hangBe =
        new ImageIcon(getClass().getResource(Global.IKON_ELERES+"/hangbe.png"));
    private Icon hangKi =
        new ImageIcon(getClass().getResource(Global.IKON_ELERES+"/hangki.png"));
    private int gombMeret = 30;
    private Vezerlo vezerlo;

    private DefaultListModel<Hal> halModell = new DefaultListModel<>();

    public VezerloPanel() {
        initComponents();
        gombAktivitas(false);
        btnHang.setSize(gombMeret, gombMeret);
        btnHang.setLocation(this.getWidth()/2 - gombMeret/2,
                            this.getHeight()-2*gombMeret);
        btnHang.setIcon(hangBe);
        lstHalak.setModel(halModell);
    }

    public void setVezerlo(Vezerlo vezerlo) {
        this.vezerlo = vezerlo;
    }
```

```
    private void btnAdatbevitelActionPerformed(java.awt.event.ActionEvent evt) {
        if(vezerlo.adatBeolvasas()) gombAktivitas(true);
    }

    public void listabaIr(List<Hal> beolvasottHalak) {
        for (Hal hal : beolvasottHalak) {
            halModell.addElement(hal);
        }
    }
```

A `Vezerlo` osztály hivatkozott beolvasási metódusa:

```
    public boolean adatBeolvasas() {
        try {
            JFileChooser fajlValaszto = new JFileChooser(new File("."));
            if (fajlValaszto.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
                File fajl1 = fajlValaszto.getSelectedFile();
                AdatInput adatInput = new FajlBevitel(fajl1, Global.FELSO_KEPMERET,
                        Global.ALSO_KEPMERET, Global.KEPFAJL_ELERES);
                List<Hal> beolvasottHalak = adatInput.halLista();
                vezerloPanel.listabaIr(beolvasottHalak);
                return true;
            }
        } catch (Exception ex) {
            Logger.getLogger(Vezerlo.class.getName()).log(Level.SEVERE, null, ex);

        }
        return false;
    }
```

A halak vízbedobása elég egyszerű: a kiválasztott halakat egyenként ki kell törölni a modellből, és ugyanakkor megkérni a vezérlőt, hogy rakja vízbe.

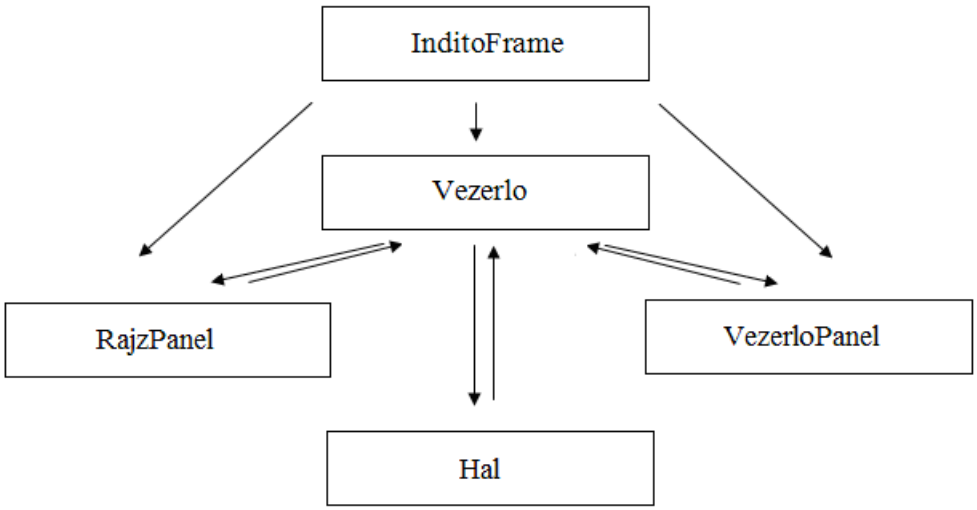
```
    private void btnVizbeActionPerformed(java.awt.event.ActionEvent evt) {
        List<Hal> halak = lstHalak.getSelectedValuesList();
        for (Hal hal : halak) {
            vezerlo.vizbeDob(hal);
            halModell.removeElement(hal);
        }
    }
```

A zenegomb működése még ennél is egyszerűbb:

```
    private void btnHangActionPerformed(java.awt.event.ActionEvent evt) {
        if(btnHang.getIcon().equals(hangBe)){
            vezerlo.zeneBekapcsolas();
            btnHang.setIcon(hangKi);
        }
        else{
            vezerlo.zeneKikapcsolas();
            btnHang.setEnabled(false);
        }
    }
```

Beszéljük meg azt is, hogy hol hívjuk meg a vezérlő setterét. De hogy ne aprózzuk el teljesen, gondoljuk végig az egész szerkezetet.

A feladatmegoldás során az újrahasznosíthatóság is célunk, ezért úgy akarjuk megoldani, hogy az egyes osztályok minél függetlenebbek legyenek egymástól, vagyis úgy, hogy mindegyik csak a vezérlővel tartson kapcsolatot.



A kapcsolatok közül eddig még csak arról volt szó, hogy a vezérlőpanelnek ismernie kell a vezérlőt, hiszen neki kell szólnia, ha megnyomták valamelyik gombját. Azonban az ismeretségnek kölcsönösnek kell lennie, hiszen a vezérlőnek meg tudnia kell, hogy melyik panelt kérje meg az adatok kiíratására.
A rajzpanel azt rajzolja majd, amit a vezérlő mond neki, ezért a rajzpanelnek is ismernie kell a vezérlőt. Ez az ismeretség is kölcsönös, mert a vezérlőnek is ismernie kell a rajzpanelt, hiszen időnként frissítésre kéri.
A vezérlő kezeli a kiválasztott halakat, ezért nyilván ismernie kell őket, ugyanakkor a hal példányoknak is ismerniük kell a vezérlőt, hiszen időnként őt kérik majd frissítésre.

Megjegyzés: A most tárgyalt megoldás nem ennyire tiszta szerkezetű, ugyanis a vezérlőpanel listamodellje `Hal` típusú példányokat tartalmaz, vagyis a megoldásunkban van egy „titkos” kapcsolat a vezérlőpanel és a `Hal` osztály között. Ennek kiküszöböléséről majd egy másik segédletben lesz szó (*tovabbi_halas_variaciok.pdf*).

Az osztályok közötti kapcsolat az `AkvariumFrame`-n alakítható ki. Mivel beállításról van szó, a metódus most meghívható a frame konstruktorából (vagy a konstruktorból hívott `beallitas()` metódusból – természetesen nem muszáj külön metódust írni, csak most így talán jobban látszik, hogy mit kell beállítanunk):

```
private void beallit() {
    Vezerlo vezerlo = new Vezerlo(rajzPanel1, vezerloPanel1);
    vezerlo.setRajzPanel(rajzPanel1);
    vezerloPanel1.setVezerlo(vezerlo);
    rajzPanel1.setVezerlo(vezerlo);
}
```

A **rajzpanel**nek nagyon egyszerű dolga van: ki kell rajzolnia azt, amit a vezérlő mond, illetve szólni kell neki, ha rákattintottak a felületére:

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.drawImage(hatterKep, 0, 0, this.getWidth(), this.getHeight(), null);
    if(vezerlo != null) vezerlo.rajzol(g);
}

private void formMousePressed(java.awt.event.MouseEvent evt) {
    vezerlo.talalatVizsgalat(evt.getX(), evt.getY());
}
```

A hal példányok létrehozásakor már szó volt a `Hal` osztály konstruktoráról, de most részletesebben meg kellene tárgyalni ezt is és a `Vezerlo` osztályt is.
Mindkét osztályból csak a kommentezett kódrészleteket szeretném közölni, ezért most röviden beszéljük meg a feladataikat.

A hal feladatai:

- Megmondja, hogyan lehet kirajzolni.
- Azt is megmondja, hogyan lehet kiíratni (toString())
- Mivel mozog, ezért szálként kezelendő, és meg kell írni a `run()` metódusát.
- Vizsgálni kell, hogy eltalálták-e.
- Lehetőséget kell adnia a „működésváltásra”, vagyis arra, hogy hol ússzon, hol ne.

A vezérlő feladatai:

- Egy adott hal vízbedobása: ekkor be kell állítani a hal szükséges adatait, elindítani a szálat, és hozzáadni a kirajzolandó halak listájához.
- A halak kirajzolása.
- A rajzpanel frissítésének kérése
- Annak vizsgálata, hogy mi történjen, ha a rajzfelületre kattintottak.
- A zene be- és kikapcsolása.

Lényegében ennyi, de még egy dolgot kiemelek: a rajzpanel átvételekor be kell állítanunk az akvárium méreteit is. Mivel az összes hal ugyanabban az akváriumban úszkál, ezért ezek a méretek statikusak.

Ezek után a `Hal` osztály (setterek/getterek nélkül):

```
public class Hal extends Thread{

    private String nev;
    private int kepX;
    private int kepY;
    private int kepSzelesseg, kepMagassag;

    private KepPar kepPar;
    private Image kep;

    private long ido;
    private boolean aktiv;
    private int lepesKoz;
    private Vezerlo vezerlo;
    private boolean uszik;

    private static int akvariumSzelesseg;
    private static int akvariumMelyseg;
    // ilyen magasságig úszhatnak a halak
    private static int vizMagassag;

    public Hal(String nev, int kepSzelesseg, int kepMagassag, KepPar kepPar) {
        this.nev = nev;
        this.kepSzelesseg = kepSzelesseg;
        this.kepMagassag = kepMagassag;
        this.kepPar = kepPar;
    }

    public void beallit (int kepX, int kepY, long ido, boolean aktiv,
                         int lepesKoz, Vezerlo vezerlo) {
        this.kepX = kepX;
        this.kepY = kepY;
        this.ido = ido;
        this.aktiv = aktiv;
        this.lepesKoz = lepesKoz;
        this.vezerlo = vezerlo;
        kepBeallitas();
    }

    private void kepBeallitas() {
        kep = (lepesKoz < 0) ? kepPar.getBalKep() : kepPar.getJobbKep();
    }

    public void rajzol(Graphics g){
        g.drawImage(kep, kepX, kepY, kepSzelesseg, kepMagassag, null);
    }
```

```
@Override
public void run() {
    while (aktiv) {
        varakozik();
        mozdul();
        frissit();
        pihen();
    }
}

private void mozdul() {
    kepX += lepesKoz;
    if(kepX >= akvariumSzelesseg - this.kepSzelesseg || kepX <= 0){
        lepesKoz = -lepesKoz;
        kepBeallitas();
    }
}

private void pihen() {
    try {
        Thread.sleep(ido);
    } catch (InterruptedException ex) {
        Logger.getLogger(Hal.class.getName()).log(Level.SEVERE, null, ex);
    }
}

private void frissit() {
    vezerlo.frissit();
}

public boolean eltalaltak(int x, int y) {
    return kepX <= x && x <= kepX + kepSzelesseg &&
           kepY <= y && y <= kepY + kepMagassag;
}

public synchronized void mukodesValtas() {
    uszik = !uszik;
    if(uszik) notify();
}

private synchronized void varakozik(){
    if(!uszik){
        try {
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Hal.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

A `Vezerlo` osztály (az adatbevitel nélkül, hiszen az már szerepelt korábban):

```java
public class Vezerlo {

    private RajzPanel rajzPanel;
    private VezerloPanel vezerloPanel;

    private Zene zene;
    // Szálpéldányok kezelésére ez a listafajta biztonságos
    private List<Hal> uszoHalak = new CopyOnWriteArrayList<>();

    public Vezerlo(RajzPanel rajzPanel, VezerloPanel vezerloPanel) {
        this.rajzPanel = rajzPanel;
        this.vezerloPanel = vezerloPanel;
        beallitas();
    }

    private void beallitas() {
        Hal.setAkvariumMelyseg(rajzPanel.getHeight()-Global.FELSO_KEPMERET);
        Hal.setAkvariumSzelesseg(rajzPanel.getWidth());
        Hal.setVizMagassag((int) (rajzPanel.getHeight()*
                                  Global.KEP_MAGASSAG_ARANY));
        zene = new Zene();
    }

    public void vizbedob(Hal hal) {
        int kepX = (int) (Math.random()
                  *(Hal.getAkvariumSzelesseg()-hal.getKepSzelesseg()));
        int kepY = (int) (Math.random()
                  *(Hal.getAkvariumMelyseg()-Hal.getVizMagassag()
                                             -hal.getKepMagassag()));
        long ido = (long) (Math.random()*(Global.FELSO_IDO-Global.ALSO_IDO)
                                          + Global.ALSO_IDO);
        int lepeskoz = (Math.random() < 0.5)? 1 : -1;
        hal.beallitas(kepX, kepY, true, this, ido, lepeskoz);
        uszoHalak.add(hal);
        // akkor van szükség frissítésre, ha nem akarunk szálat indítani
        // frissit();
        hal.setUszik(true);
        hal.start();
    }

    public void rajzolas(Graphics g) {
        for (Hal hal : uszoHalak) {
            hal.rajzolas(g);
        }
    }

    public void frissit() {
        rajzPanel.repaint();
    }
```

```
    public void zeneBekapcsolas() {
        if(!zene.isAlive()){
            zene.setZeneFajlEleres(Global.ZENEFAJL_ELERES);
            zene.start();
        }
    }

    public void zeneKikapcsolas() {
        zene.leall();
    }

    public void talalatVizsgalat(int x, int y) {
        boolean eltalaltakEgyet = false;
        for (Hal hal : halak) {
            if(hal.eltalaltak(x,y)){
                kiszed(hal);
                eltalaltakEgyet = true;
                break;
            }
        }
        if(!eltalaltakEgyet){
            for (Hal hal : halak) {
                hal.mukodesValtas();
            }
        }
    }

    private void kiszed(Hal hal) {
        hal.setAktiv(false);
        halak.remove(hal);
        frissit();
    }
}
```