

# JavaFx

## pokračovanie

Už vieme (quadterm2):

- kresliť do Canvas, vložiť Canvas->Pane->Scene->Stage,
- simulovať (Thread+Platform.runlater, Timeline, AnimationTimer) ,
- chytat' ActionEvent, KeyEvent a MouseEvent,
- a že uhol dopadu sa rovná uhlu odrazu ☺



Dnes:

- rôzne spôsoby návrhu jednoduchých (pravouhlej) hry,
- aspekt škálovateľnosti,
- perzistencia,
- príklady ex-skúškových príkladov

Zdroj a literatúra:

- [Introduction to Java Programming, !!!!Tenth Edition](#)

Cvičenia: jednoduché aplikácie s GUI:

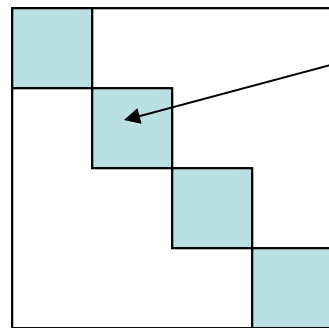
- euro-kalkulačka,
- logické hry: hra15, **pexeso**, ...

# Hracia plocha

hracia plocha je často šachovnica rôznych rozmerov. Ako ju implementujeme:

## 1. jeden veľký canvas v Pane-li:

- musíme riešiť transformáciu pixelových súradníc do súradníc hracej plochy:

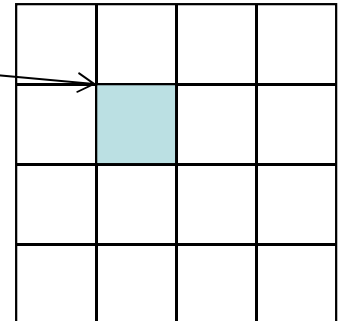


`[event.getX(),event.getY()]->[1,1]`

- a naopak, v metóde `paintMôjCanvas/paintMôjComponent [i,j] -> [pixelX, pixelY]`

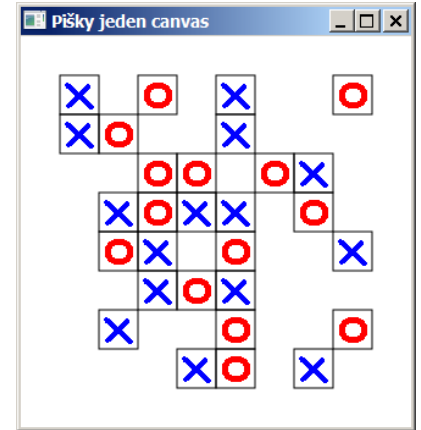
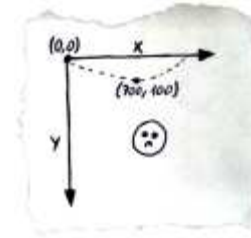
## 2. grid canvasov/Pane-lov:

- každý canvas/panel má svoje súradnice od `[0,0]`
- každý canvas/panel má svoj mouse event handler
- každý canvas panel má svoju metódu `paint/paintMôjCanvas`
- veľkosť gridu upravíme podľa veľkosti obrázkov,  
resp. veľkosť obrázku upravíme podľa veľkosti panelu



## 3. grid buttonov/Button-ov, Button môže mať obrázok ako ikonu

# 1. Riešenie Canvas



```
class Piskyground extends Canvas {  
    Image imageO = new Image("o.gif");           // čítanie obrázku  
    Image imageX = new Image("x.gif");  
    double cellSize = 2+Math.max( // 2+ znamená dva pixle pre orámovanie obrázku  
        Math.max(imageX.getWidth(), imageO.getWidth()), // zoberieme najväčší  
        Math.max(imageX.getHeight(), imageO.getHeight())); // z rozmerov obrázkov  
  
    public Piskyground() {  
        setWidth(SIZE * cellSize);                // veľkosť hracej plochy  
        setHeight(SIZE * cellSize);  
        setOnMouseClicked(event -> {              // mouse event handler pre celú plochu  
            int col = getCol(event.getX());         // transformácia z pixlov na riadok  
            int row = getRow(event.getY());         // stĺpec  
            if (ps.playground[col][row] != 0) return; // logika hry:niekto tam už...  
            ps.playground[col][row]=(ps.nextPlayerIsX) ? 1 : -1; // kto je na ťahu  
            paintCell(col, row);                    // prekresli len kliknuté políčko  
            ps.nextPlayerIsX = !ps.nextPlayerIsX;   // // logika hry:ďalší na ťahu  
        } );  
    }  
}
```

Súbor: [PiskvorkyCanvas.java](#)

# 1. Riešenie Canvas

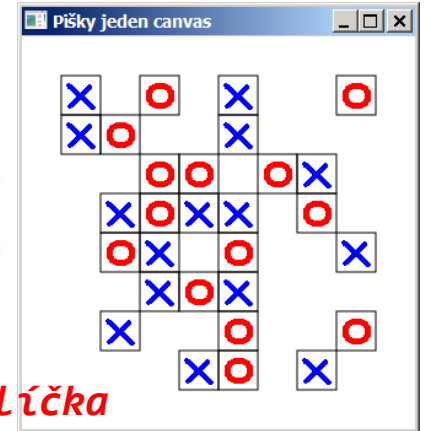
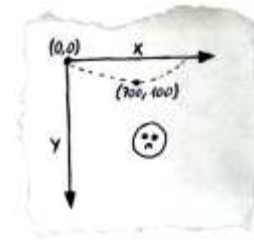
```
class Piskyground extends Canvas {
```

```
...
```

```
    public void paintCell(int col, int row) { // kreslenie políčka
        double px = getPixelX(col); // transformácia row, col
        double py = getPixelY(row); // na pixlové súradnice px, py
        GraphicsContext gc = getGraphicsContext2D(); // do gc kreslíme
        gc.strokeRect(px, py, cellSize, cellSize); // kreslí rámček šírky 1px
        if (ps.playground[col][row] == 1) gc.drawImage(imageX, px + 1, py + 1);
        else
            if (ps.playground[col][row] == -1) gc.drawImage(imageO, px + 1, py + 1);
    }
```

Napriek tomu, že transformácie row, col do pixelových súradníc sú často jednoduché lineárne transformácie (\* / niečo, +- niečo), doprajte si tú abstrakciu a vytiahnite ich do extra metód !!!

```
    private int getRow/Col(double pixel) {
        return (int)(pixel/cellSize);
    }
    private double getPixelX/Y(int i) {
        return i*cellSize;
    }
```

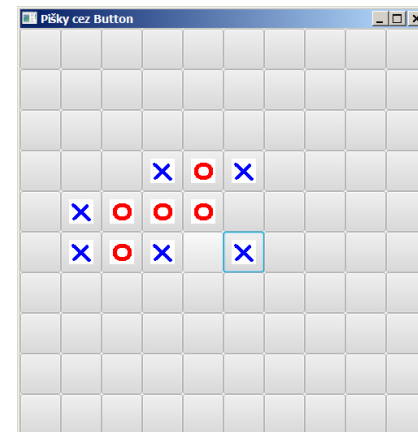


## 2. Riešenie GridPane/Button

Aby ste vedeli uložiť a načítať konfiguráciu hry, reprezentujte ju extra triedou, ktorá je serializovateľná

```
class PiskyState implements Serializable {
    public int[][] playground = new int[SIZE][SIZE];
    public boolean nextPlayerIsX = false; // na ťahu
    public long elapsedTime = 0;          // čas...
                                         // ďalšie veci, čo predstavujú konfiguráciu
}

class Piskyground extends GridPane {
    public Piskyground() {
        for (int i = 0; i < SIZE; i++)
            for (int j = 0; j < SIZE; j++)
                add(new PiskyCell(i, j), i, j);
                                         // pridaj všetky políčka
    }
}
```



Výhody:

- nepotrebujeme transformácie pixel<->cell,
- nikdy si nepomýľte riadok, stĺpec, lebo každé políčko má svoj lokálny event-handler,
- pomerne ľahké riešenie, ak to grafika úlohy dovolí

Súbor: [PiskvorkyGridButton.java](#)

## 2. Riešenie GridPane/Button

```
class PiskyCell extends Button {  
    int i, j;      // políčko si pamätá svoje súradnice  
    public PiskyCell(int i, int j) {  
        this.i = i; this.j = j;      // odtiaľto ...  
        setPreferredSize(50, 50); // vyexperimentovaná veľkosť  
        setOnAction(event -> {  
            if (ps.playground[i][j] != 0) return;  
            if (ps.nextPlayerIsX) {  
                ps.playground[i][j] = 1; // button.setGraphic  
                setGraphic(new ImageView(new Image("x.gif")));  
            } else {  
                ps.playground[i][j] = -1; // ImageView, nie Image  
                setGraphic(new ImageView(new Image("o.gif")));  
            }  
            ps.nextPlayerIsX = !ps.nextPlayerIsX;  
        } );  
    }  
}
```

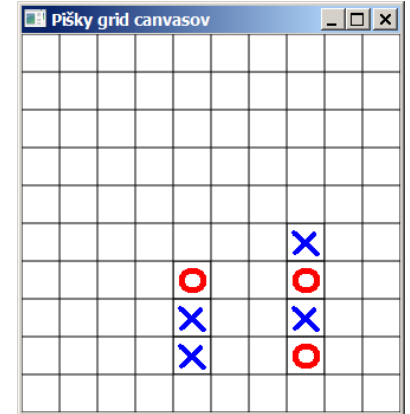
Nevýhody:

- renderovanie gridu nemáte úplne pod kontrolou
- nevieme sa zbaviť škaredého lemu okolo obrázka



Súbor: [PiskvorkyGridButton.java](#)

# 3. Riešenie Grid/Canvas

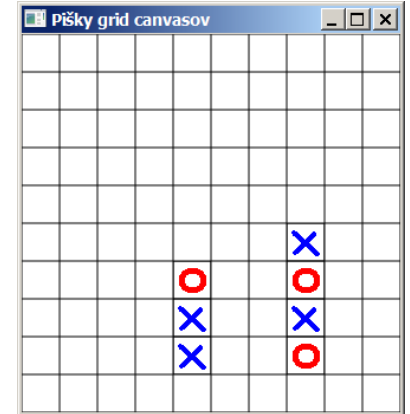


```
class PiskyCell extends Canvas {
    int i, j; // rovnako, políčko si pamätá svoje súradnice
    Image imageO = new Image("o.gif");
    Image imageX = new Image("x.gif");
    double cellSize = 2 + // veľkosť bunky aj s orámovaním
        Math.max(Math.max(imageX.getWidth(), imageO.getWidth()),
            Math.max(imageX.getHeight(), imageO.getHeight()));

    public PiskyCell(int i, int j) {
        this.i = i; this.j = j;
        setWidth(cellSize); setHeight(cellSize); // nastav veľkosť bunky
        setOnMouseClicked(event -> {
            if (ps.playground[i][j] != 0) return; // „Logika“ hry
            ps.playground[i][j] = (ps.nextPlayerIsX)?1:-1;
            paintCell(); // treba ju prekresliť po zmene stavu
            ps.nextPlayerIsX = !ps.nextPlayerIsX;
        });
    }
}
```

# 3. Riešenie Grid/Canvas

```
class PiskyCell extends Canvas {  
    public void paintCell() {           // prekreslenie políčka  
        GraphicsContext gc = getGraphicsContext2D();  
        gc.strokeRect(0, 0, getWidth(), getHeight()); // rámček  
        if (ps.playground[i][j] == 1) gc.drawImage(imageX, 1, 1); // obrázok x,o  
        else if (ps.playground[i][j] == -1) gc.drawImage(imageO, 1,1);  
    } }  
}
```



Vo všetkých troch riešenia sme použili vnorené triedy

```
public class PiskvorkyGridCanvas extends Application {  
    PiskyState ps = new PiskyState();           // konfigurácia hry, vidia všetci  
    public void start(Stage primaryStage) { ... }  
    public static void main(String[] args) { ... }  
    class Piskyground extends GridPane { ... } // vytvorí konfiguráciu  
    class PiskyCell extends Canvas { ... }     // kreslí z konfigurácie  
    public PiskyCell(int i, int j) { ... }  
}  
// akurát PiskyState nie je vnorená, prečo ?  
public class PiskyState implements Serializable { ... }
```

Súbor: [PiskvorkyGridCanvas.java7](#)



# Pexeso

s týmito vedomosťami sme sa pustili do Pexesa,  
a dopadlo to ... neslávne

```
public class Pexeso extends Application {
    State state = new State();
    Playground playground;
    public class Playground extends GridPane {
        public class Cart extends Pane {
        }
    }
}

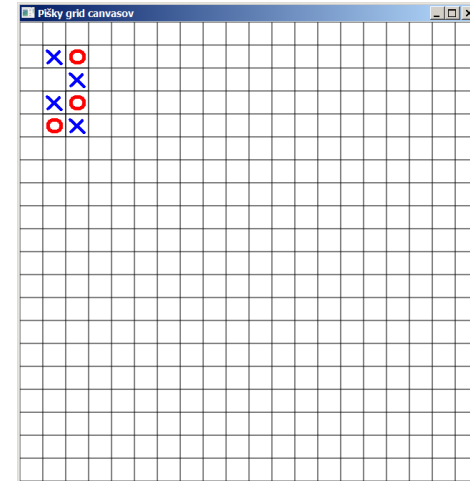
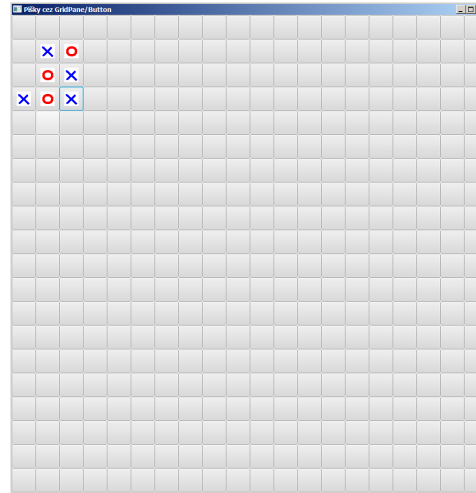
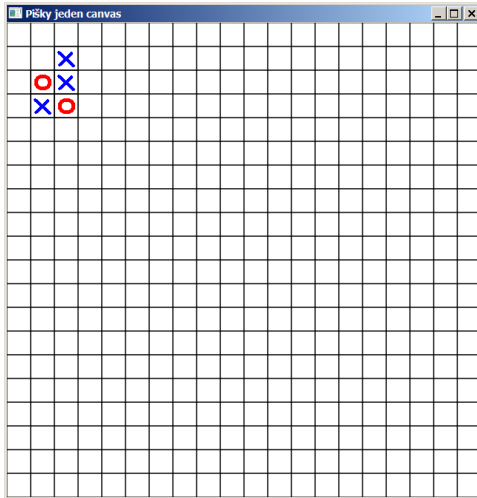
// POZOR, TOTO NEMOŽE BYŤ VNORENÁ TRIEDA, lebo ... ani public
class State implements Serializable {
    private static final long serialVersionUID = 918972645L;
    public class CartObject implements Serializable {
        private static final long serialVersionUID = 911775039L;
        int id;
        boolean visible = false;
        transient // znamená, že nechceme serializovať
            ImageView pikaImage; //toto na cviku navrhoval Peter
    }
}
```

Súbor: [Pexeso.java](#)

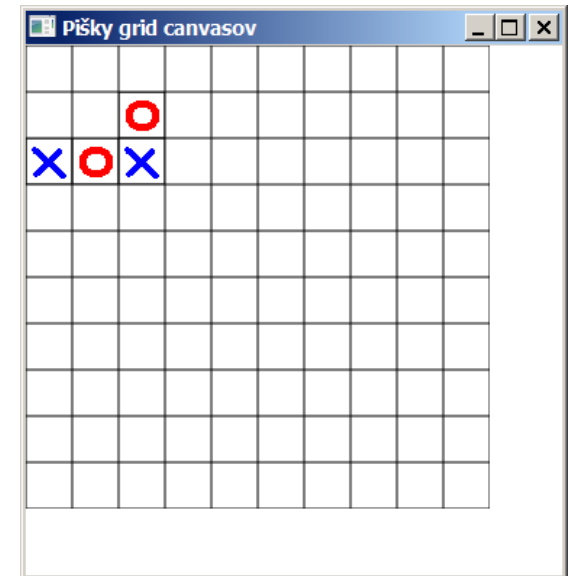
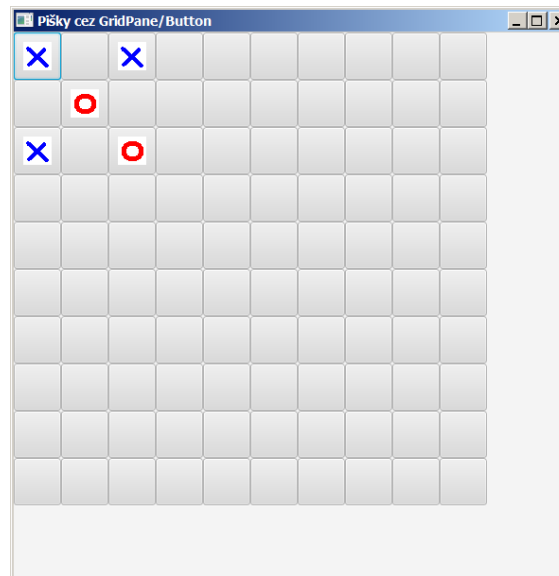
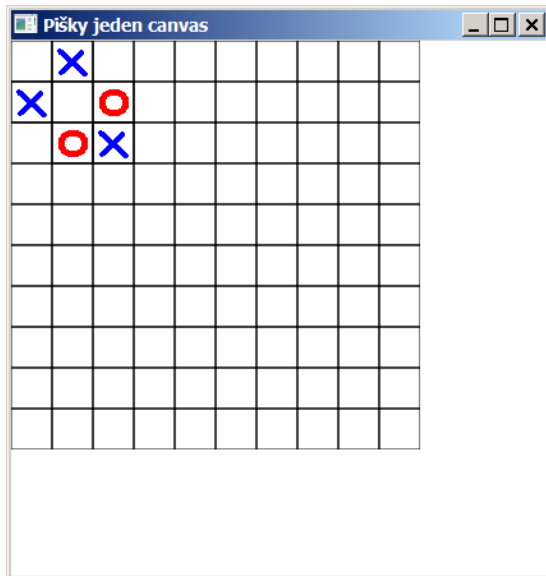


# Škálovateľnosť

- Škálovateľnosť hry (miesto 10x10 chceme hrať 20x20):



- Škálovateľnosť GUI (zmeníme rozmer okna):



# Škálovateľný Canvas

```
final int SIZE = 10;

class Playground extends Canvas {
    public Playground() { // ak sa zmení veľkosť, prekresli celý canvas
        widthProperty().addListener(event -> paint());
        heightProperty().addListener(event -> paint());
    }
    private void paint() {
        double width = getWidth(); // zisti aktuálnu veľkosť, šírku
        double height = getHeight(); // a výšku
        GraphicsContext gc = getGraphicsContext2D(); // kresli pravouhlu mriežku
        gc.clearRect(0, 0, width, height); // ale najprv si to vygumuj
        gc.setStroke(Color.BLACK);
        for(int i = 0; i<SIZE; i++) gc.strokeLine(0, i*height/SIZE, width, i*height/SIZE);
        for(int i = 0; i<SIZE; i++) gc.strokeLine(i*width/SIZE, 0, i*width/SIZE, height);
    }
}

public void start(Stage stage) throws Exception {
    Playground pg= new Playground();
    Pane p = new Pane(pg);
    pg.widthProperty().bind(p.widthProperty()); // pg.width = p.width
    pg.heightProperty().bind(p.heightProperty()); //pg.height=p.height
    stage.setScene(new Scene(p, 400, 400));
```

Súbor: [ResizableCanvas.java](#)

# Bindings

```
DoubleProperty polomer = new SimpleDoubleProperty();
DoubleProperty priemer = new SimpleDoubleProperty();
priemer.bind(polomer.multiply(2)); // priemer = 2*polomer

DoubleProperty obvod = new SimpleDoubleProperty();
obvod.bind(polomer.multiply(2).multiply(Math.PI)); // obvod = 2*PI*polomer

NumberBinding stvorec = Bindings.multiply(polomer, polomer);
DoubleProperty obsah = new SimpleDoubleProperty(); // stvorec=polomer*polomer
obsah.bind(stvorec.multiply(Math.PI)); // obsah = PI*stvorec

// polomer.bind(polomer.divide(2)); // cyklická referencia, to nedá ☹️
for (double r = 0; r < 2; r += 0.5) {
    polomer.set(r);
    // obvod.set(r); // génius nie je!
    System.out.printf(
        "polomer=%6.2f, priemer=%6.2f, obvod=%6.2f, obsah=%6.2f\n",
        polomer.getValue(),
        priemer.getValue(), obvod.getValue(), obsah.getValue());
}
```

Súbor: [RealBindings.java](#)

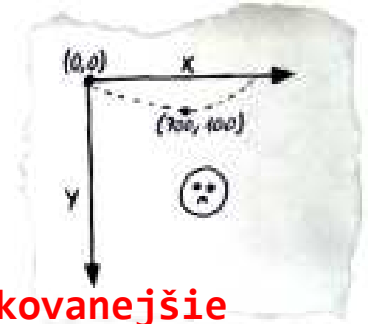
# 1. Riešenie škálovateľné jeden Canvas

```
Piskyground pg = new Piskyground(); // one big Canvas
Scene scene = new Scene(new Group(pg), 500, 500); // najaká iníciaľna veľkosť
pg.widthProperty().bind(scene.widthProperty()); // pg.width = scene.width
pg.heightProperty().bind(scene.heightProperty()); // pg.height = scene.height
pg.paintAll(); // inak by sa nič nevykreslilo

scene.widthProperty().addListener(event -> pg.paintAll()); // changeListener
scene.heightProperty().addListener(new ChangeListener<Number>() { //full verzia:
    @Override
    public void changed(ObservableValue<? extends Number> observableValue,
        Number oldSceneHeight, Number newSceneHeight) {
        System.out.println("Height: " + newSceneHeight);
        pg.paintAll();
    }
});

primaryStage.setTitle("Resizable Pišky jeden canvas");
...
```

# Transformácie



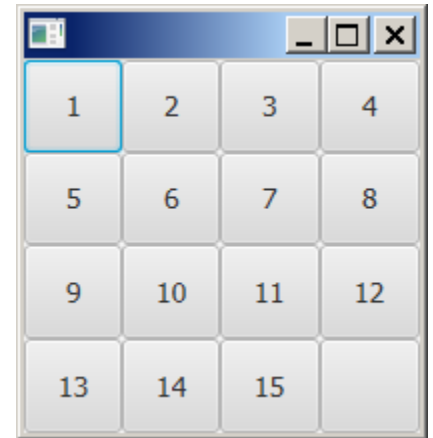
```
class Piskyground extends Canvas { // sú komplikovanejšie
    // a už záleží na x,y lebo plocha môže byť obĺžnik
    private double cellWidth() { return getWidth()/SIZE; }
    private double cellHeight() { return getHeight()/SIZE; }
    private int getRow(double pixelY) { return (int) (pixelY / cellHeight()); }
    private int getCol(double pixelX) { return (int) (pixelX / cellWidth()); }
    private double getPixelX(int row) { return row * cellHeight(); }
    private double getPixelY(int col) { return col * cellWidth(); }
}
```

```
public void paintCell(int i, int j) {
    Image imageO = // obrázok danej šírky a výšky
        new Image("o.gif", cellWidth()-2, cellHeight()-2, false, false);
    Image imageX =
        new Image("x.gif", cellWidth()-2, cellHeight()-2, false, false);
    . . . .
```

# Hra 15

## BoundsProperty listener

```
public class Hra15 extends Application {  
    final int SIZE = 4;    final int COLS = SIZE;    final int ROWS = SIZE;  
    @Override  
    public void start(final Stage primaryStage) throws Exception {  
        GridPane gp = new GridPane();  
        for (int i = 0; i < 16; i++) {           // vytvorí hraciu plochu  
            Button button = (i == 15) ? new Button("") : new Button("" + (i + 1));  
            gp.add(button, i % COLS, i / COLS);    // mod, div=súradnice políčka i  
        }  
        gp.layoutBoundsProperty().addListener( // ak sa zmenia rozmery gp  
            (observable, oldBounds, newBounds) -> {  
                double cellHeight = newBounds.getHeight() / ROWS;  
                double cellWidth = newBounds.getWidth() / COLS;  
                for (final Node child : gp.getChildren()) {  
                    final Control tile = (Control) child;  
                    tile.setPrefSize(cellWidth, cellHeight);  
                }           // prekreslí všetky Node v gp  
            }  
        ));
```



Súbor: [Hra15.java](#)

# 2. Riešenie škálovateľné

## fitWidth/HeightProperty

[Súbor: PiskvorkyGridButtonResizable.java](#)

@Override

```
public void start(Stage primaryStage) {
    Piskyground pg = new Piskyground();
    pg.layoutBoundsProperty().addListener((observable, old, newBounds) -> {
        for (final Node child : pg.getChildren()) { // ak sa zmení rozmer pg
            final Control tile = (Control) child; // zmeň veľkosti buniek
            tile.setPrefSize(newBounds.getWidth() / SIZE,
                            newBounds.getHeight() / SIZE);
        }
    });
}

class PiskyCell extends Button {
    ImageView imageO = new ImageView(new Image("o.gif"));
    ImageView imageX = new ImageView(new Image("x.gif"));
    public PiskyCell(int i, int j) {
        setMinSize(50, 50); // menej nedovolí
        imageX.fitWidthProperty().bind(widthProperty()); // X.width = this.width
        imageX.fitHeightProperty().bind(heightProperty()); // X.height = this.height
        imageO.fitWidthProperty().bind(widthProperty()); // O.width = this.width
        imageO.fitHeightProperty().bind(heightProperty()); // O.height = this.height
    }
}
```



# 3. Riešenie škálovateľné

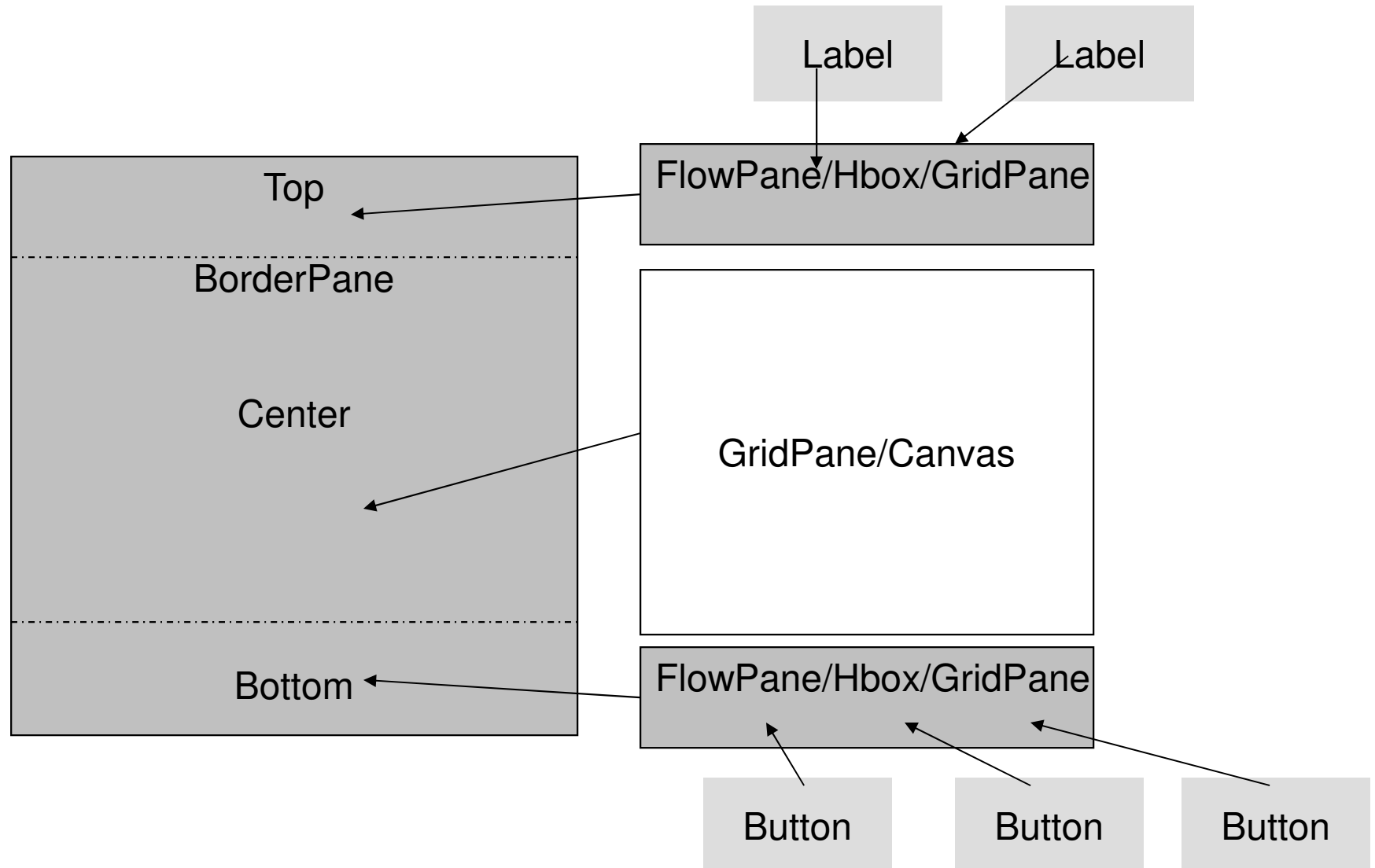
Súbor: [PiskvorkyCanvasResizable.java](#)

```
public class PiskvorkyGridCanvasResizable extends Application {
    pg = new Piskyground();
    scene.widthProperty().addListener((observableValue, old, newSceneWidth)->{
        pg.prefWidth((double) newSceneWidth);
    });
    pg.paint();
    }); // to isté pre height

class Piskyground extends GridPane {
    public Piskyground() {
        for (int i = 0; i < SIZE; i++) for (int j = 0; j < SIZE; j++) {
            PiskyCell pc = canvasGrid[i][j] = new PiskyCell(i, j);
            add(pc, j, i);
            pc.widthProperty().bind(widthProperty().divide(SIZE)); // tiež height
        }
    }

class PiskyCell extends Canvas {
    public void paintCell() {
        GraphicsContext gc = getGraphicsContext2D();
        Image imageX=new Image("x.gif",getWidth()-2,getHeight()-2,false,false);
        Image imageO=new Image("o.gif",getWidth()-2,getHeight()-2,false,false);
    }
}
```

# Scéna hry



# Layout

```
Piskyground pg = new Piskyground();           // pôvodná hracia plocha
BorderPane bp = new BorderPane();               // vonkajší rámec
bp.setCenter(pg);

HBox labelPane = new HBox(                      // vrchný panel, FlowPane, GridPane, ...
    new Label("Score:"), lbScore = new Label("0"),
    new Label("Elapsed time:"), lbTime = new Label("0"),
    new Label("Next:"), lbOnMove = new Label("o"));
labelPane.setSpacing(20);                       // hrubý layout, s tým sa dá vyhrať...
lbScore.setFont(Font.font(18)); ...
bp.setTop(labelPane);                           // umiestnime na vrch

HBox buttonPane = new HBox(                     // spodný panel plný tlačidiel, gombíkov
    btnLoad = new Button("Load"), btnSave = new Button("Save"),
    btnQuit = new Button("Quit"));
buttonPane.setSpacing(50);
bp.setBottom(buttonPane);                       // umiestnime na spodok
```

# Control

```
btnQuit.setOnAction(event -> System.exit(0));
```

```
btnLoad.setOnAction(event -> {                                // načítanie konfigurácie
    try {
        ObjectInputStream is=new ObjectInputStream(new FileInputStream("p.cfg"));
        ps = (PiskyState) is.readObject();
        is.close();
        pg.paintAll();                                          // prekresli scénu, inak sa zmení len stav
    } catch (Exception e) { e.printStackTrace();}
} );
```

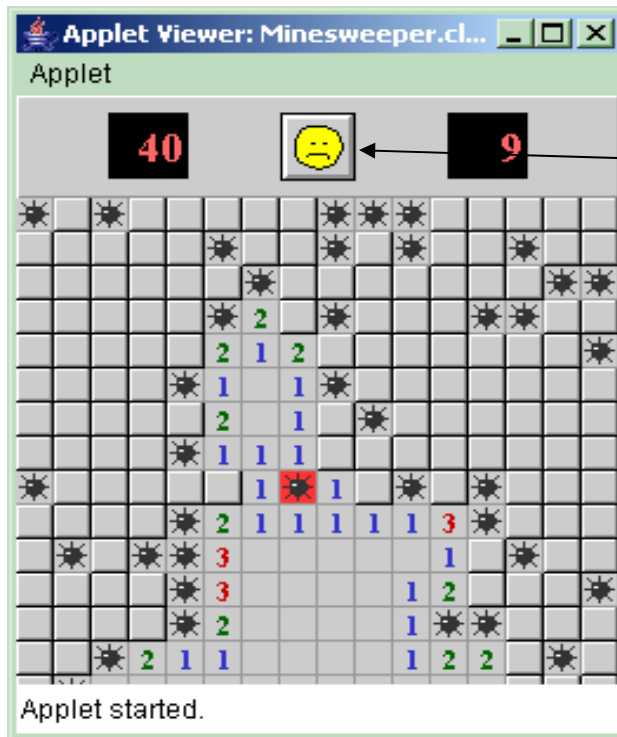
```
btnSave.setOnAction(event -> {                                // uloženie konfigurácie
    try {
        ObjectOutputStream fs=new ObjectOutputStream(new FileOutputStream("p.cfg"));
        fs.writeObject(ps);
        fs.close();
    } catch (Exception e) { e.printStackTrace(); }
} ); // pozor ! Väčšina javafx objektov nie je serializovateľná, ani Image...
```

# Timer

```
Timeline tl = new Timeline(1000);           // počítame spotrebovaný čas
tl.setCycleCount(Timeline.INDEFINITE);
tl.getKeyFrames().add(new KeyFrame(Duration.seconds(1), event -> {
    ps.elapsedTime++;
    Platform.runLater(new Runnable() {
        @Override
        public void run() {
            lbTime.setText(""+ps.elapsedTime); // a prekreslujeme do info políčka
        }
    });
}));
tl.play();
```

# Case 1

- Hra minesweeper (riešenie:A.D.Birrell). Celá hra je kreslená graficky, pre zaujímavosť si pozrite, ako maľuje usmievačika/smutňáčka, resp. do dokonalosti dovedené maľovanie bomby s tieňom ☺



```
private Image initOneSmiley(int theSadness) {
    Image off = createImage(faceSize, faceSize);
    Graphics g = off.getGraphics();
    g.setColor(Color.black);
    g.fillRect(0, 0, faceSize, faceSize);
    g.setColor(baseColor);
    g.fill3DRect(1, 1, faceSize-2, faceSize-2, true);
    g.fill3DRect(2, 2, faceSize-4, faceSize-4, true);
    g.setColor(Color.yellow);
    g.fillOval(6, 6, faceSize-12, faceSize-12);
    g.setColor(Color.black);
    g.drawOval(6, 6, faceSize-12, faceSize-12);
    if (theSadness==sad) {
        g.drawArc(10, faceSize-13,
                  faceSize-20, faceSize-20, 135, -100);
    } else if (theSadness==happy) {
        g.drawArc(10, 10,
                  faceSize-20, faceSize-20, -35, -100);
    } else {
        g.fillRect(12, faceSize-12, faceSize-23, 1);
    }
    g.fillOval(13, 13, 2, 2);
    g.fillOval(faceSize-12-2, 13, 2, 2);
    return off;
}
```

<http://birrell.org/andrew/minesweeper/Minesweeper.java>

# Case 2

- grafické dopracovanie projektu je dôležité, nepodceňujte ho !
- získa vám to body a nie je to v Jave ☺



Zdroj: Lukáš Zdechovan

# Grafická príprava projektu

## Ako hrať

Na začiatku hry máte k dispozícii **10€** a po roztočení valcov je automaticky odpočítaný vklad **0,50€**.

Valce roztočíte stisknutím klávesy **Enter** alebo **Medzerník**, prípadne stlačením tlačidla **"Roztoč"**.

Hra končí, keď má hráč na konte menej ako **0,50€** a teda nemá už čo vložiť.

Toto okno zavriete kliknutím na ľubovoľné miesto v hre.

## Výhry

2x rovnaká minca =  
**2x** hodnota euro mince  
3x rovnaká minca =  
**8x** hodnota euro mince

## O hre

Hru vytvoril ako projekt k predmetu **Algoritmy** v rámci štúdia aplikovanej informatiky na VŠB – TU Ostrava.

Pravidlá

Roztoč

Pravidlá

Roztoč





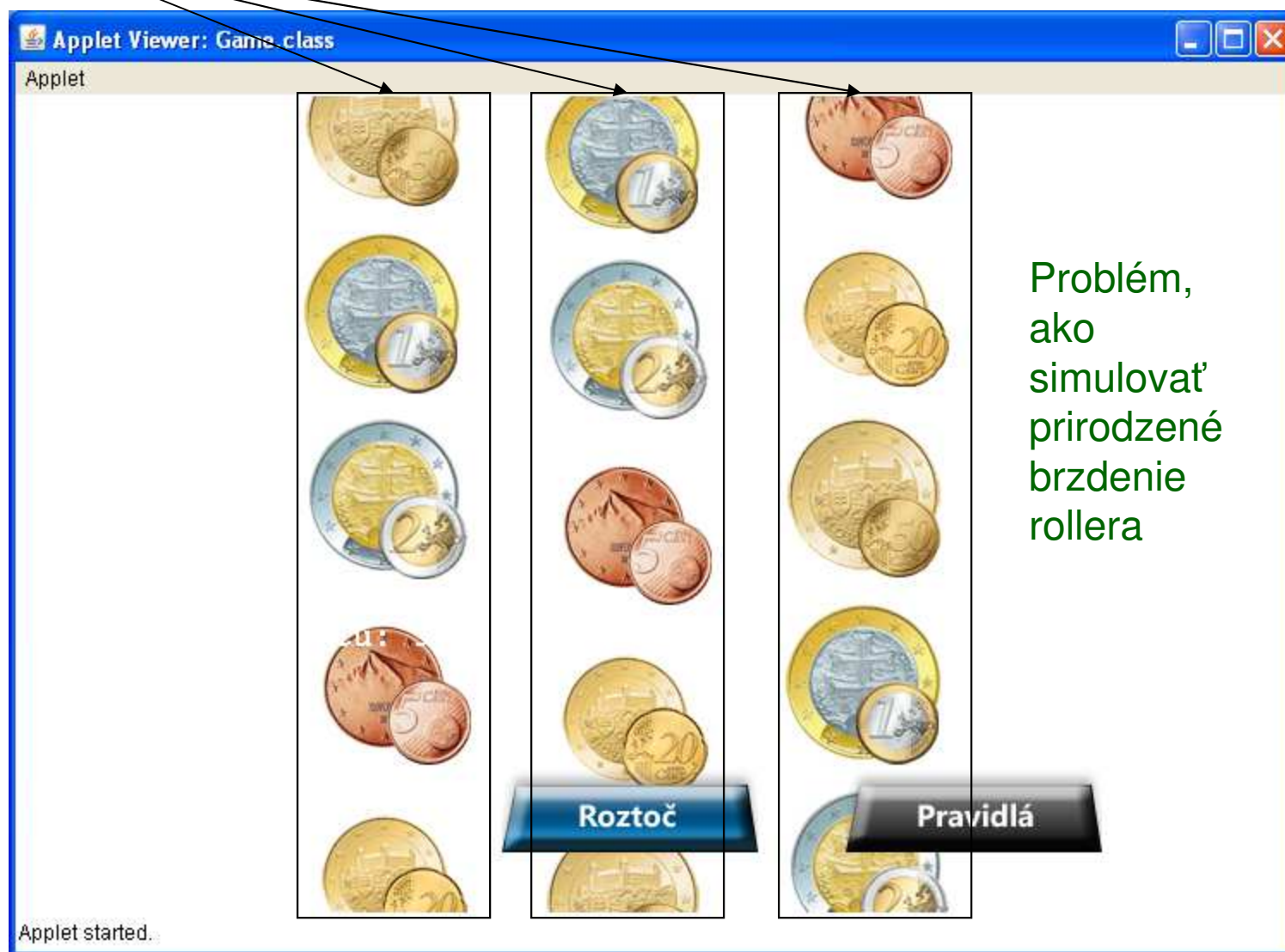
# Spracovanie v Java



Zdroj: Lukáš Zdechovan

Čo roller, to thread

# Simulácia



# Case 3

- zadanie Plumber z predtermínu 2008:
- tri vzorové skúšky (zadania) visia na stránke predmetu
- príklad ilustruje štruktúru skúšky:
  - **čítanie konfigurácie** hry súboru a vykreslenie plochy, konštrukcia scény,  
8  
4  
12345623  
34613532  
35216311  
23654545
  - **ošetrenie udalostí** a rozpohybovanie scény v intenciách pravidiel danej hry,
  - **počítanie a zobrazenie** krokov, životov, časomiera, zistenie, či v danej konfigurácii už sme boli a pod,
  - **škálovateľnosť** hracej plochy,
  - **load a save** konfigurácie (serializácia),
  - **algoritmus** (napr. kam dotečie voda - hľadanie cesty v grafe (labyrinte), analýza víťaznej konfigurácie, ...)

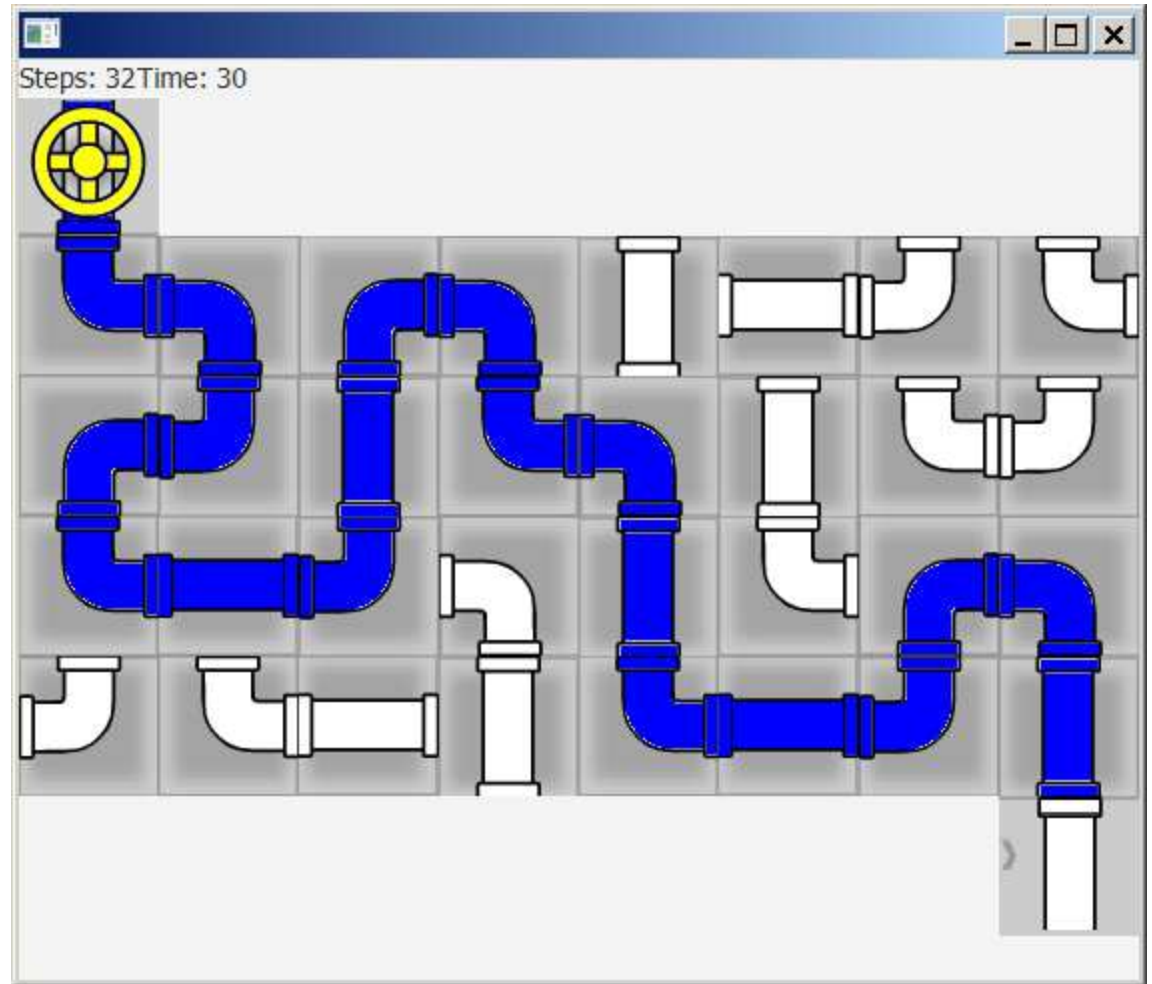
# Plumber (inštalatér)

## Oddel'te GUI

- kreslenie objektov,
- komponentov,
- appletu

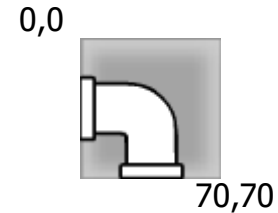
## od logiky hry

- analýza ťahov,
- víťazná konfigurácia,
- zacyklenie, ...



- Plumber – BorderPane/GridPane/Canvas,
- PlumberCanvas – Mouse Event Handler, kreslenie rúr .png,
- PlumberThread - časomiera,

# Plumber



- čítanie obrázkov:

```
for (int i = 1; i <= 8; i++) {  
    img[i] = new Image("plumber" + i + ".png");  
    img_blue[i] = new Image("plumber" + i + "_blue.png");  
    – ak vám nekreslí obrázok, pravdepodobne ste ho nenačítali správne,  
    – najčastejšie nie je v správnom adresári
```

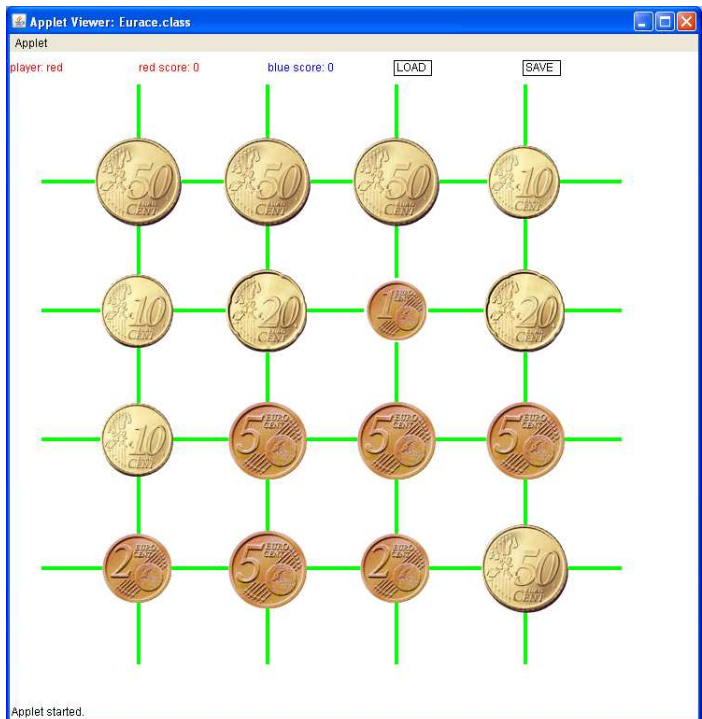
- čítanie vstupnej konfigurácie

```
try {  
    BufferedReader br =  
        new BufferedReader(new FileReader(new File("Plumber.txt")));  
    ... // čítanie textového súboru  
} catch (Exception E) {  
    System.out.println("file does not exist");  
}  
  
– nezanedbajte výnimky,  
– píšete na konzolu, čo čítate, kontrolné pomocné výpisy vás nijako nehandicapujú,  
– ak čítate vstup po znakoch (celé zle), nezabudnite, že riadok končí znakmi 13, 10,  
– rozdiel medzi cifrou a jej ascii kódom je 48, úplne zle, ...
```

- uloženie konfigurácie počas hry

```
– najjednoduchšie pomocou serializácie (pozri prednášku java.io)  
– neserializujte celú aplikáciu, ale len triedu popisujúcu konfiguráciu hry – PiškyState..
```

Súbor: [Plumber.java](#)



# Škálovanie

Naprogramujte mriežku škálovateľnú od rozmeru okna (štvorcová mriežka sa rozťahuje podľa veľkosti okna, v ktorom sa nachádza, NIE KONŠTANTA V PROGRAME)



```
private static int dist = 120;
private static int offset = dist;
```

// počiatočné nastavenia

```
public void init() {
    setSize(offset+N*dist, offset+N*dist); // originálna veľkosť hracej pl.
```

```
public void paint() {
    // nastavenia podľa aktuálnej
    // veľkosti okna
```

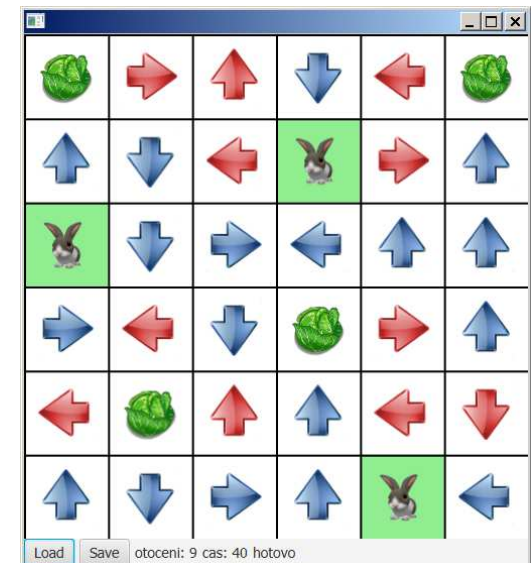
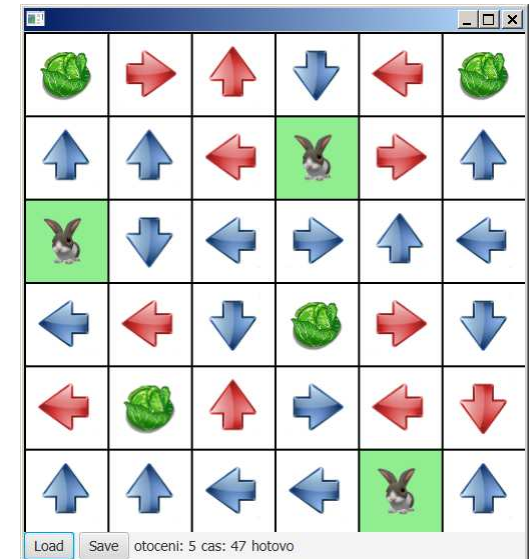
```
offset = dist = (Math.min(getHeight(),getWidth()))/(N+1);
```



# Niečo novšie

## (Zajace a kapusty)

- Naprogramujte hru pre jedného hráča Zajace a kapusty. Hrá sa na štvorcovej mriežke NxN štvorcov. V niektorých štvorcoch sa nachádzajú zajace, v niektorých kapusty a v ostatných šípky smerujúce jedným zo štyroch smerov. Počet zajacov, kapusty a šípok môže byť vzhľadom na N rôzny. Niektoré šípky sú červené - tie smerujú stále rovnakým smerom, niektoré sú modré a tie sa pri kliknutí myšou otáčajú o 90°. Príklad hernej situácie je na obrázku:
- Cieľom hráča je pootáčať modré šípky tak, aby sa všetky zajace mohli podľa šípok dostať ku kapuste. Keď zajac stúpi na políčko, kde je šípka, musí pokračovať smerom podľa šípky. Ak narazí na okraj poľa, alebo sa medzi nejakými šípkami zacyklí, ku kapuste sa nedostal. Začiatočná konfigurácia hry je uložená v súbore ...

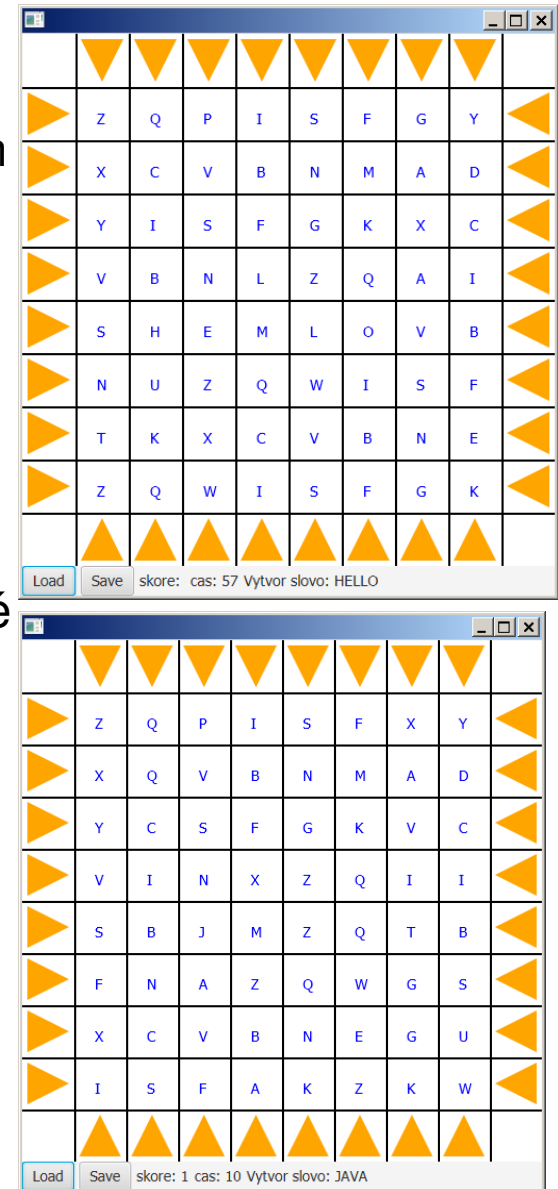


# Niečo novšie

## (Písmenkovica)

- V hre Písmenkovica sú v štvorcovej mriežke rozmiestnené písmená anglickej abecedy. Na okrajoch všetkých strán štvorca sú šípky. Ich stlačením dojde k otočeniu riadka alebo stĺpca o jedno písmenko podľa smeru šípky. Niekde v okne je zobrazené slovo, ktoré treba zo susedných písmen v mriežke vytvoriť: buď vodorovne zľava-doprava, zvislo zhora-nadol, šikmo nadol vpravo, alebo šikmo nahor vpravo. Ak sa to hráčovi podarí, písmená vytvoreného slova zmiznú a sú nahradené za ďalšie. Hráč tým získava bod, cieľové slovo sa zmení a hra pokračuje ďalej. Na každé slovo má 60 sekúnd času, ktoré sa mu odpočítavajú a zostávajúci čas sa zobrazuje. Ak to nestihne, hra končí. Tlačidlami Save/Load uloží/načíta aktuálny stav hry, pričom z načítaného stavu môže pokračovať v hre ďalej. Začiatočná situácia hry, cieľové slová a písmená, ktoré postupne nahrádzajú písmená z vytvorených slov, sú uložené v súbore a na začiatku hry sa z neho načítajú. Formát súboru je nasledujúci

...

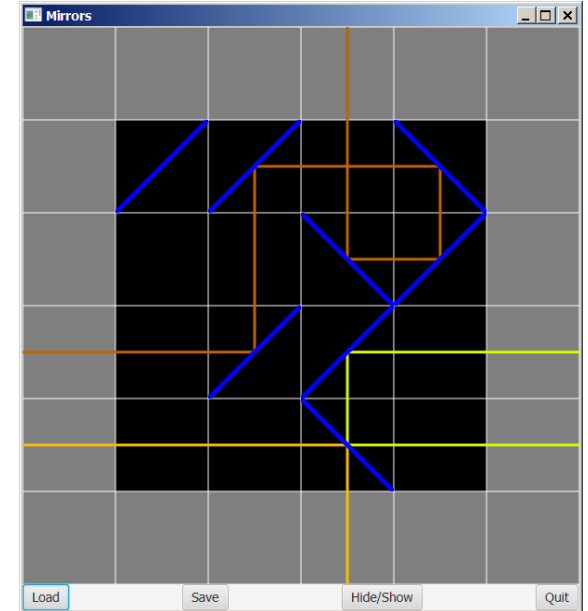




# Niečo novšie

## (Zrkadlová sieň)

- V štvorcovej sále s rozmermi  $N \times N$  sú v niektorých políčkach umiestnené diagonálne zrkadlá, v ilustráciach sú zobrazené modrou farbou. Môžu byť dvoch typov, / alebo \. Na kraji štvorcovej sály sú políčka, ktoré obsahujú zdroje svetla rôznych farieb. Krajné ľavé a pravé políčka ( $N+1$ ) obsahujú vodorovný zdroj svetla, krajné horné a dolné políčka ( $N+1$ ) obsahujú zvislý zdroj svetla. Rožné políčka (4) nemajú žiadnu funkciu. Pre jednoduchosť znázornenia scény plochu kreslíme do štvorcovej mriežky s rozmermi  $(N+2) \times (N+2)$ .



Ak zapneme svetelný zdroj, vodorovný či zvislý, svetlo sa začne šíriť daným smerom cez hraciu plochu. Ak je políčko prázdne, prejde ním. Ak je v ňom diagonálne zrkadlo, odrazí sa od neho presne v duchu príslovia: uhol odrazu je uhol dopadu. Samozrejme, keďže ide o diagonálne zrkadlá, tak tento uhol môže byť len 45 stupňov. Pri rôznych polohách zrkadiel môžu vzniknúť 4 situácie (premýšľajte si...). Niektorým políčkom lúč opustí hraciu plochu, vy znázorňujete jeho cestu. Svetelné zdroje sú rôznych farieb, a farieb máte dosť (stačí si ich nejakým spôsobom vygenerovať).

# Niečo novšie

## (Atomix)

- V hre Atomix sa z atómov konštruujú molekuly rozličných zlúčenín. Každý atóm má naznačený smer väzby a počas hry sa nedokáže otočiť - väzba smeruje stále tým istým smerom. V našej verzii hry sa zameriame iba na molekulu vody. Po kliknutí myšou na niektorý atóm sa tento atóm zvýrazní. Druhé kliknutie na voľné políčko v prázdnej uličke, ktorá vychádza od atómu jedným zo štyroch smerov, atóm uvedie do pohybu. Atóm sa však zastaví, až keď narazí do steny, alebo do iného atómu. Potom je možné kliknúť na nejaký atóm znova. V prípade, že sa podarí vytvoriť molekulu vody, t.j. vedľa seba sa bude vodorovne alebo zvislo nachádzať atóm vodíka, kyslíka a zasa vodíka a budú previazané vzájomnými väzbami, hráč level splnil a môže postúpiť do ďalšieho levelu.

