JavaFx

pokraļovanie

Už vieme (quadterm2):

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

Dnes:

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

Zdroj a literatúra:

Å **Introduction to Java Programming, !!!!Tenth Edition**

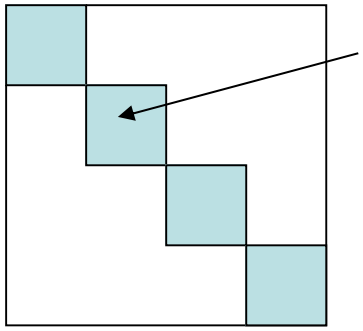
Cvičenia: jednoduché aplikácie s GUI:

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

Hracia plocha

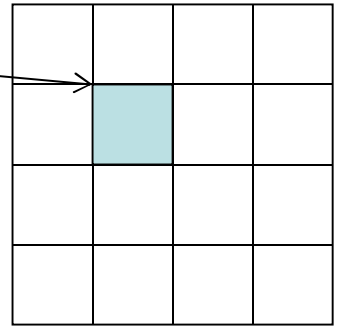
hracia plocha je často šachovnica rôznych rozmerov. Ako ju implementujeme:

1. jeden veľký canvas v Pane-li:
   - musíme riešiť transformáciu pixelových súradníc do súradníc hracej plochy:



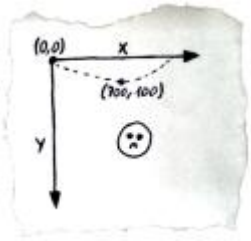
[event.getX(),event.getY()]->[1,1]

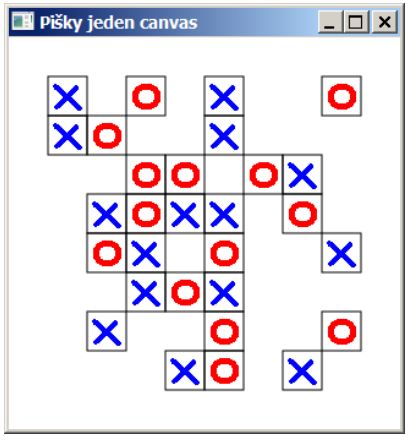
   - a naopak, v metóde paintMôjCanvas/paintMôjComponent [i,j] -> [pixelX, pixelY]
2. grid canvasov/Pane-lov:
   - každý canvas/panel má svoje súradnice od [0,0]
   - každý canvas/panel má svoj mouse event handler
   - každý canvas panel má svoju metódu paint/paintMôjCanvas
   - veľkosť gridu upravíme podľa veľkosti obrázkov,
     resp. veľkosť obrázku upravíme podľa veľkosti panelu



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

1.Rieġenie Canvas

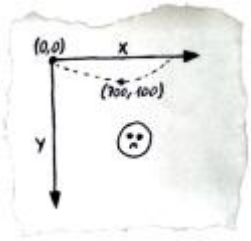


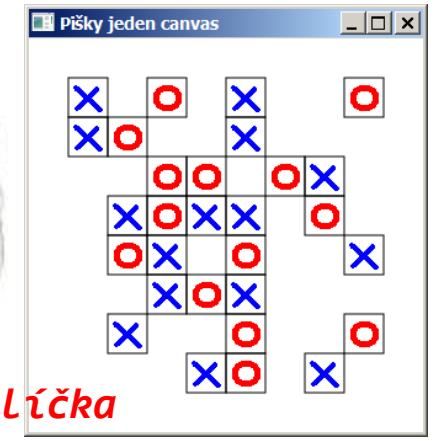



```
class Piskyground extends Canvas {
  Image imageO = new Image("o.gif");       // čítanie obrázku
  Image imageX = new Image("x.gif");
  double cellSize = 2+Math.max(  // 2+ znamená dva pixle pre orámovanie obrázku
      Math.max(imageX.getWidth(), imageO.getWidth()), // zoberieme najväčší
      Math.max(imageX.getHeight(), 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.Rieġenie Canvas





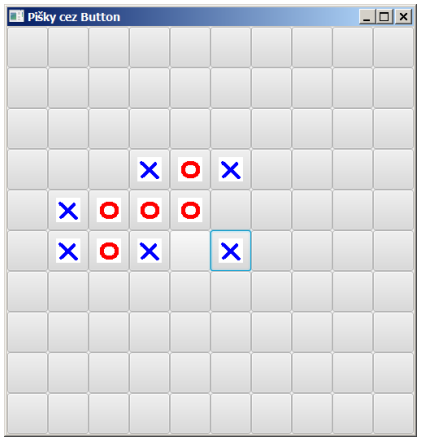

```
class Piskyground extends Canvas {
. . .
  public void paintCell(int col, int row) { // kreslenie políčka
   double px = getPixelX(col);    // transformácia row, col
   double py = getPixelY(row);    // na pixlové súradnice px, py
   GraphicsContext gc = getGraphicsContext2D();    // do gc kreslíme
   gc.strokeRect(px, py, cellSize, cellSize);      // kresli 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, ġe transformácie row, col do pixelových súradníc sú ļ asto jednoduché lineárne transformácie (*/ nieļ o, +- nieļ o), 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;
  }
```

Súbor: PiskvorkyCanvas.java

2.Rieġenie GridPane/Button



Aby ste vedeli uloġiŠa naļ ítaŠkonfiguráciu hry, reprezentujte ju extra triedou, ktorá je serializovateČná

```
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 prestavujú 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 nepomýlite riadok, stĎec, lebo kaġdé políļko má svôj lokálny event-handler,
- pomerne ĎahkéŠ rieġenie, ak to grafika úlohy dovolí

Súbor: PiskvorkyGridButton.java

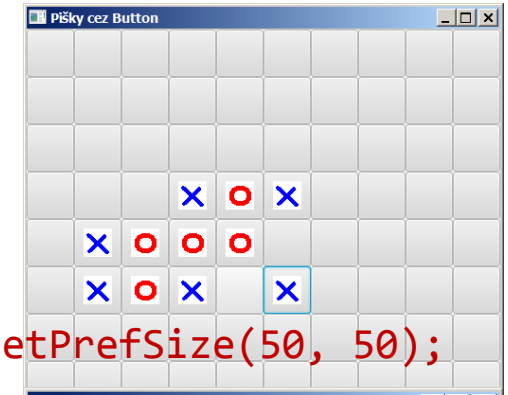
2.Rieġenie

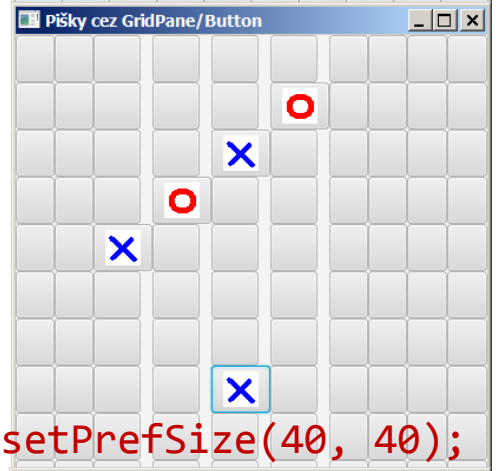
GridPane/Button

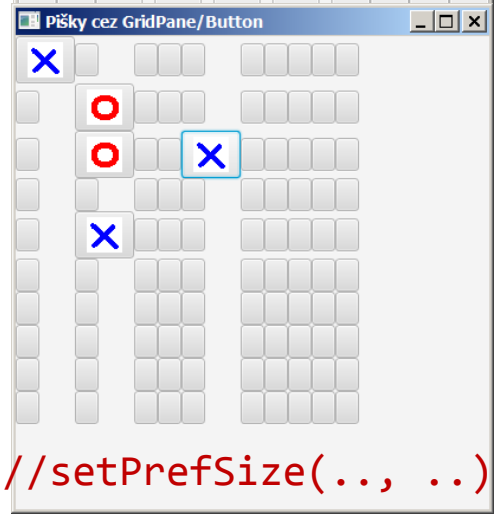
```
class PiskyCell extends Button {
  int i, j;      // políčko si pamätá svoje súradnice
  public PiskyCell(int i, int j) {
    this.i = i; this.j = j;       // odtiaľto ...
    setPrefSize(50, 50); // vyexperimentovaná veľkosť
    setOnAction(event -> {
     if (ps.playground[i][j] != 0) return;
     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 sivého lemu okolo obrázka

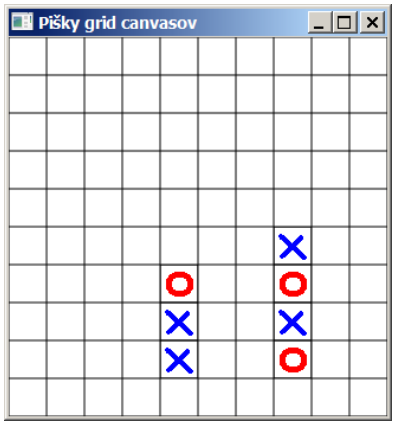






Súbor: PiskvorkyGridButton.java

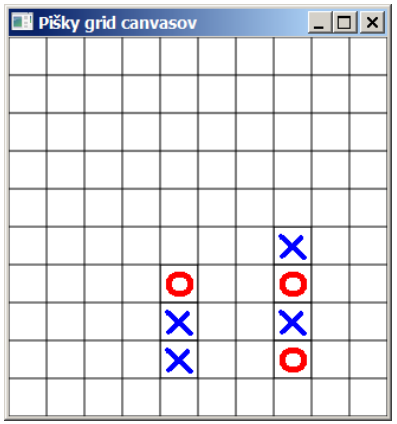
3.Rieġenie Grid/Canvas




```
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;
   });
  }
```

Súbor: PiskvorkyGridCanvas.java

3.Rieġenie 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);
} }
```

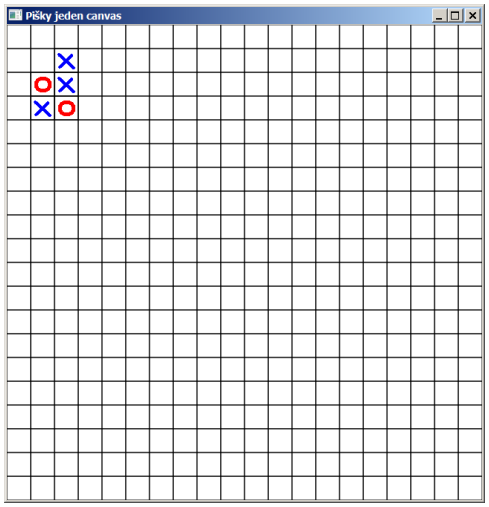
Vo vġetkých troch rieġenia sme pouġili vnorené triedy

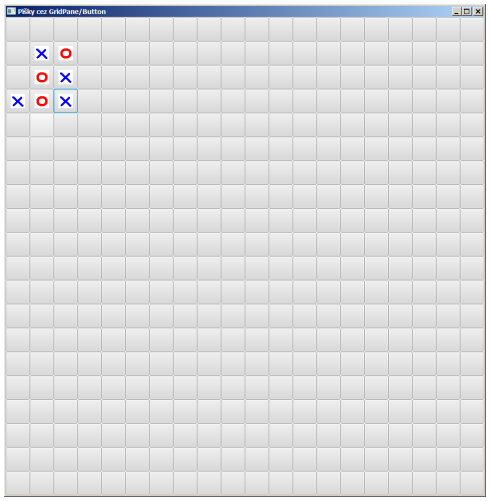
```
public class PiskvorkyGridCanvas extends Application {
    PiskyState ps = new PiskyState();      // konfigurácia hry, vidia všetci
    public void start(Stage primaryStage) { ... }
    public static void main(String[] args) { ... }
  class Piskyground extends GridPane { ... }  // vytvorí konfiguráciu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

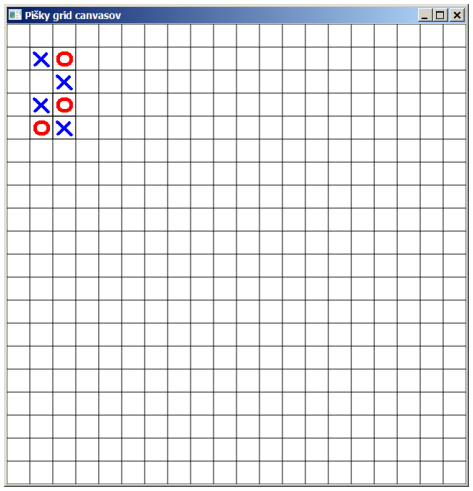
Súbor: PiskvorkyGridCanvas.java

ĠkálovateĎosŠ

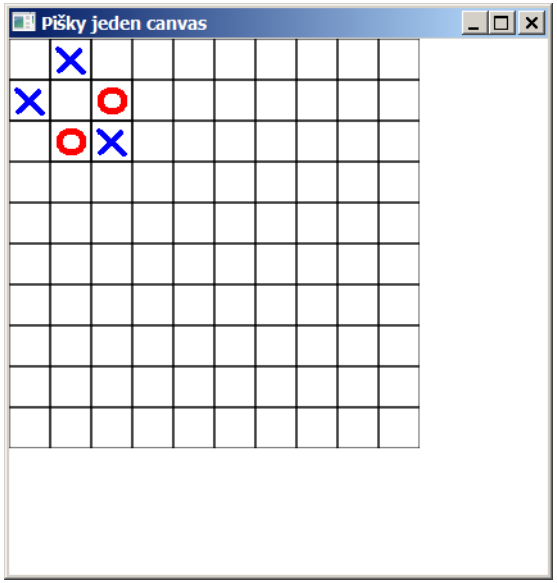
- ĠkálovateĎosŠhry (miesto 10x10 chceme hraŠ20x20):

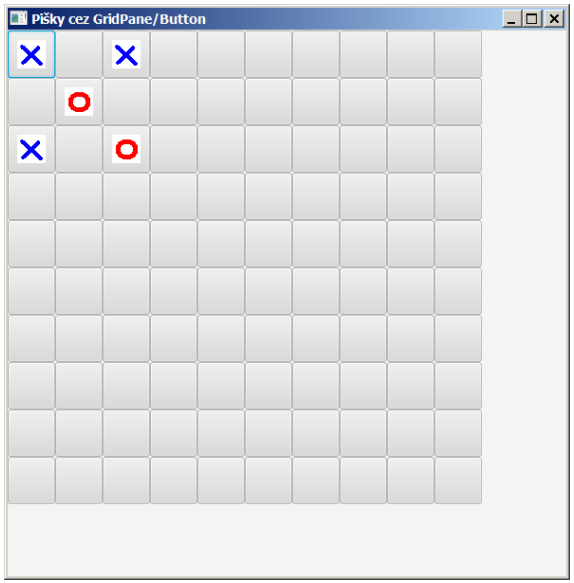


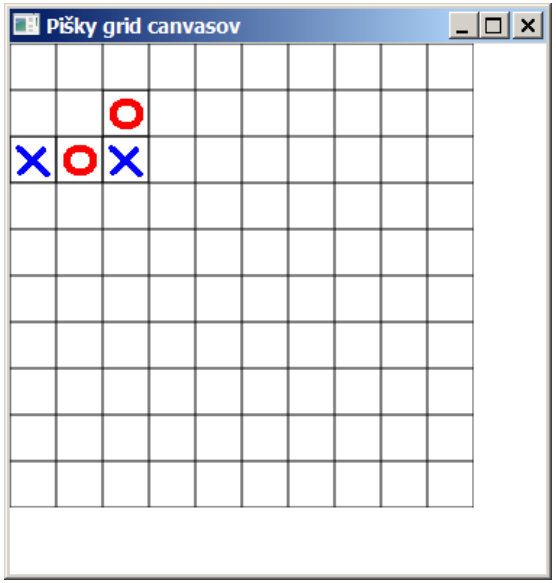




- ĠkálovateĎosŠGUI (zmeníme rozmer okna):







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 pravoúhlu 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());
}
```

```
polomer=  0,00, priemer=  0,00, obvod=  0,00, obsah=  0,00
polomer=  0,50, priemer=  1,00, obvod=  3,14, obsah=  0,79
polomer=  1,00, priemer=  2,00, obvod=  6,28, obsah=  3,14
polomer=  1,50, priemer=  3,00, obvod=  9,42, obsah=  7,07
```

Súbor: RealBindings.java

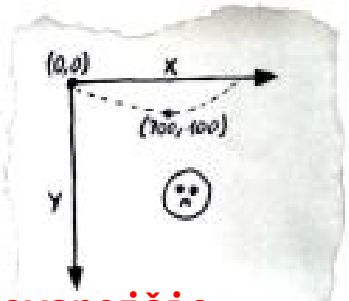
1. Riešenie s jedným veľkým plátnom

jeden Canvas

```
Piskyground pg = new Piskyground();                  // one big Canvas
Scene scene = new Scene(new Group(pg), 500, 500); // najaká iniciálna 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");
...
```

Súbor: PiskvorkyCanvasResizable.java

Transformácie



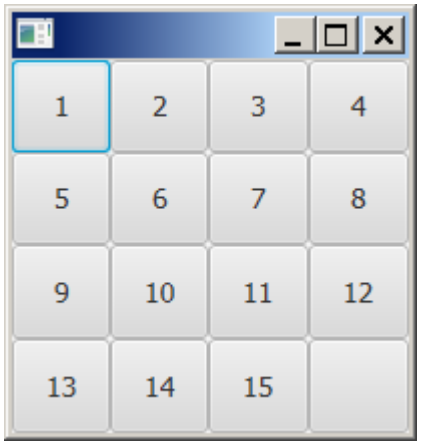

```
class Piskyground extends Canvas {                    // sú komplikovanejšie
                          // a už záleží na x,y lebo plocha môže byť obĺžnik
  private double cellWidth()        { return getWidth()/SIZE; }
  private double cellHeight()       { return getHeight()/SIZE; }
  private int 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);
  . . . .
```

Súbor:PiskvorkyCanvasResizable.java

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);
         }         // prekresli všetky Node v gp
   });
```



Súbor:Hra15.java

2.Riešenie škálovateľné

fitWidth/HeightProperty

Súbor:PiskvorkyGridButtonResizable.java

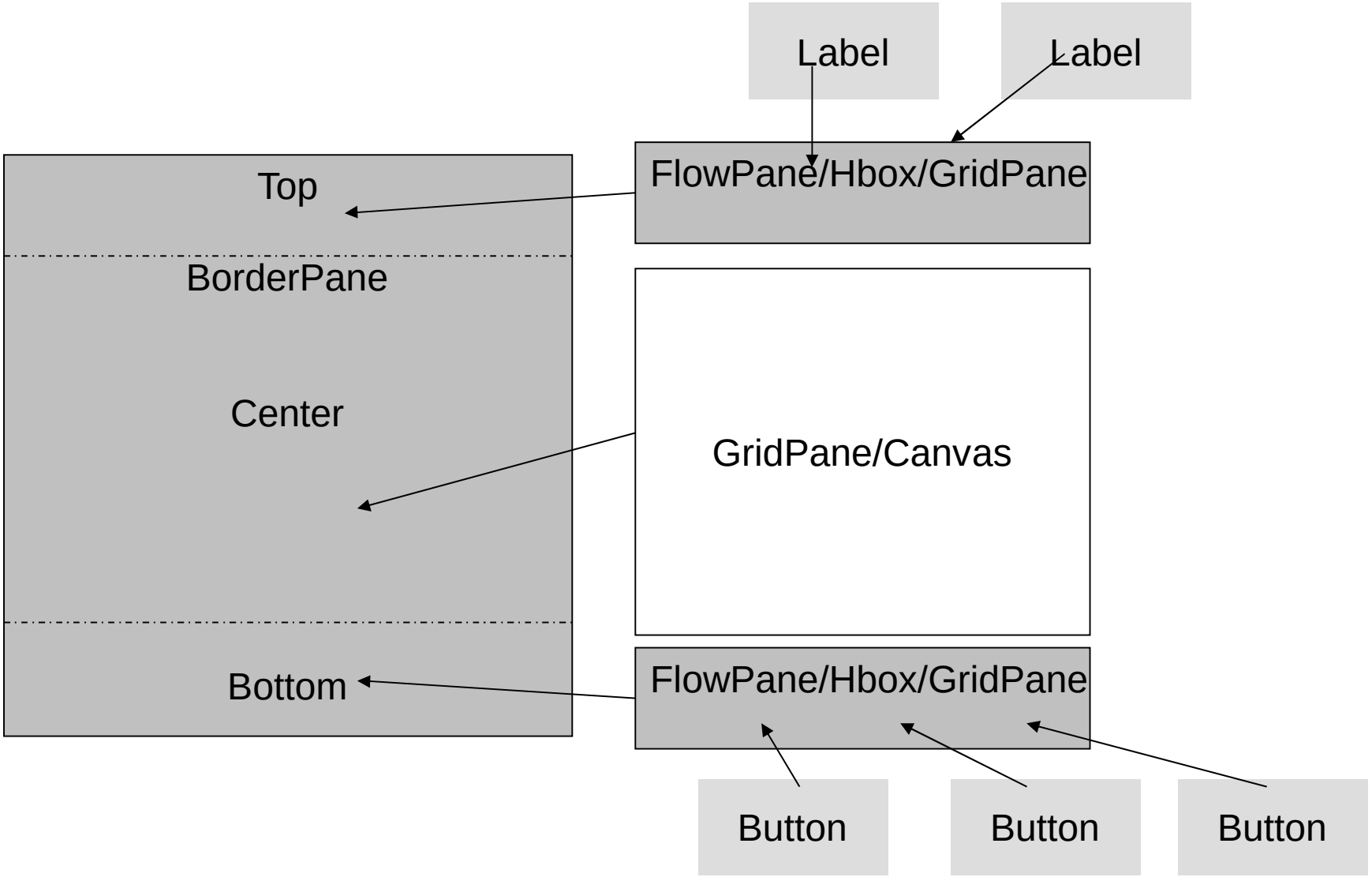
```
@Override
public void start(Stage primaryStage) {
    Piskyground pg = new Piskyground();
    pg.layoutBoundsProperty().addListener((observable, old, newBounds) -> {
        for (final Node child : pg.getChildren()) { // ak sa zmení rozmer pg
            final Control tile = (Control) child; // zmeň 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.Rieġenie ġkálovateĎné

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


Label
Label
FlowPane/Hbox/GridPane
Top
BorderPane
Center
GridPane/Canvas
Bottom
FlowPane/Hbox/GridPane
Button
Button
Button

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
```

PiskvorkyCanvasResizableControls.java

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(); }
} );
```

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 usmievačíka/smutňáčika, resp. do dokonalosti dovedené maľovanie bomby s tieňom ☺

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

```
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;
}
```

Case 2

- grafické dopracovanie projektu je dôležite, 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, prípane stlačením tlačidla "Roztoč".

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

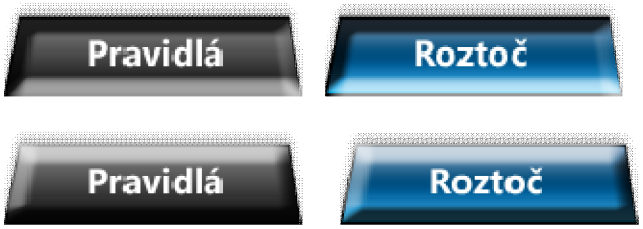
Toto okno zavriete kliknutím na ľubovoľné miesto v hre.

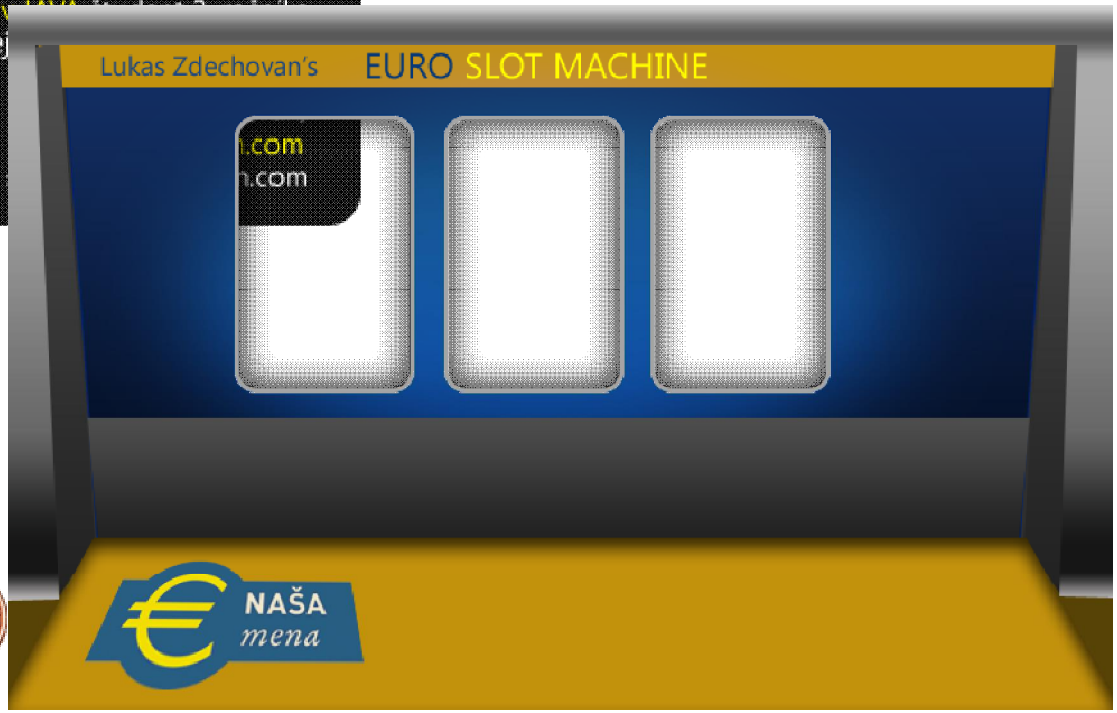
Výhry

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

O hre

Hru vytvoril ako projekt k predmetu Algoritmy v
aplikovanej



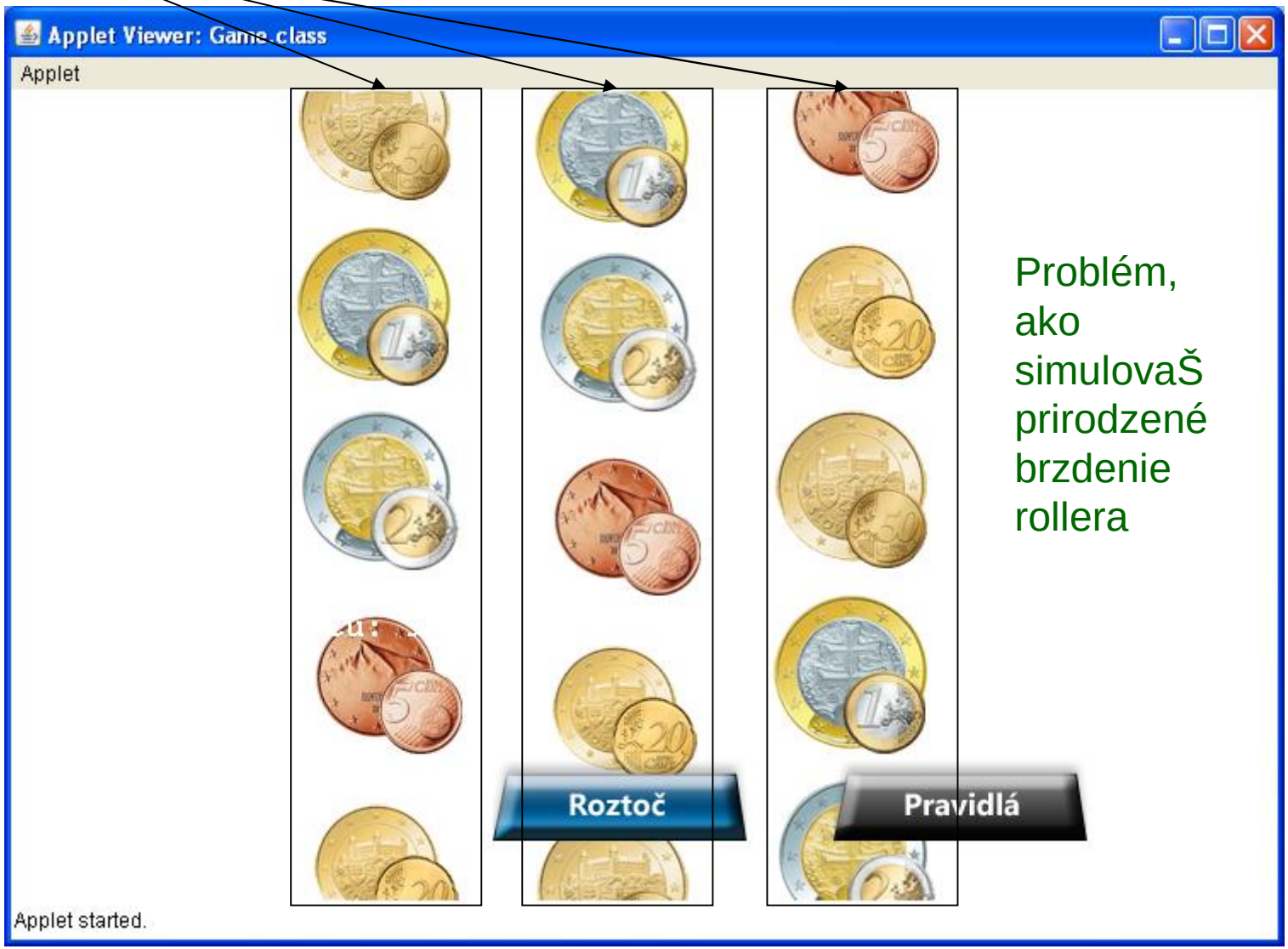


Spracovanie v Jave

Zdroj: Lukáš Zdechovan

Ļ o roller, to thread

Simulácia



Problém, ako simulovaŠ prirodzené brzdenie rollera

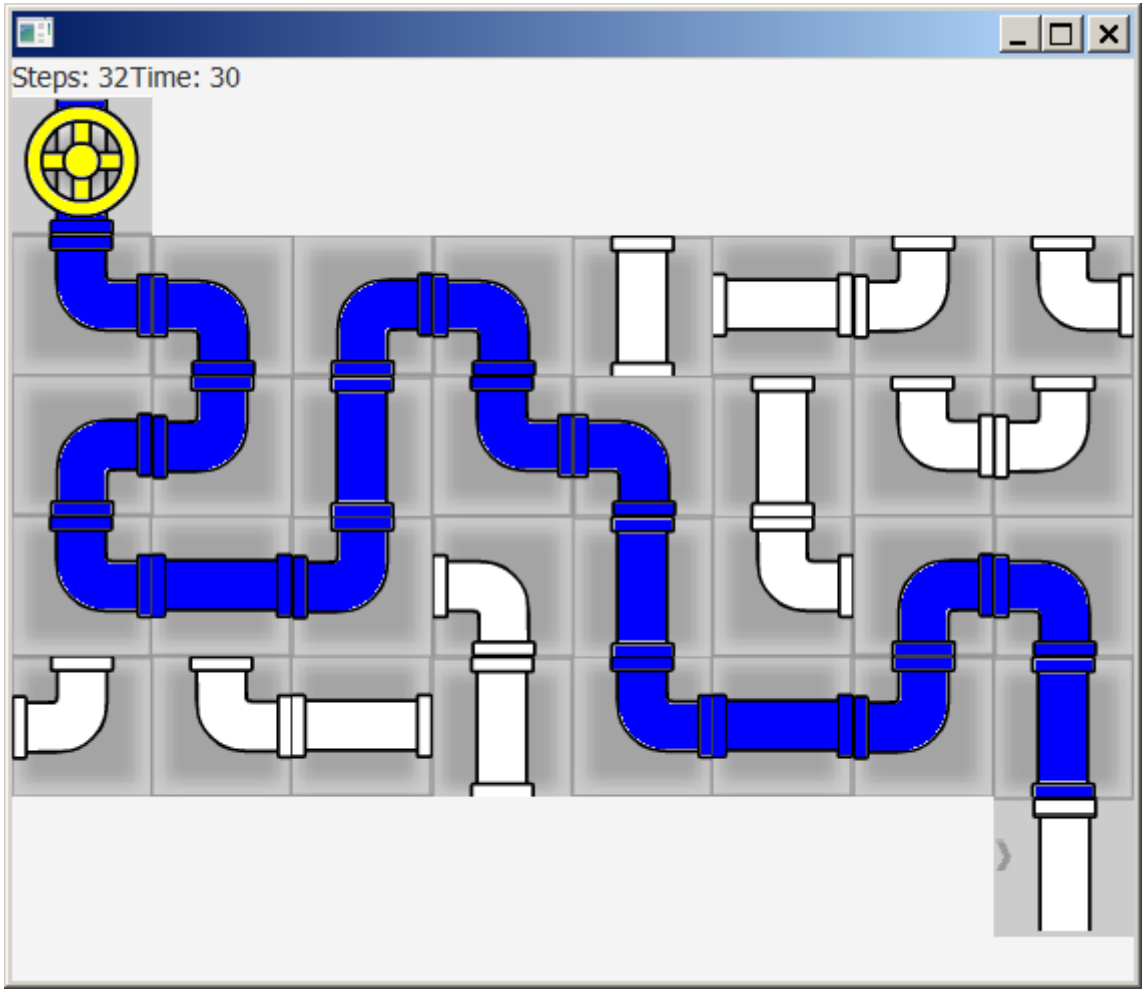
Case 3

- 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 appletu,

    8
    4
    12345623
    34613532
    35216311
    23654545
  - **ošetrenie udalostí** a rozpohybovanie appletu 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)



Oddeľte GUI

- kreslenie objektov,
- komponentov,
- appletu

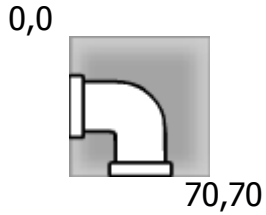
od logiky hry

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

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

Súbor: Plubmer.java

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

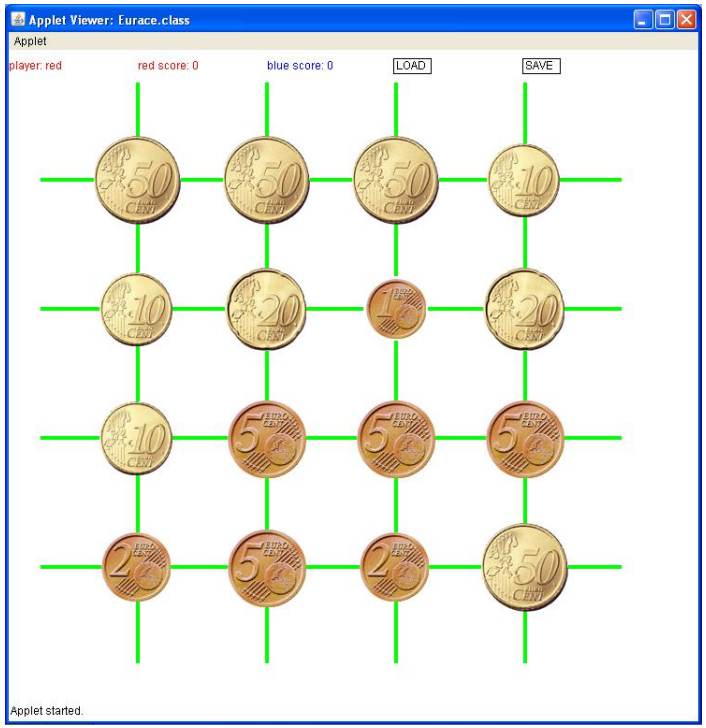
- čítanie vstupnej konfigurácie

```
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,
  - píšte 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

Súbor: Plubmer.java



Škálovanie

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


```
private static int dist = 120;                          // poľ iatoľ né nastavenia
private static int offset = dist;

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);
```

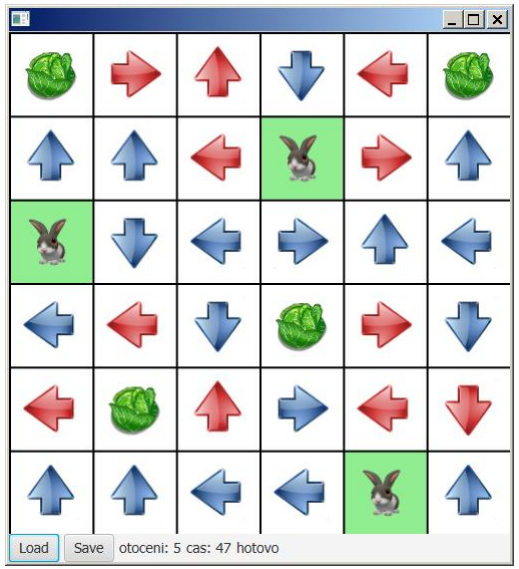
Súbor: Eurace.java

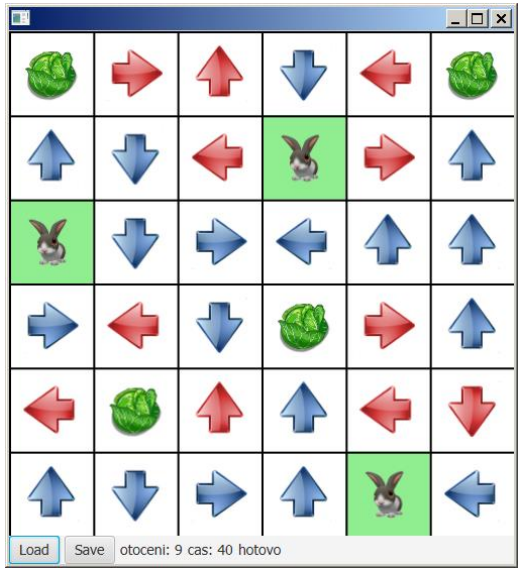
Nieľ o minuloroľ né

(Zajace a kapusty)

- Naprogramujte hru pre jedného hráľ a Zajace a kapusty. Hrá sa na ġtvorcovej mrieġke NxN ġtvorcov.

  V niektorých ġtvorcoch sa nachádzajú zajace, v niektorých kapusty a v ostatných ġípky smerujúce jedným zo ġtyroch smerov. Poľ et zajacov, kapusty a ġípiek môġe byŠvzhČadom na N rôzny. Niektoré ġípky sú ľ ervené - tie smerujú stále rovnakým smerom, niektoré sú modré a tie sa pri kliknutí myġou otáľ ajú o 90º. Príklad hernej situácie je na obrázku:



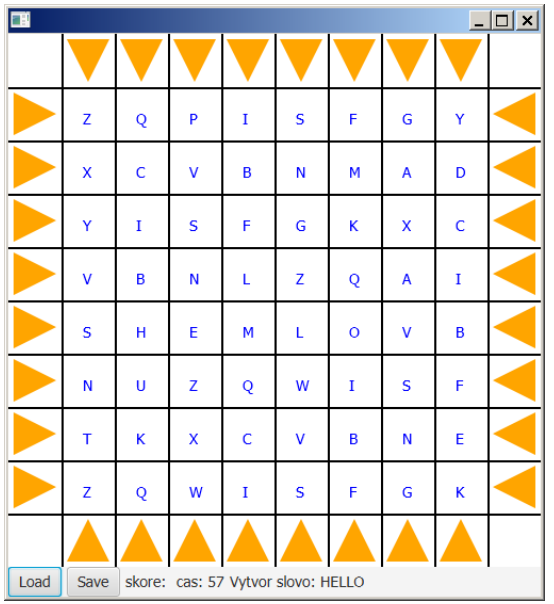


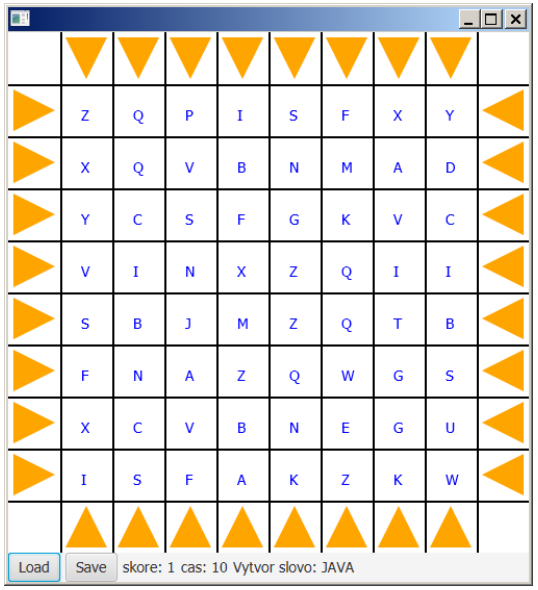
- CieČom hráľ a je pootáľ aŠmodré ġípky tak, aby sa vġetky zajace mohli podČa ġípiek dostaŠku kapuste. KeĽ zajac stúpi na políľ ko, kde je ġípka, musí pokraľ ovaŠsmerom podČa ġípky. Ak narazí na okraj poČa, alebo sa medzi nejakými ġípkami zacyklí, ku kapuste sa nedostal. Zaľ iatoľ ná konfigurácia hry je uloġená v súbore é

Nieľ o minuloroľ né

(**Písmenkovica)**

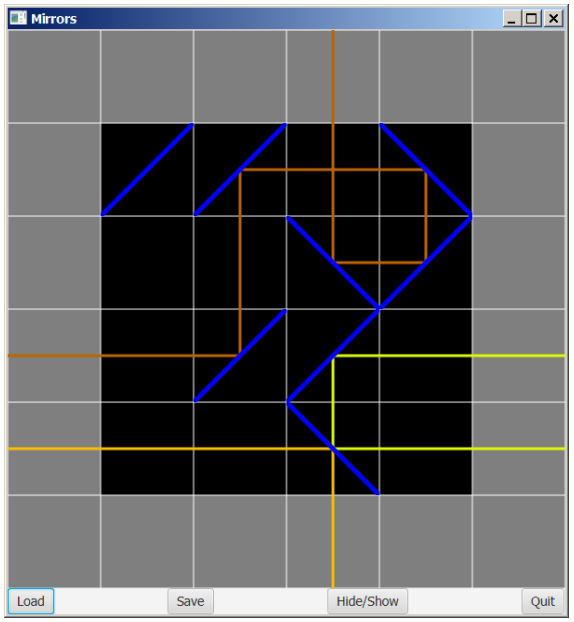
- V hre Písmenkovica sú v ģtvorcovej mrieģke rozmiestnené písmená anglickej abecedy. Na okrajoch vģetkých strán ģtvorca sú ģípky. Ich stlaľ ením dojde k otoľ eniu riadka alebo stĎca o jedno písmenko podĎa smeru ģípky. Niekde v okne je zobrazené slovo, ktoré treba zo susedných písmen v mrieģke vytvoriŠ buĎ vodorovne zĎava-doprava, zvislo zhora-nadol, ģikmo nadol vpravo, alebo ģikmo nahor vpravo. Ak sa to hráľ ovi podarí, písmená vytvoreného slova zmiznú a sú nahradené za ĎalģIe. Hráľ tým získava bod, cieĎové slovo sa zmení a hra pokraľ uje Ďalej. Na kaģdé slovo má 60 sekúnd ľ asu, ktoré sa mu odpoľ ítavajú a zostávajúci ľ as sa zobrazuje. Ak to nestihne, hra konľ í. Tlaľ idlami Save/Load uloģí/naľ íta aktuálny stav hry, priľ om z naľ ítaného stavu môģe pokraľ ovaŠv hre Ďalej. Zaľ iatoľ ná situácia hry, cieĎové slová a písmená, ktoré postupne nahradzujú písmená z vytvorených slov, sú uloģené v súbore a na zaľ iatku hry sa z neho naľ ítajú. Formát súboru je nasledujúci é





Nieľ o minuloroľ né

(**Zrkadlová sieŕ**)

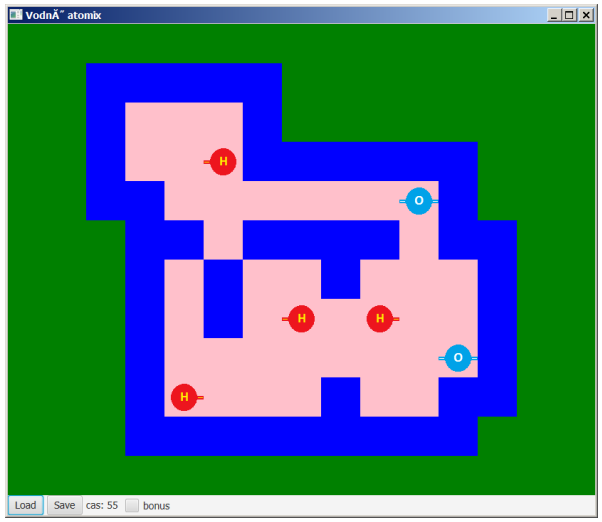


Å V ģtvorcovej sále s rozmermi NxN sú v niektorých políľ kach umiestnené diagonalne zrkadlá, v ilustráciach sú zobrazené modrou farbou. Môģu byŠ dvoch typov, / alebo \. Na kraji ģtvorcovej sály sú políľ ka, ktoré obsahujú zdroje svetla rôznych farieb. Krajné Ľavé a pravé políľ ka (N+N) obsahujú vodorovný zdroj svetla, krajné horné a dolné políľ ka (N+N) obsahujú zvislý zdroj svetla. Roģné políľ ka (4) nemajú ģiadnu funkciu. Pre jednoduchosŠ znázornenia scény plochu kreslíme do ģtvorcovej mrieģky s rozmermi (N+2)x(N+2).

Ak zapneme svetelný zdroj, vodorovný ľ i zvislý, svetlo sa zaľ ne ģíriŠ daným smerom cez hraciu plochu. Ak je políľ ko prázdne, prejde ním. Ak je v Ŕom diagonálne zrkadlo, odrazí sa od neho presne v duchu príslovia: uhol odrazu je uhol dopadu. Samozrejme, keŇže ide o diagonálne zrkadlá, tak tento uhol môģe byŠ len 45 stupŕov. Pri rôznych polohách zrkadiel môģu vzniknúŠ 4 situácie (premyslite si...). Niektorým políľ kom lúľ opustí hraciu plochu, vy znázorŕujete jeho cestu. Svetelné zdroje sú rôznych farieb, a farieb mate dosŠ (staľ í si ich nejako vygenerovaŠ).

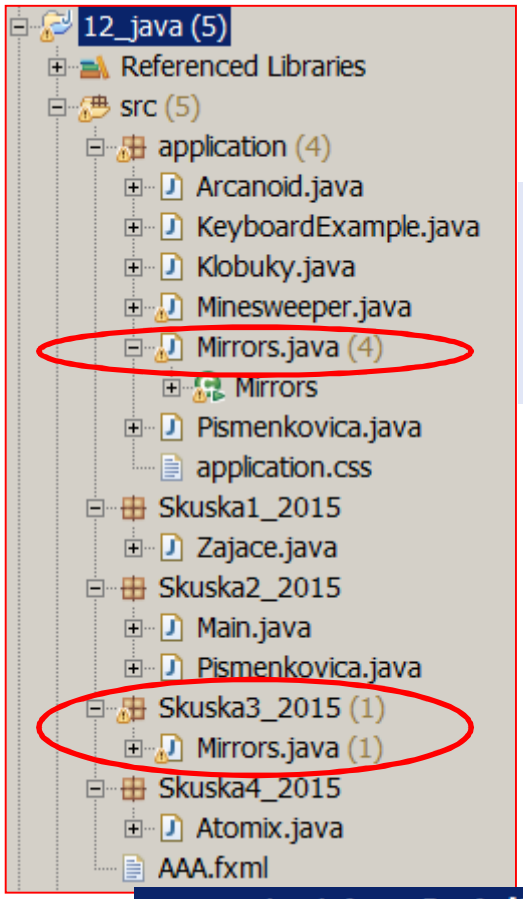
Nieľ o minuloroľ né

(Atomix)



Å V hre Atomix sa z atómov konġtruujú molekuly rozliľ ných zlúľ enín. Kaġdý atóm má naznaľ ený smer väzby a poľ as hry sa nedokáġe otoľ íŠ- väzba smeruje stále tým istým smerom. V naġej verzii hry sa zameriame iba na molekulu vody. Po kliknutí myġou na niektorý atóm sa tento atóm zvýrazní. Druhé kliknutie na voĎné políľ ko v prázdnej uliľ ke, ktorá vychádza od atómu jedným zo ġtyroch smerov, atóm uvedie do pohybu. Atóm sa vġak zastaví, aġ keĽ narazí do steny, alebo do iného atómu. Potom je moġné kliknúŠna nejaký atóm znova. V prípade, ġe sa podarí vytvoriŠmolekulu vody, t.j. vedĎa seba sa bude vodorovne alebo zvislo nachádzaŠatóm vodíka, kyslíka a zasa vodíka a budú previazané vzájomnými väzbami, hráľ level splnil a môġe postúpiŠdo ĽalġieHo levelu.

Prémia: Súťaž krásy




```
class MyColor implements Serializable {
    int red, green, blue;
    private static final long serialVersionUID = -6552319171850636236L;
```

application.Mirrors$MyColor is serializable but also an inner class of a non-serializable class [Troubling(14), High confidence]

```
static GridPane gp;

@Override
public void start(final Stage primaryStage) throws Exception {
```

Write to static field Skuska3_2015.Mirrors.gp from instance method Skuska3_2015.Mirrors.start(Stage) [Of Concern(15), High confidence]