A pillangós feladat játék részének megoldásrészletei

Feladat:

Írjunk egy kis lepkefogó játékot!
Az 1000*600-as belső felület felső, 50 pixel magas részén lévő gombbal lehet indítani a játékot. Ugyanitt olvasható a hátralévő játék-idő, és az elért pontszám is.

A mező fölött időnként megjelenik egy-egy lepke, és elkezd repülni – most elég egyszerűen, csak négy lehetséges irányban mozog egyenletes, de véletlen sebességgel.
(Véletlen pozíció, véletlen méret, véletlen sebesség, véletlen, hogy milyen irányban indulnak: balra le/föl vagy jobbra le/föl ↖↙↗↘).

A gomb megnyomásakor a lepkevadász is megjelenik a mező jobb alsó sarkában. Őt az egérrel tudjuk mozgatni, és az a cél, hogy minél több lepkét el tudjon kapni. Az elkapott lepkékért egy-egy pontszám jár.

Folytatásként oldja meg, hogy akár egy kis lepkefogó versenyt is részt vehessünk:

Induláskor adjuk meg a játékos nevét, és ha lejárt a játékideje, a listában jelenjen meg a neve és a pontszáma. Ha egy játékos többször is játszik, akkor csak a legutolsó eredménye jelenjen meg (vagy ha lágyabb szívű, akkor a legjobb). A listafelületen pontszám szerint csökkenően rendezve legyenek az adatok.

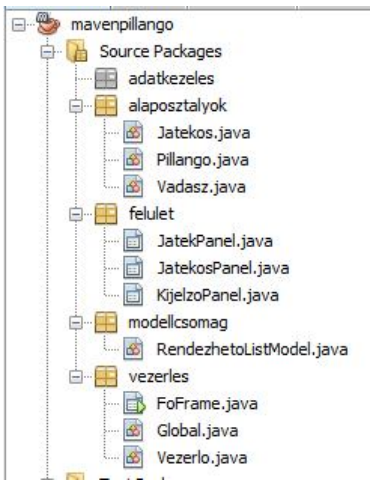
Még további folytatásként a listában lévő adatokat mentsük el egy adatbázisba, a következő induláskor az adatbázisból olvassa ki a játékosok adatait, és jelenítse is meg a listafelületen.
Mentéskor értelemszerűen csak az újakat kell beszúrni, a korábban is létezőket pedig módosítani.

Ha belejött az adatbázis-kezelésbe, akkor tegyen fel még egy gombot, és törölje ki az adat bázisból a listafelületen kijelölt embereket.

Néhány megoldásrészlet:

A közölt kódrészletek az indulásként kiadott osztályokból, ill. a setterek/getterek közül csak a megértéshez feltétlenül szükséges részleteket tartalmazzák majd.

A megoldás csomagszerkezete:



A `Global` osztály tartalmazza a szükséges konstansokat, a `Jatekos` és a `Pillango` osztályokat kicsit részletezem majd, a `Vadasz` pedig összesen annyi, hogy meg tudja mondani (`rajzolas()` metódus), hogy a hozzá tartozó képet kell majd kirajzolni.

A feladatot egyetlen panellel is meg lehetne oldani, de három részre szedve jobban elkülönülnek a funkciók, illetve könnyebben lehet módosítani további igények esetén, hiszen bármelyik három bármikor helyettesíthető egy másik panellel.

A `JatekPanel`-en zajlik a játék. Ha csak ezt az egyetlen panelt tennénk fel a frame-re, akkor is lehetne játszani, csak persze, jóval kevesebbet tudna a játék. Betöltéskor azonnal elindulnának a pillangók, és a vadász tudná fogdosni őket. (Annyit kellene csak változtatni a kész projekthez képest, hogy az indítást nem a gombnyomás, hanem a form betöltésének eseményéhez rendelnénk.)

Ha ehhez hozzárakjuk a `KijelzoPanel`-t, akkor már az idő múlását is láthatnánk, a pontokat is, és akkor már innen indíthatnánk a játékot. Ez a két panel is tud együtt dolgozni, ill. akkor is működőképes a projekt, ha csak ez a két panel szerepel a frame-n.

Ha még a `JatekosPanel`-t is hozzárakjuk, akkor játékosokat is tudunk rendelni a játékhoz, és az eredményüket is tudjuk kezelni. Elvileg akkor is működőképes a projekt (ugyancsak egy minimális módosítás árán), ha csak ez és a `JatekPanel` van fent a frame-n, de így kevésbé lenne érdekes a játék.

Ahogy már megszokta, és remélem, nem csak megszokta, hanem érti is és egyet is ért evvel a logikával, a panelek közti kapcsolatot, illetve a pillangókat, vadászt is a `Vezerlo` kezeli.

Az adatkezelést majd később részletezzük.

A továbbiakban néhány megoldásrészlet.

Alaposztályok:

A `Jatekos` osztályból csak egy kis kódrészletet emelek ki:

```
    //Akkor hívjuk meg, amikor a játékban létrehozunk egy példányt
    public Jatekos(String nev) {
        this.nev = nev;
    }

    public void pontotKap(){
        pontSzam++;
    }
```

A pillangó mozog, ezért szálként oldjuk meg:

```
public class Pillango extends Thread{

    private Image kep;
    private int kepX, kepY;
    private int kepSzelesseg, kepMagassag;
    private Vezerlo vezerlo;
    private long ido;
    private int dx;
    private int dy;
    private boolean elkaptak;

    private static int feluletSzelesseg, feluletMagassag;

    public Pillango(Image kep, int kepX, int kepY,
                    int kepSzelesseg, int kepMagassag,
                    int dx, int dy, Vezerlo vezerlo, long ido) {
        this.kep = kep;
        this.kepX = kepX;
        this.kepY = kepY;
        this.kepSzelesseg = kepSzelesseg;
        this.kepMagassag = kepMagassag;
        this.dx = dx;
        this.dy = dy;
        this.vezerlo = vezerlo;
        this.ido = ido;
    }

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

```
@Override
public void run() {
    // Addig mozog, amíg a felület felett van és nem kapták el.
    while (kepX > -kepSzelesseg && kepX < feluletSzelesseg &&
            kepY > -kepMagassag && kepY < feluletMagassag && !elkaptak) {
        mozdul();
        frissit();
        kesleltet();
    }
    if(elkaptak) vezerlo.torol(this);
}

// A kezdeti irányt tartva ferdén mozog.
private void mozdul() {
    kepX += dx;
    kepY += dy;
}

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

private void kesleltet() {
    try {
        Thread.sleep(ido);
    } catch (InterruptedException ex) {
        Logger.getLogger(Pillango.class.getName()).log(Level.SEVERE, null, ex);
    }
}
/**
 * Akkor fogták meg, ha az adott koordinátájú pont a pillangó
 * képén belülre esik.
 *
 * @param kx
 * @param ky
 */
public void megfogtak(int kx, int ky) {
    if (kepX <= kx && kx <= kepX + kepSzelesseg
            && kepY <= ky && ky <= kepY + kepMagassag) {
        elkaptak = true;
    }
}
```

+ setterek, getterek.

Két panel:

A `JatekPanel` és a `KijelzoPanel` feladata nagyon egyszerű: végre kell hajtaniuk a vezérlő parancsait (na jó, teljesíteniük kell a vezérlő kéréseit), illetve jelezni a vezérlőnek, ha valamilyen esemény történt rajtuk.
A harmadik panel feladatáról majd kicsit később esik szó.

A `JatekPanel` két metódusa (a deklarációkat, settereket nem másolom ide):

```
// Kirajzolja a háttérképet, és azt, amit a vezérlő mond neki.
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    int kezdox = 0, kezdoy = 0,
        szelesseg = this.getWidth(),
        magassag = this.getHeight();
    g.drawImage(hatterKep, kezdox, kezdoy, szelesseg, magassag, null);

    if(vezerlo != null) vezerlo.rajzol(g);
}
```

```
// Szól a vezérlőnek, hogy arrébb mozgatták az egeret.
private void formMouseDragged(java.awt.event.MouseEvent evt) {
    vezerlo.arrebbMozgat(evt.getX(), evt.getY());
}
```

A `KijelzoPanel` metódusai ugyancsak deklaráció és setter nélkül:

```
// Jelzi a vezérlőnek, hogy megnyomták a gombot
private void btnInditasActionPerformed(java.awt.event.ActionEvent evt) {
    vezerlo.jatekInditas();
}
```

```
public void idoKiir(long aktIdo) {
     lblIdo.setText(String.format("Hátralévő idő: %d sec", aktIdo));
}

public void pontszamKiir(int pontSzam) {
    lblPontszam.setText(String.format("Pontszám: %d pont", pontSzam));
}

// Aktívvá teszi a gombot, és eltünteti az időre vonatkozó kiírást.
public void jatekVeg() {
    btnIndit.setEnabled(true);
    lblIdo.setText("");
}

// Inaktívvá teszi a gombot
public void jatekKezdet() {
    btnIndit.setEnabled(false);
}
```

Következzen a **`Vezerlo`** osztály, de hogy ne kelljen több részletben közölni, ezért előbb beszéljünk még meg valamit, mégpedig azt, hogy hogyan keletkeznek a pillangók.

A feladat szerint bizonyos időközönként keletkeznek, ezért szükségünk van valamilyen időzítőre. Talán legegyszerűbb a Timer használata lenne, de most a szálkezelés gyakorlása a célunk, ezért ezt is szállal oldjuk meg (ami egyébként nem bonyolultabb a Timer használatánál.)

Kérdés, hogy honnan szedjünk szálat. Erre több megoldás is kínálkozik:

1. Hozzunk létre egy új szál osztályt, (pl. **`PillangoGenerator`** `extends Thread {...}`), és az indító gomb hatására ezt indítsuk el.

2. Nem kell új osztály, a vezérlő is viselkedhet szálként.

Mindkét megoldás elfogadható, bár az első talán kicsit nehézkesebb, mint az első kettő. Vagyis nagyrészt szubjektív, hogy ki melyik megoldást választja.

Órán a 2. megoldást választottuk, most is ezt mutatom meg.

Szálat kétféle módon hozhatunk létre:
- kiterjesztjük a `Thread` osztályt;
- implementáljuk a `Runnable` interfészt, és az így kapott osztály egy példányát adjuk át egy `Thread` példánynak.

Esetünkben mindkét megoldás jó, azért választjuk a másodikat, hogy erre is lásson egy kidolgozott példát.

Még egy kérdés: hogyan lehet megoldani, hogy több játékot is lehessen indítani egymás után. Egy szálat csak egyszer lehet elindítani, ezért ebben is kétféle lehetőségünk van:
- Minden indításkor új szálat hozunk létre. (Ekkor persze gondoskodni kell arról is, hogy a korábbi szál leálljon, de ez sokszor a `run()` metódus ügyes megírásával is megoldható.)
- Egyetlen szállal dolgozunk, ha a játékos ideje lejárt, akkor várakoztatjuk a szálat, ha új játékos jelentkezik, akkor pedig felengedjük. Azt gondolom, hogy esetünkben ez most bonyolultabb megoldás.

Most tehát a vezérlő osztályt a `Runnable` interfész implementálásaként definiáljuk, és minden játékindításkor új szálat hozunk létre. **FONTOS**: Új szál létrehozása jóval nehezebb lenne akkor, ha `Thread`-ként deklaráltuk volna a vezérlő osztályt. Így, hogy `Runnable` típusú, bármikor létrehozhatunk belőle egy-egy szál osztályt.
A `run()` metódus a játékidő lejártáig fut, utána leáll, ezért nem kell külön foglalkoznunk a szál leállításával, a pillangóknál azonban erre sem árt odafigyelni. Ha ugyanis a pillangó elkapásakor kiszedjük a rajzolandók listájából, akkor csak annyit érünk el, hogy ne lássuk, de maga a szál még él egy kis ideig, ezért törléskor célszerű a szálat is megszüntetni.

A `Vezerlo` osztály:

```
public class Vezerlo implements Runnable{

    private JatekPanel jatekPanel;
    private KijelzoPanel kijelzoPanel;
    private JatekosPanel jatekosPanel;

    private double felsoMeret;
    private double alsoMeret;
    private long felsoIdo;
    private long alsoIdo;
    private int kepSzam;

    private long jatekIdo;
    private long keletkezesiIdo;

    private List<Image> kepek = new ArrayList<>();
    //Mivel szálpéldányokat tartalmaz, ezért szálbiztos lista kell
    private List<Pillango> pillangok = new CopyOnWriteArrayList<>();

    private Jatekos jatekos;
    private Vadasz vadasz;

    public Vezerlo(JatekPanel jatekPanel, KijelzoPanel kijelzoPanel,
                   JatekosPanel jatekosPanel) {
        this.jatekPanel = jatekPanel;
        this.kijelzoPanel = kijelzoPanel;
        this.jatekosPanel = jatekosPanel;

        // Beállítja a pillangókhoz tartozó felületméretet.
        Pillango.setFeluletSzelesseg(jatekPanel.getWidth());
        Pillango.setFeluletMagassag(jatekPanel.getHeight());
    }

    /**
     * Átveszi az adatokat a Global osztályból - ez akár ki is maradhat,
     * csak talán olvashatóbb a kód, ha rövidebb változóneveket használunk.
     *
     * Ezen kívül elvégez néhány, az indításkor szükséges teendőt.
     */
    void beallitas() {
        alsoIdo = Global.ALSO_IDO_MSEC;
        felsoIdo = Global.FELSO_IDO_MSEC;
        alsoMeret = Global.ALSO_PILLANGO_MERET_PIXEL ;
        felsoMeret = Global.FELSO_PILLANGO_MERET_PIXEL;
        kepSzam = Global.KEP_SZAM;
        jatekIdo = Global.JATEK_IDO_SEC;
        keletkezesiIdo = Global.KELETKEZESI_IDO_MSEC;

        // Beállítja a vadász méretét.
        Vadasz.setKepSzelesseg(Global.VADASZ_SZELESSEG);
        Vadasz.setKepMagassag(Global.VADASZ_MAGASSAG);
```

```
        // Beállítja a felület méretét.
        Pillango.setFeluletSzelesseg(jatekPanel.getWidth());
        Pillango.setFeluletMagassag(jatekPanel.getWidth());

        kepFeltoltes();
    }

    public void kepFeltoltes(){

        for (int i = 1; i <= kepSzam; i++) {
            kepek.add(new ImageIcon(this.getClass().
                    getResource("/kepek/pillango"+i+".gif")).getImage());
        }
    }
```

Mivel a játékpanel azt rajzolja, amit a vezérlő mond neki, ezért itt kell megadni a vadász kirajzolására vonatkozó leírást is, természetesen a pillangók rajzolást leíró metódusának meghívása mellett.

```
    public void rajzol(Graphics g) {
        for (Pillango pillango : pillangok) {
            pillango.rajzol(g);
        }
        if(vadasz != null) vadasz.rajzol(g);
    }

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

Nem akartam megtörni, ezért a következő oldalon folytatódik a kód

```java
/**
 * Bizonyos időnként elindít egy-egy pillangót,
 * az idő lejárta után jelzi a paneleknek, hogy ez a játék véget ért
 * és elvégzi a leállításkor szükséges műveleteket.
 */
@Override
public void run() {
    long aktIdo = jatekIdo*1000;
    while(aktIdo >= 0 ){
        try {
            kijelzoPanel.idoKiir(aktIdo/1000);
            pillangotIndit();
            Thread.sleep(keletkezesiIdo);
            aktIdo -= keletkezesiIdo;
        } catch (InterruptedException ex) {
            Logger.getLogger(KijelzoPanel.class.getName()).log(Level.SEVERE, r
        }
    }
    kijelzoPanel.jatekVeg();
    jatekosPanel.jatekVeg();
    leallit();
}

/**
 * Beállítja a szükséges adatokat, létrehoz egy pillangót, és elindítja.
 */
public void pillangotIndit() {
    int kepSzelesseg = (int) (Math.random()*(felsoMeret-alsoMeret)
                                                    + alsoMeret);
    int kepMagassag = (int) (Math.random()*(felsoMeret-alsoMeret)
                                                    + alsoMeret);
    int kx = (int) (Math.random()*(jatekPanel.getWidth()
                                            - kepSzelesseg) + kepSzelesseg);
    int ky = (int) (Math.random()*(jatekPanel.getHeight()
                                            - kepMagassag) + kepMagassag);
    long ido = (long) (Math.random()*(felsoIdo - alsoIdo) + alsoIdo);
    int kepIndex = (int) (Math.random()*kepek.size());
    Image kep = kepek.get(kepIndex);
    int dx = (Math.random() < 0.5) ? 1 : -1;
    int dy = (Math.random() < 0.5) ? 1 : -1;

    // Létrehoz és elindít egy pillangót.
    Pillango pillango = new Pillango(kep, kx, ky,
                                    kepSzelesseg, kepMagassag,
                                    dx, dy, this, ido);
    pillango.start();
    pillangok.add(pillango);
}
```

Egyelőre csak egy fiktív játékost hozunk létre, és vele indítjuk a játékot, később majd visszatérünk az indítás és a játékos megadásának kérdésére.

```
public void jatekInditas() {

    this.jatekos = new Jatekos("nev");
    vadasz = new Vadasz(jatekPanel.getWidth()-Vadasz.getKepSzelesseg()/2,
                        jatekPanel.getHeight()-Vadasz.getKepMagassag()/2);
    frissit();

    // Létrehoz és elindít egy új szálat,
    // ez a szál hozza létre a pillangókat.
    Thread szal = new Thread(this);
    szal.start();
    kijelzoPanel.jatekKezdet();
}
```

Ha a pillangót elkapták, akkor ezt a törlő metódust hívja meg, a játékidő lejártakor pedig a játékot leállító metódust. Most egyik metódus sem tartalmazza a szálak leállítását, csak láthatatlanná tettük őket, de működnek addig, amíg ténylegesen le nem járnak (azaz a pillangó ki nem repül a felületről).
A szálak leállításának kérdéskörét röviden leírtam a pillango_szalak_leallitasa.pdf fájlban.

```
public void torol(Pillango pillango) {
    pillangok.remove(pillango);
    jatekos.pontotKap();
    kijelzoPanel.pontszamotKiir(jatekos.getPontSzam());
}

private void leallit() {

    pillangok.clear();
    vadasz = null;
    frissit();

    jatekosPanel.eredmenytIr(jatekos);
}
```

Még azt kell megbeszélnünk, hogy egyáltalán hogyan mozog a vadász, és hogy kapja el a pillangókat. Az egérmozdítás (dragged, azaz a gombot nyomva való mozgatás) tényét a játékpanel már közölte a vezérlővel, most csak a vezérlő reagálását kell megírnunk:

```java
/**
 * Ha létezik és a játékfelület fölött van a vadász, akkor
 * az adott pozícióra állítjuk a középpontját, és megvizgáljuk,
 * hogy fogott-e lepkét.
 *
 * @param x
 * @param y
 */
public void arrebbMozgat(int x, int y) {
    if (vadasz != null && 0 <= x && x <= jatekPanel.getWidth()
                                        - Vadasz.getKepSzelesseg()
            && 0 <= y && y <= jatekPanel.getHeight()
                                        - Vadasz.getKepMagassag()) {
        vadasz.setKx(x);
        vadasz.setKy(y);

        // Ez egy kis beégetés, a kép alapján nagyjából itt van
        // a lepkeháló középpontja.
        int haloKozepX = x - Vadasz.getKepSzelesseg() / 4;
        int haloKozepY = y - Vadasz.getKepMagassag() / 4;
        // fogott-e lepkét
        lepkeFogas(haloKozepX, haloKozepY);
    }
}

public void lepkeFogas(int kx, int ky) {
    for (Pillango pillango : pillangok) {
        pillango.megfogtak(kx, ky);
    }
}
```

Ha eddig eljutottunk, akkor már lehet is játszani (nem is lenne szükség a fiktív játékosra), viszont a játékosok egy része hiú, és szeretné elmenteni az eredményét, ezért kell megadni a nevét, és ezt a mentést oldja meg a játékos panel.

De hogyan kérjük be a játékos nevét, és így hogyan indítsuk a játékot? Látszólag nagyon egyszerű a dolog, ténylegesen is ☺, egyetlen apró probléma van, két különböző panelen van a név bekérése és az indító gomb. Természetesen így is meg lehet oldani, ekkor a játék indításakor meg kellene hívni a `JatekosPanel` `getNev()` metódusát, ahol ez a metódus a textfield tartalmát adja vissza. Azt, hogy egyáltalán van-e adat, a hívó osztálynak kell kezelni.

Viszont az is megoldás lehet, és talán kényelmesebbé és logikusabbá teszi a program működését, ha nem a kijelző panelen van az indító gomb, hanem átrakjuk a játékos panelre. (Ez egyetlen mozdulat, le sem kell venni a paneleket a frame-ről, így is frissül majd.)
Ekkor a játékos panelen figyelhetjük, hogy nem üres-e a textfield tartalma. Ha üres, akkor hibaüzenet, ha nem, akkor a begépelt név alapján létrehozható a játékos példány, és ezek után így hívható az indítás:

```
private void btnInditasActionPerformed(java.awt.event.ActionEvent evt) {
    String nev = txtNev.getText();
    if(nev.isEmpty()){
        JOptionPane.showMessageDialog(this, "Nem adott meg nevet");
    }
    else{
        Jatekos jatekos = new Jatekos(nev);
        vezerlo.jatekInditas(jatekos);
    }
}
```

A vezérlésben át kell vennünk ezt a játékost, vagyis

```
this.jatekos = jatekos;
```

de minden más változatlan.

Mindkét megoldás mellett szólhatnak érvek. Ha marad a gomb, akkor akár el is hagyhatjuk a játékos panelt – ekkor persze a név bekérését ki kell szednünk a vezérlésből.
Ha viszont akarjuk használni a neveket, akkor talán szerencsésebb, ha átkerül a gomb a másik panelre.

Az eredmény listába írása szintén egyszerű. Rendezetten szeretnénk, ezért rendezhető listamodellbe rakjuk.
Most úgy oldottam meg, hogy a listában mindenki csak egyszer szerepelhet, és a legfrissebb eredményét lehet olvasni. Módosítsa úgy, hogy mindig a legjobb eredmény látszódjon.

```
private RendezhetoListModel<Jatekos> jatekosModell =
                                        new RendezhetoListModel<>();

public JatekosPanel() {
    initComponents();
    lstJatekosok.setModel(jatekosModell);
}

public void eredmenytIr(Jatekos jatekos) {
    // Ha már szerepel a listában, akkor a legfrissebb
    // eredményét írjuk bele.
    if(jatekosModell.contains(jatekos)){
       jatekosModell.removeElement(jatekos);
    }
    jatekosModell.addElement(jatekos,true);
}
```

Természetesen ahhoz, hogy rendezhető modellbe tudjuk rakni, illetve, hogy működjön a `contains`() metódus, a `Jatekos` osztályt egyrészt `Comparable` típusúvá kell tennünk, másrészt definiálni kell benne az `equals`()+`hashCode`() metódust (csak a név azonosságát írjuk elő).

Már csak a panelek és a vezérlő közötti kapcsolatok kialakítása és a vezérlő kezdő metódusának indítása van hátra.

A panelek a frame-n szerepelnek együtt, tehát itt kell létrehoznunk a vezérlő példányt is, és itt tudjuk egymáshoz rendelni őket. Cél, hogy minél függetlenebbek legyenek az egyes osztályok, ezért csak a szükséges kapcsolatokat építjük ki.
A vezérlőnek mindhárom panellel van dolga, vagyis neki mindhármat ismernie kell.
A játékpanel azt rajzolja, amit a vezérlő mond neki, ezért ismernie kell a vezérlőt. A kijelző panelnek szintén, hiszen az indítás gomb megnyomásakor értesítenie kell a vezérlőt.

a) Ha a gomb marad a kijelző panelen, akkor a játékos panelnek semmi dolga nincs a vezérlővel, nem kell ismernie.

b) Ha a gomb átkerül a játékos panelre, akkor neki is ismernie kell a vezérlőt, vagyis itt is meg kell hívnunk a setVezerlo() metódust. (Ezt most nem írom meg, de ott van üresen a helye.)

A kapcsolatok beállítása után el kell indítani a vezérlő példány `beallitas`() metódusát. Ebben viszont beolvasás szerepel (a képek betöltése), ezért tilos a konstruktorból meghívnunk, csak a frame `main()` metódusából indíthatjuk.

A frame metódusa:

```
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FoFrame().start();
            }
        });
    }

    private void start() {
        setVisible(true);
        Vezerlo vezerlo = new Vezerlo(jatekPanel1, kijelzoPanel1, jatekosPanel1);
        jatekPanel1.setVezerlo(vezerlo);
        kijelzoPanel1.setVezerlo(vezerlo);

        vezerlo.beallitas();
    }
```