JavaFx

pokračovanie

Už vieme (treba pre quadterm2):

- kresliť do Canvas, vložiť Canvas->Pane->Scene->Stage, HowToWithFx
- simulovať dej pomocou Thread/Timeline/AnimationTimer,
- prekreslovať GUI komponenty pomocou Platform.runlater
- odchytiť udalosti od ActionEvent/KeyEvent/MouseEvent,
- aj to že uhol dopadu sa rovná uhlu odrazu ☺
- rôzne spôsoby návrhu jednoduchej (pravouhlej) hry (Grid/Canvas/Button)

Dnes bude:

- aspekt škálovateľnosti (re-zoom hracej plochy),
- perzistencia (ukladanie dát),

Zdroj a literatúra:

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

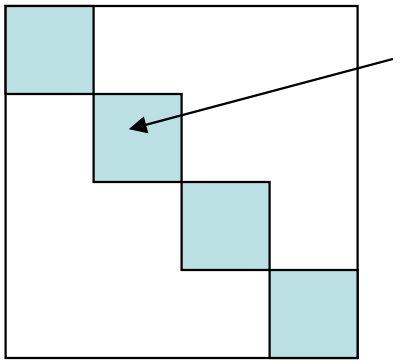
Cvičenia: jednoduché aplikácie s GUI:

- škálovateľná logická hra

Hracia plocha

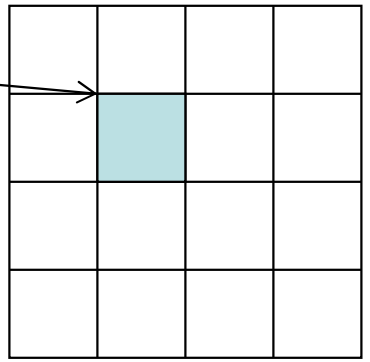
hracia plocha je často šachovnica rôznych rozmerov. Ako ju implementujeme:

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

Pexeso

na cvičení

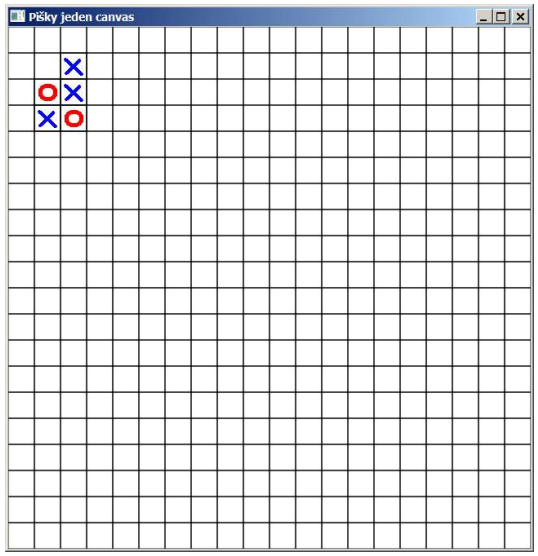

```
public class Pexeso extends Application {
State state = new State();
Playground playground;
    public class Playground extends GridPane {
       public class Cart extends Pane { ...  }
    }
}
// POZOR, TOTO NEMOŽE BYŤ VNORENÁ TRIEDA, lebo  ... ani public
class State implements Serializable {
   private static final long serialVersionUID = 918972645L;
     public class CartObject implements Serializable {
        private static final long serialVersionUID = 911775039L;
            int id;
            boolean visible = false;
            transient ImageView pikaImage;        // neserializovať
```

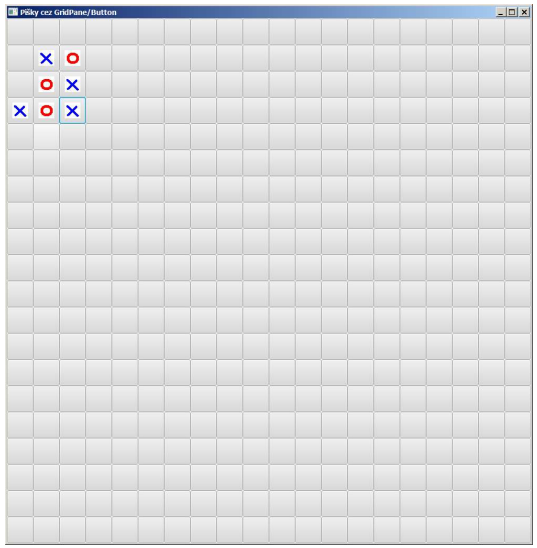
- **transient** znamená, že nechceme serializovať,
- väčšina JavaFX objektov nie je serializovatelná !!! a dostanete NotSerializableException…
- **transient static** a **transient final** nerobí nič

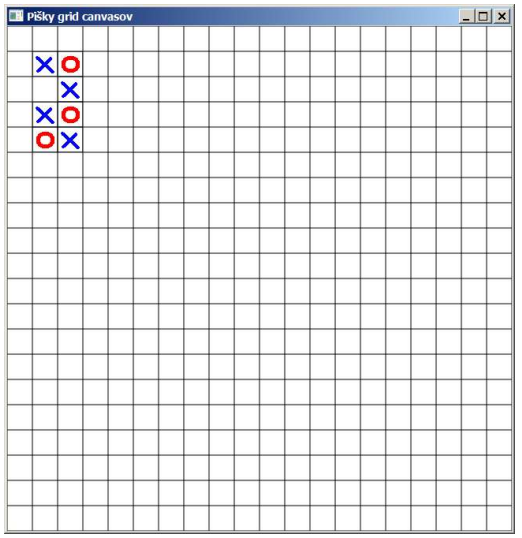
Súbor: Pexeso.java

Škálovateľnosť

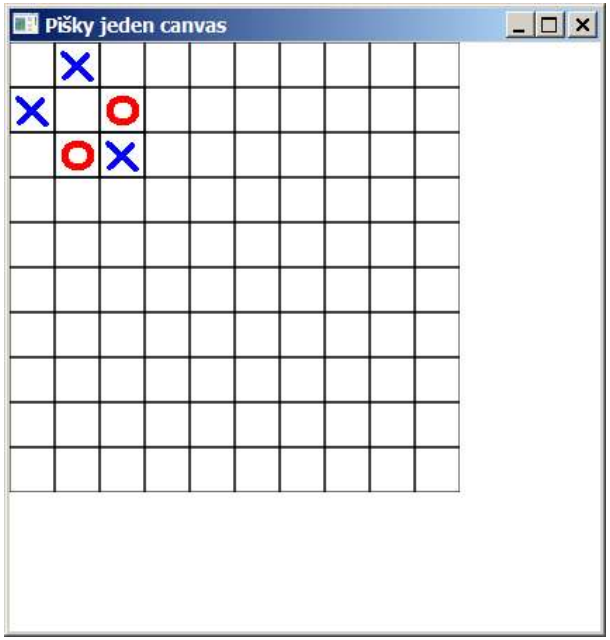
- Škálovateľnosť hry (miesto 10x10 chceme hrať 20x20):

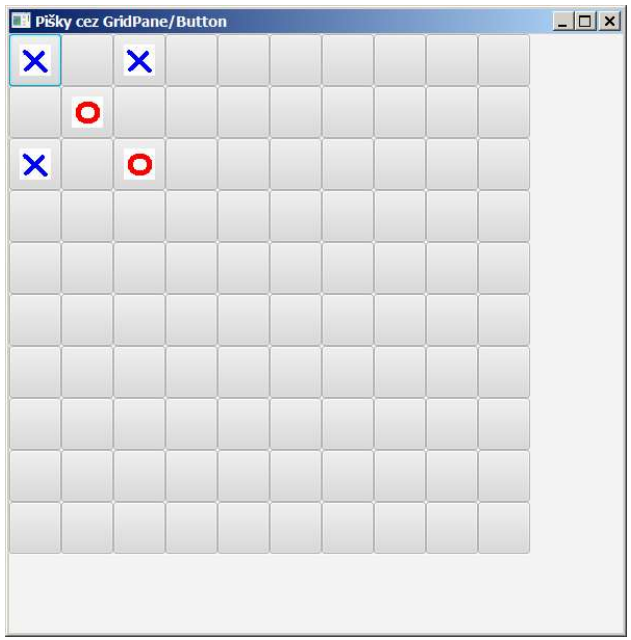


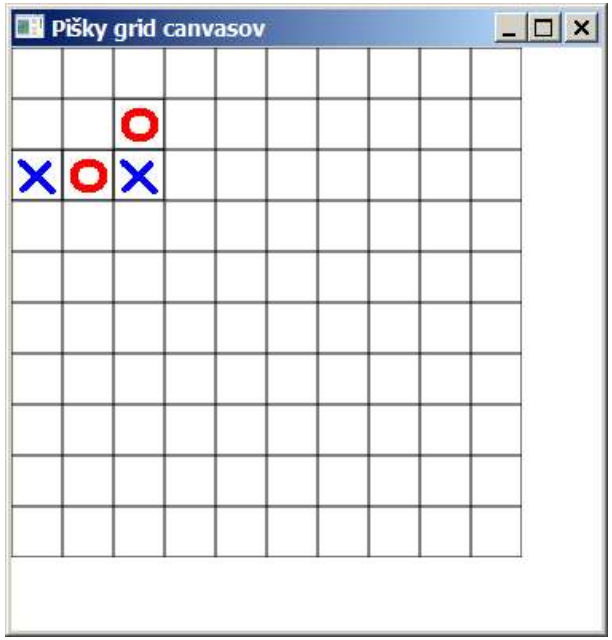




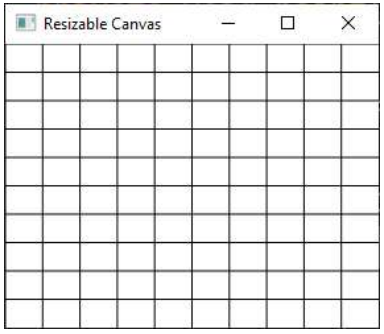
- Škálovateľnosť GUI (zmeníme rozmer okna):







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 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

Properties & 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(priemer.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

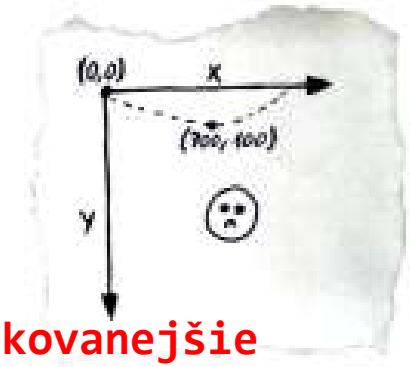
1.Riešenie škálovateľné

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(); }
} ! Napriek tomu, že ide o lineárne transformácie, abstrahujte ich do metód !

public void paintCell(int i, int j) {
  Image imageO =          // keď potrebujem 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

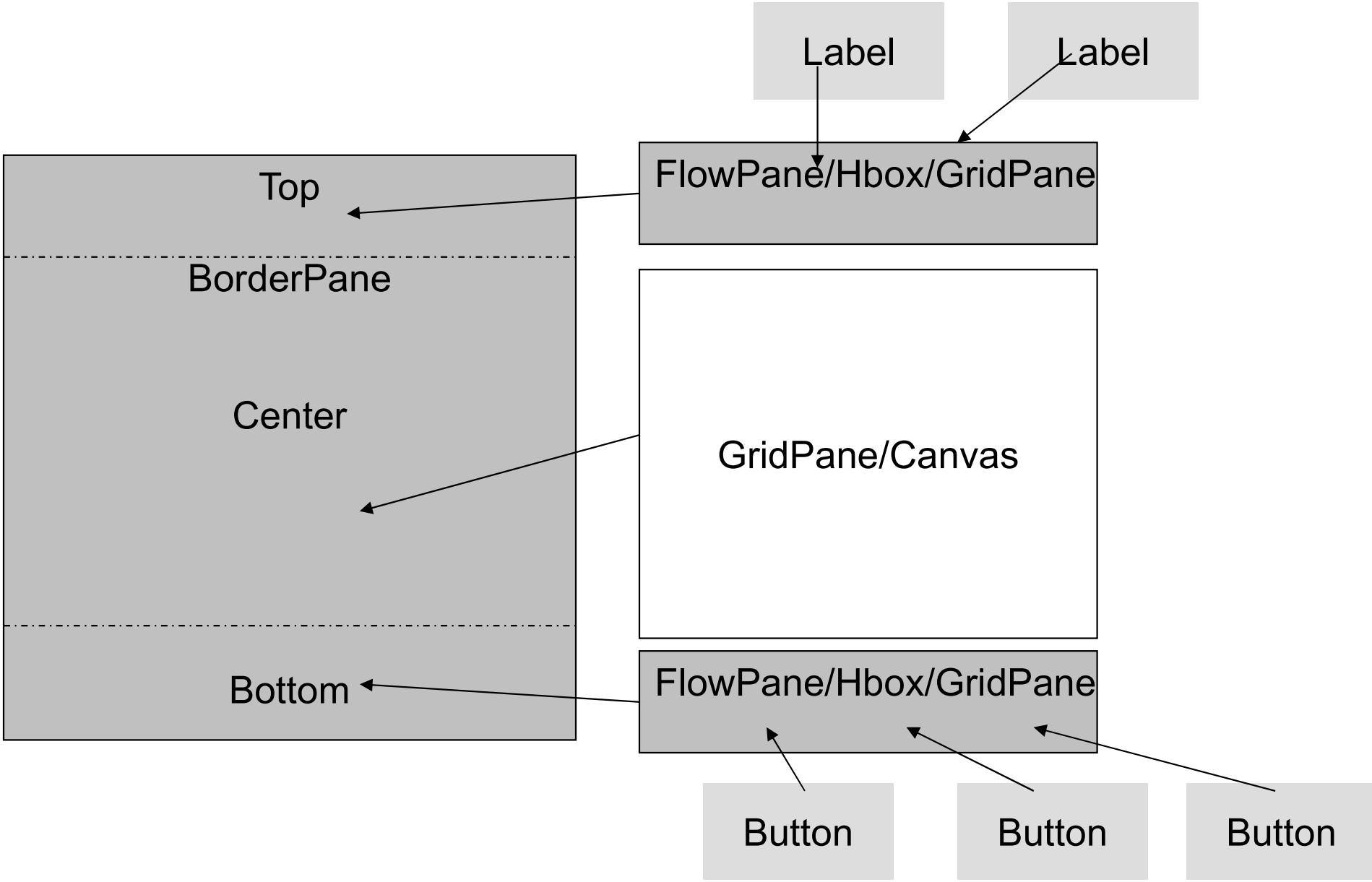
```
@Override
public void start(Stage primaryStage) {
    Piskyground pg = new Piskyground();
    pg.layoutBoundsProperty().addListener((observable, old, newBounds) -> {
        for (final Node child : pg.getChildren()) { // ak sa zmení rozmer pg
            final Control tile = (Control) child; // zmeň 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().substract(4)); // 2px okraj
        imageO.fitHeightProperty().bind(heightProperty().substract(4)); // 2px lem
```

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


Label
Label
Top
FlowPane/Hbox/GridPane
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

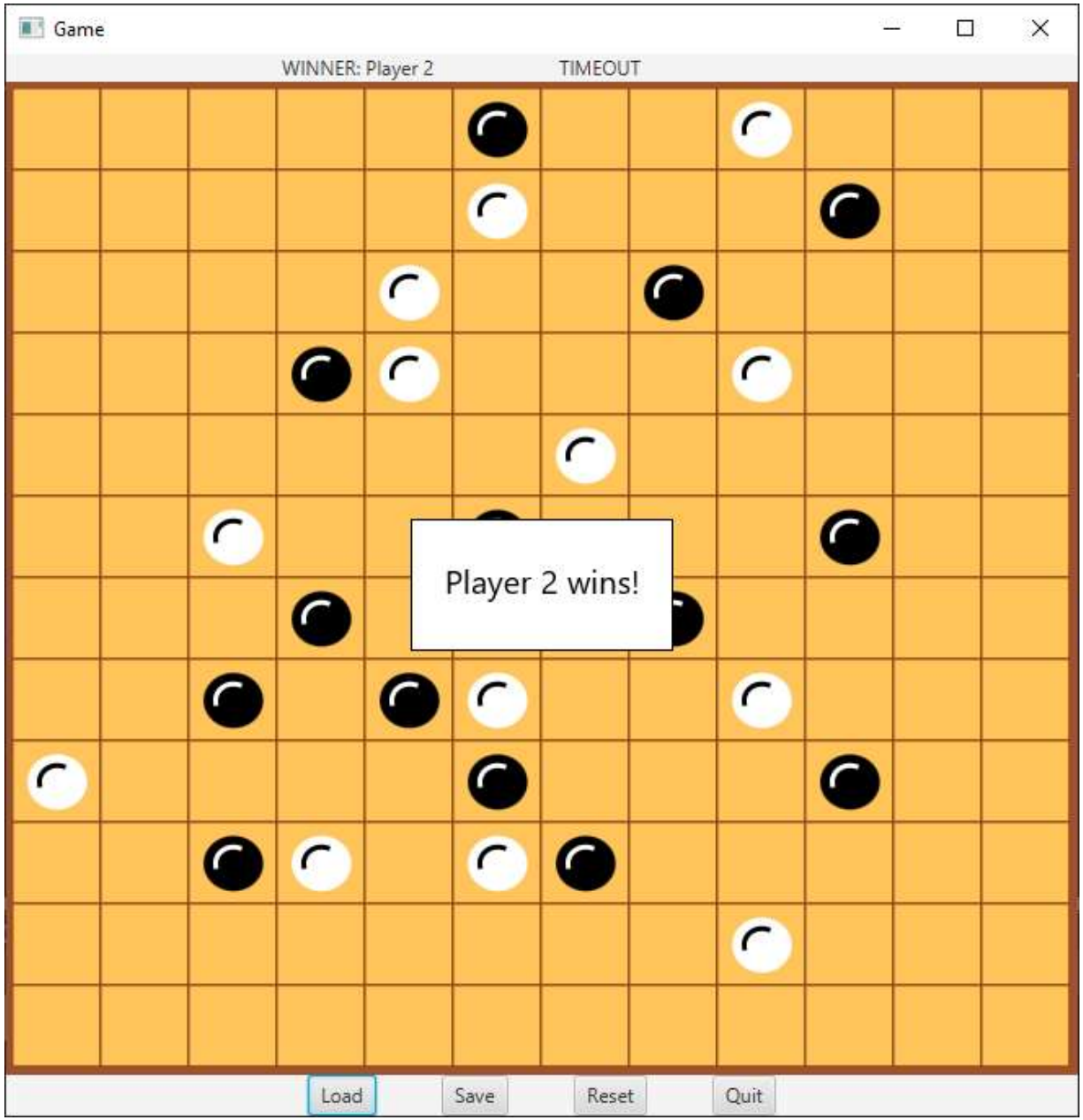
Control

```
btnQuit.setOnAction(event -> Platform.exit());

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();
```

Scéna hry



BasicGame.java

Quadterm 2

(štvrtok 14.5. **17:10**)

Bude:

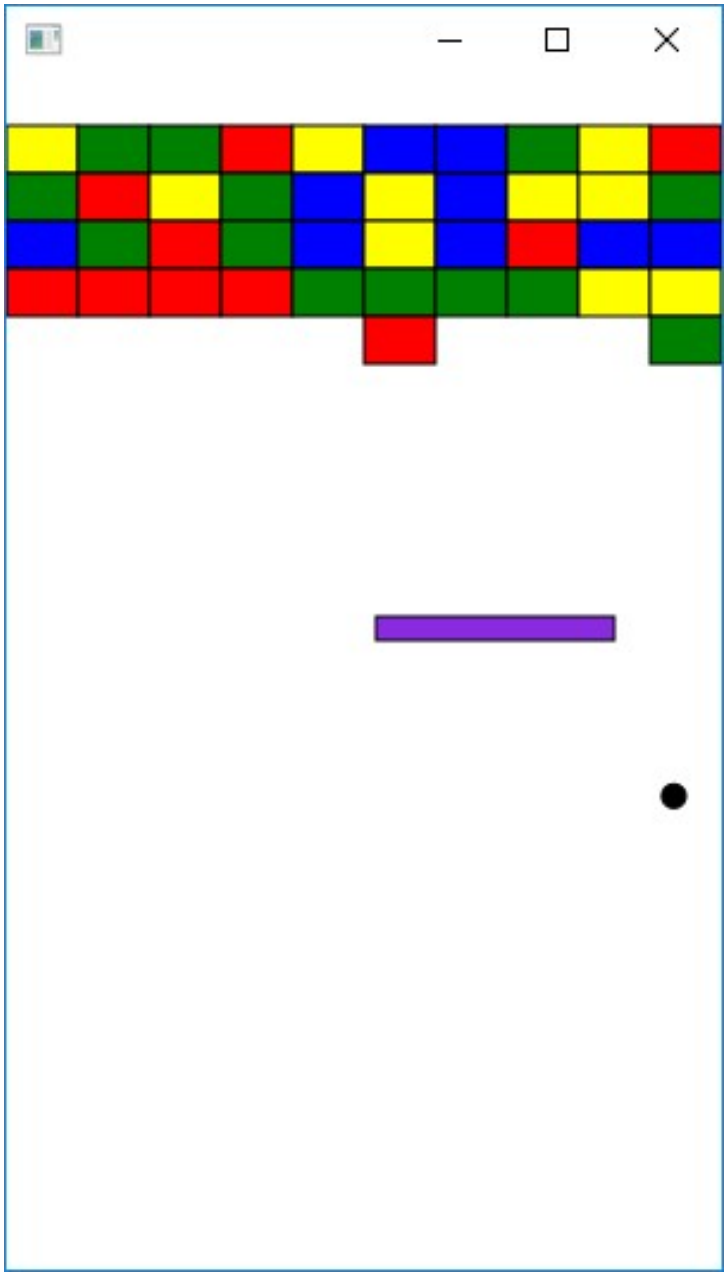
- simulácia niečoho, čo sa hýbe
  - Thread/Timeline/AnimationTimer
- odchytávanie udalostí
  - myš/klávesnica
- kreslenie do
  - Pane
    - getChildren().clear()
    - new Rect(…)
    - new Circle(…)
    - new ImageView()
    - getChildren().add(…)
  - Canvas
    - gc = getGraphicContext
    - gc.strokeLine
    - gc.fillRect()
    - gc.drawImage()

Nebude:

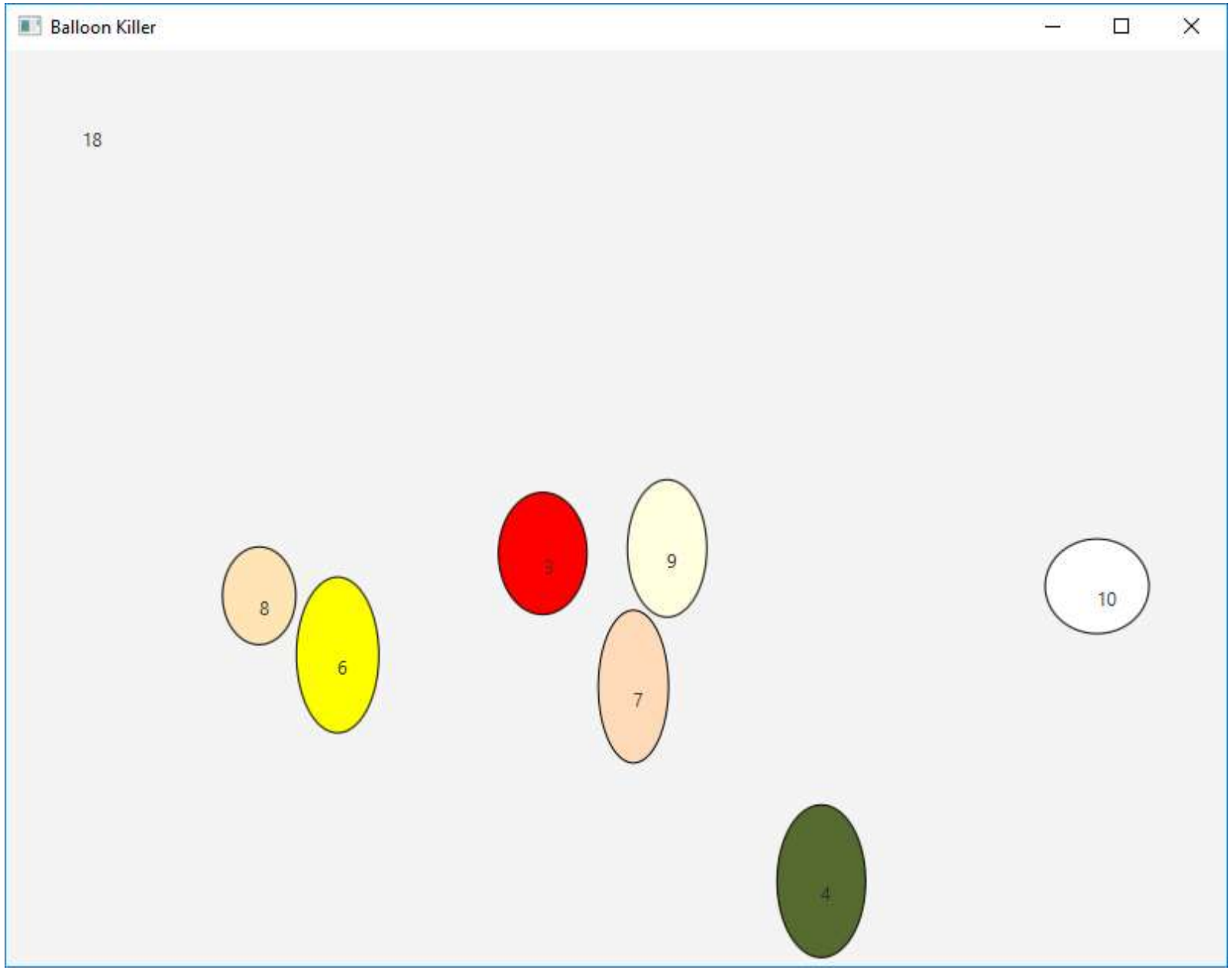
- serializácia
- zložitejší layout
- import project –

  asi nebude template ☹
- unit testy ☺

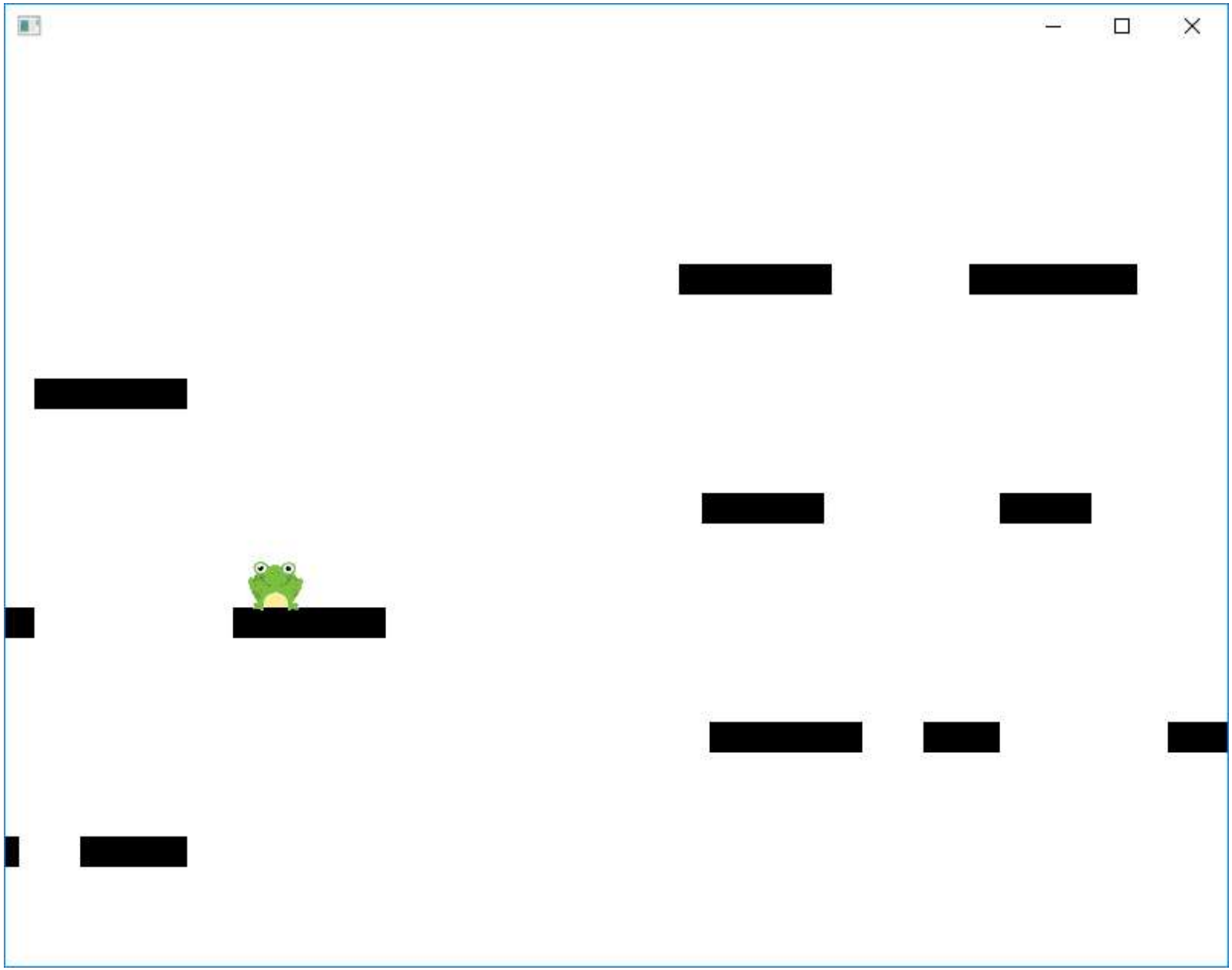
Arkanoid



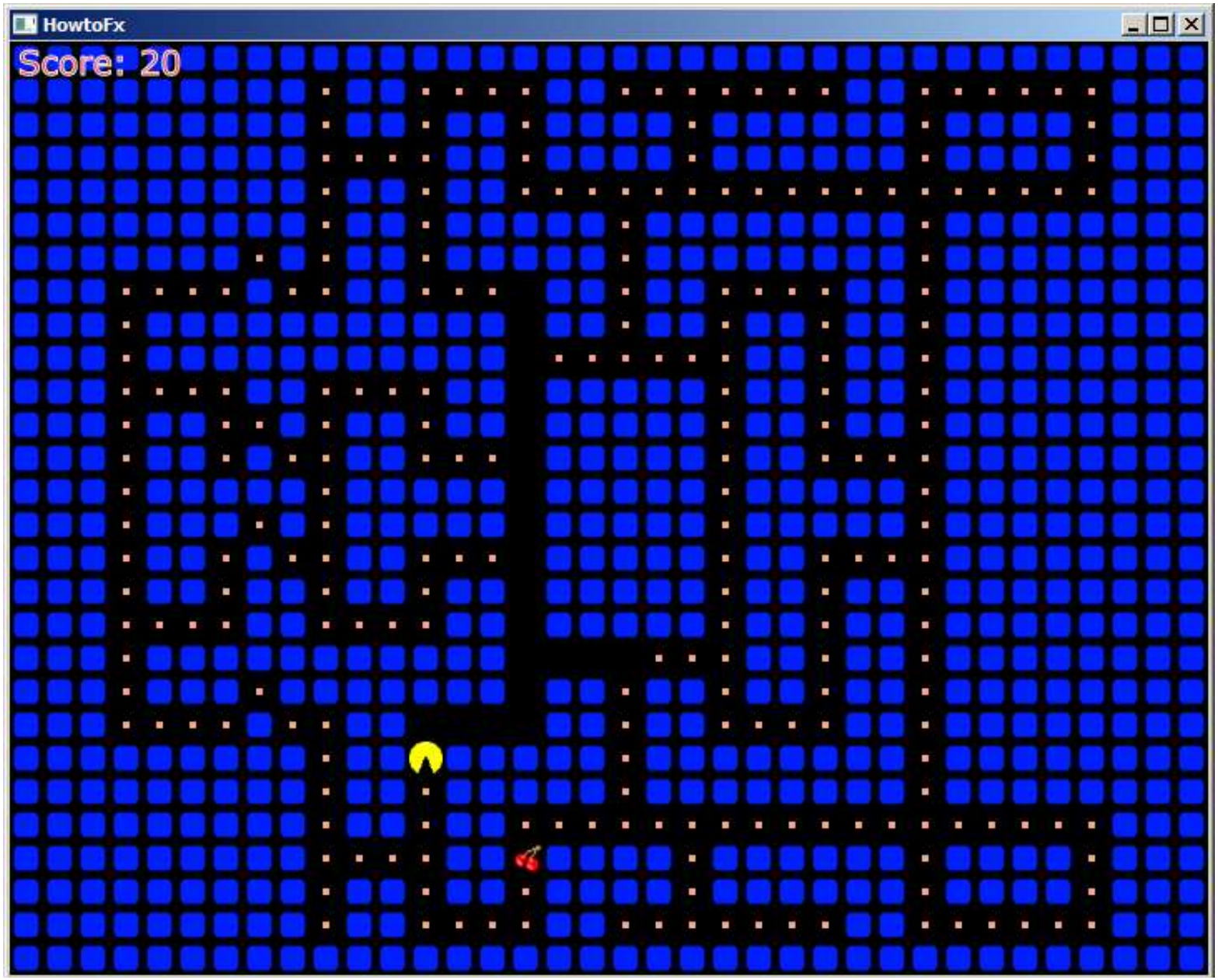
Balloon Killer



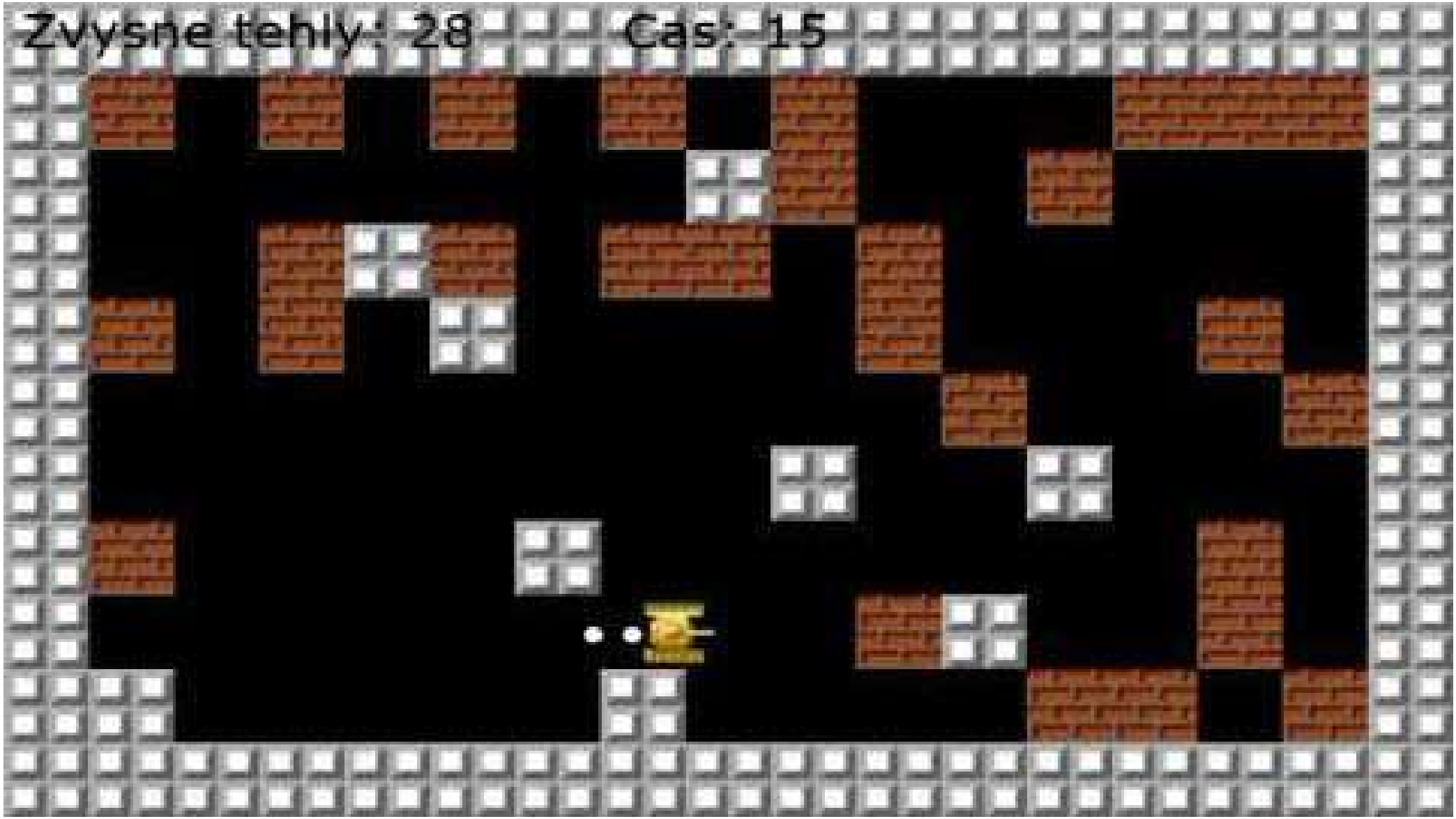
Žabky



Pacman

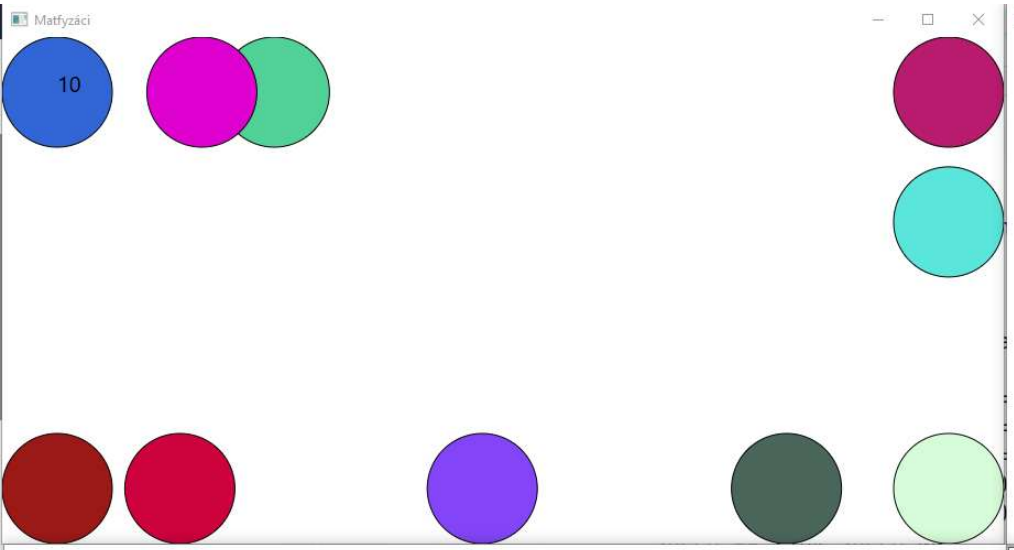


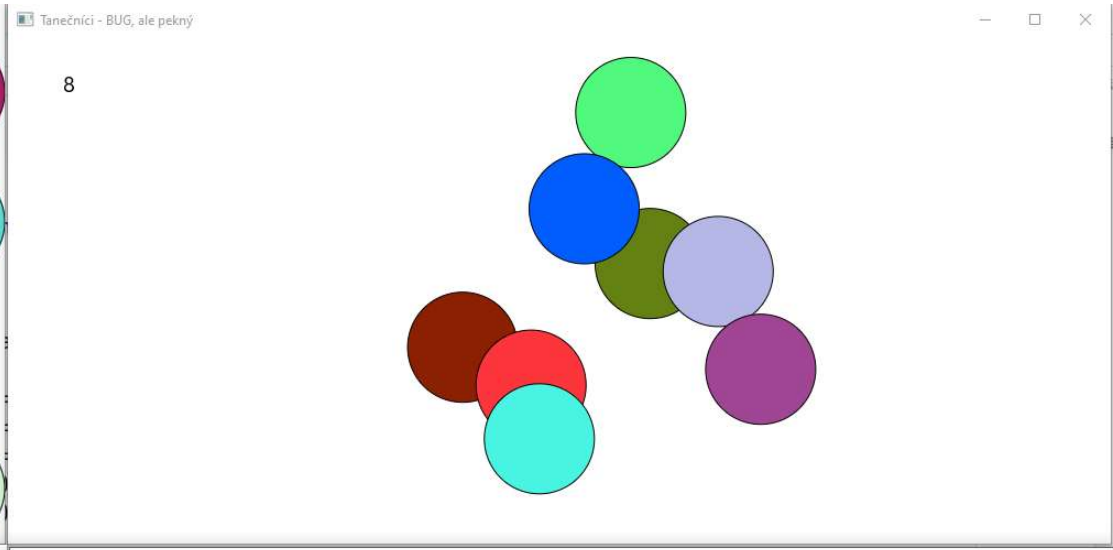
Tanky

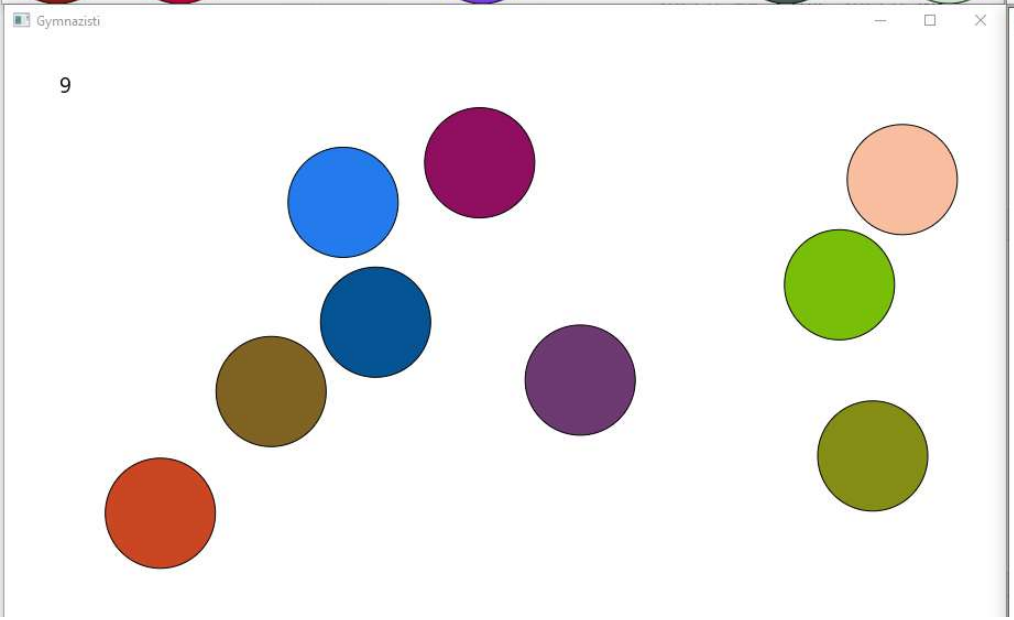


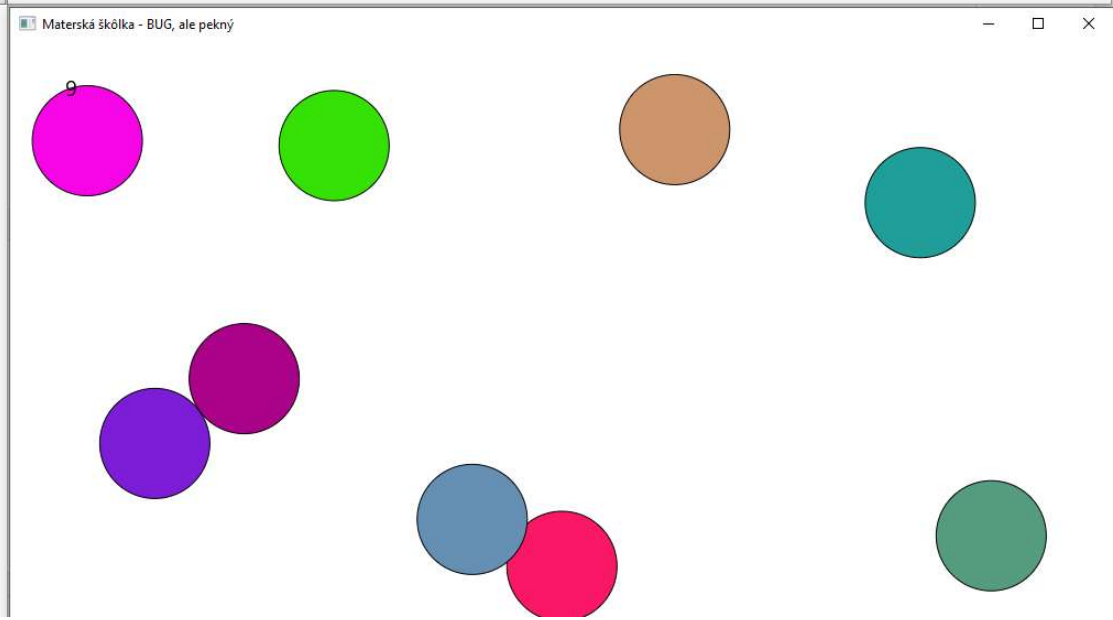
Cenzorovaná verzia

Späť do školy









Hodnotenie:

- **[2 body]** vykreslenie scény, odchytávanie polohy myši (*MouseMoved*), zobrazenie gule náhodnej hodnoty v hornej časti hracej plochy,
- **[2 body]** každá hodnota 2^n má svoju vlastnú farbu, guľa má vo svojom strede napísanú hodnotu, a plocha kruhu zobrazúceho guľu je priamo úmerná hodnote gule,
- **[2 body]** pustenie gule na (*MouseClicked*), guľa začne nejako smerovať dole, simulácia pohybu, aj to, že ostane na hracej ploche,
- **[1 bod]** guľa padá rovnomerne zrýchleným pohybom, teda zrýchľuje kvôli gravitácii,
- **[1 bod]** simulácia sa dá zastaviť ľubovoľným klávesom (KeyPressed),
- **[1 bod]** pri odraze od stien sa spomalí,
- **[2 body]** pri zrážke s inou guľou rovnakej veľkosti sa obe spoja do novej gule s dvojnásobnou hodnotou. Hodnota gule je úmerná s plochou kruhu, ktorá ju zobrazuje,
- **[3 body]** rôzne gule sa pri dotyku odrážajú, hodnotí sa vizuálny dojem simulácie čo najbližšie k biliardovým guliam,
- **[1 bod]** guľa sa odráža od stien miestnosti, uhol dopadu sa rovná uhlu odrazu,

Bonusy (hodnotia sa len, ak máte aspoň 50% z predchádzajúcich podúloh):

- **[1 bod]** niekde v hracej ploche zobrazujte súčet hodnôt všetkých gúľ a uplynutý čas v sekundách,
- **[1 bod]** hracia plocha sa zá zväčšiť a zmenšiť počas hry a scéna sa adaptuje na zmenenú hraciu plochu.

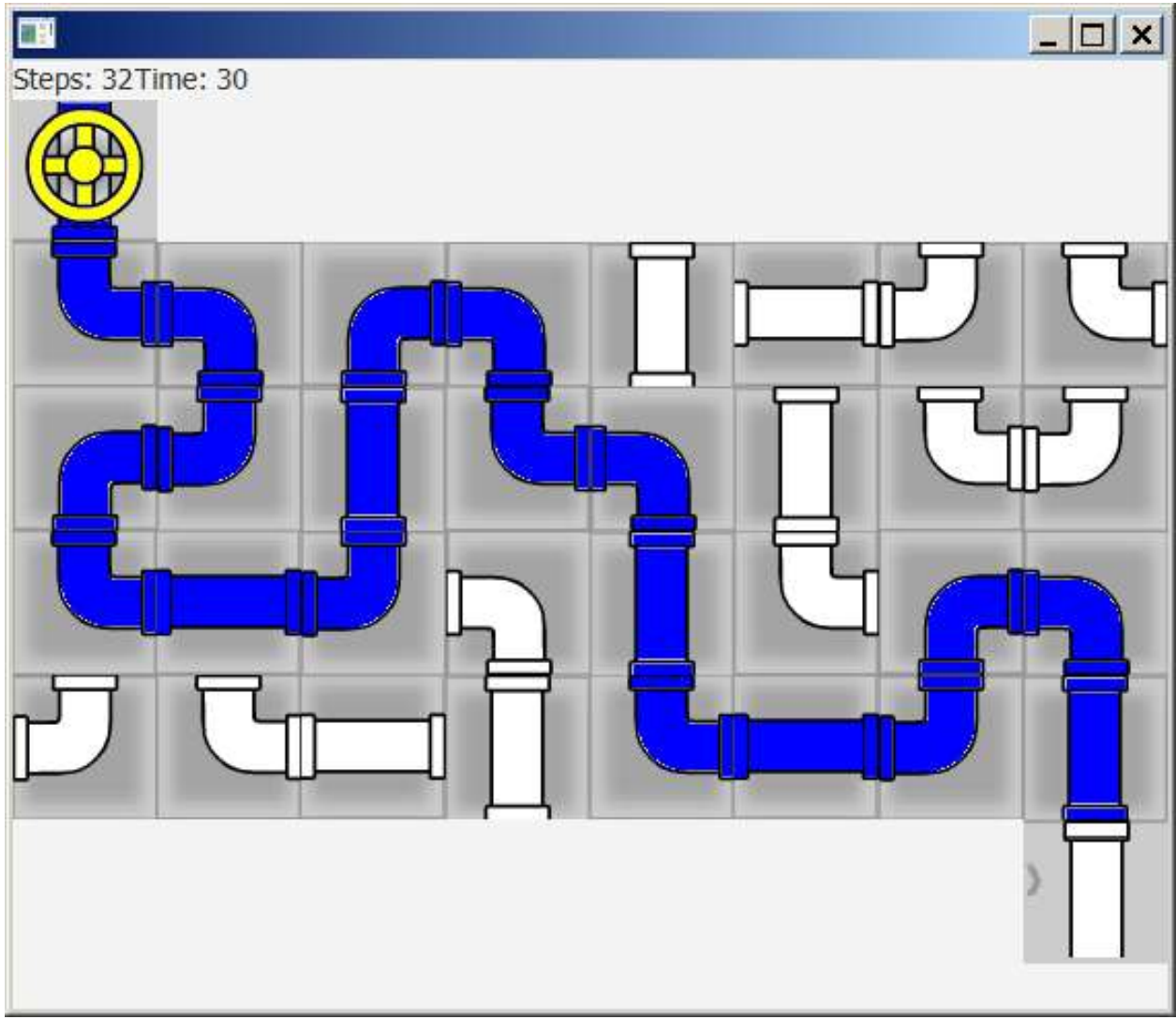
Ohňostroj

Case 3

- zadanie Plumber:
- tri skúšky (zadania) visia na stránke predmetu

- príklad ilustruje štruktúru skúšky:
  - **čítanie konfigurácie** hry súboru a vykreslenie plochy, konštrukcia sceny,
    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)



Oddeľte GUI

- kreslenie objektov,
- komponentov,

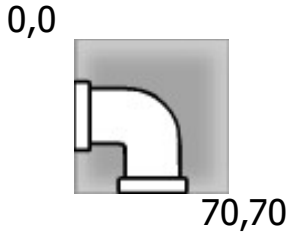
od logiky hry

- 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

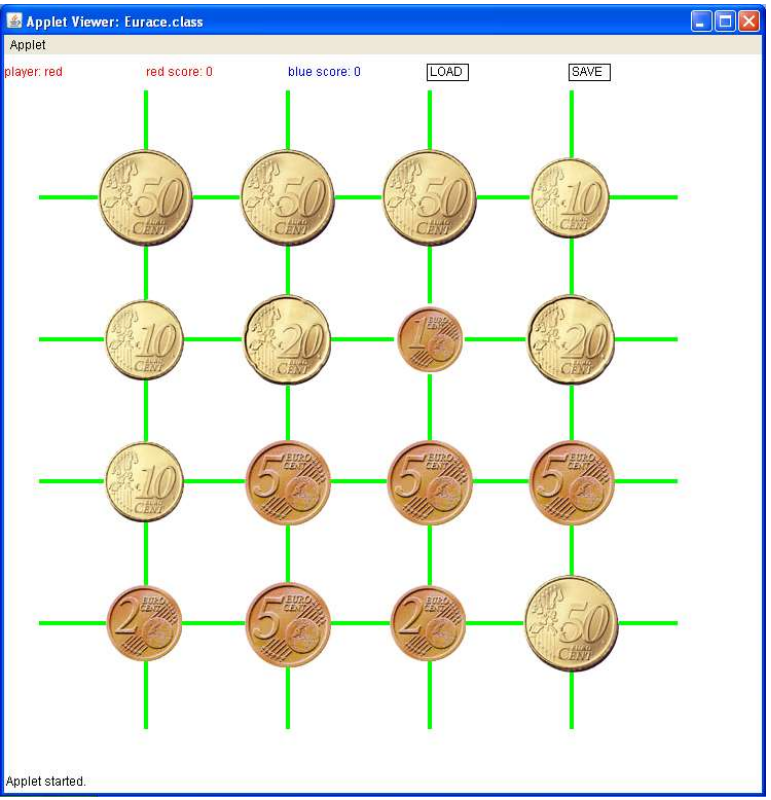
- čítanie vstupnej konfigurácie

```
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 – PiškyState..

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;                    // počiatočné 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);
```

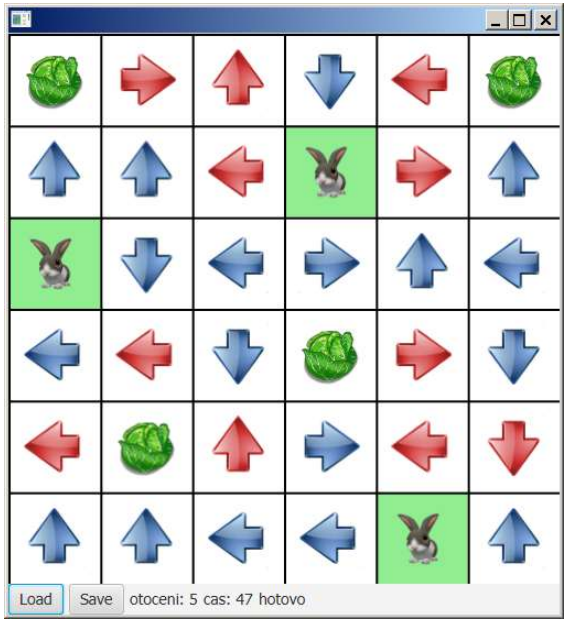
Súbor: Eurace.java

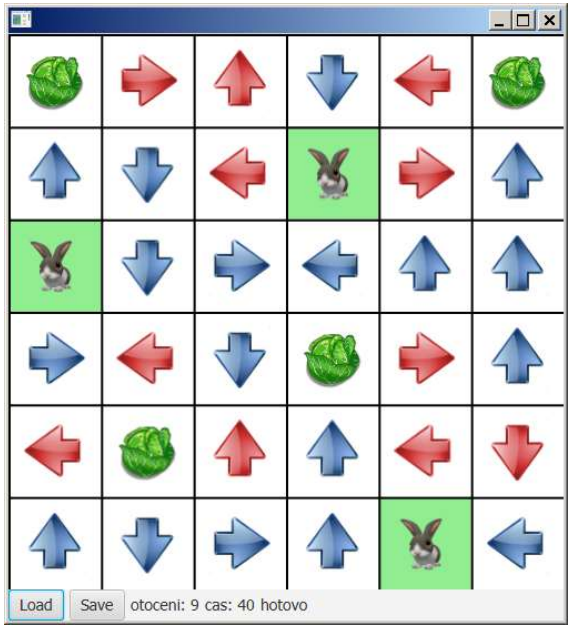
Niečo novšie

(**Zajace a kapusty**)

- Naprogramujte hru pre jedného hráča Zajace a kapusty. Hrá sa na štvorcovej mriežke NxN štvorcov.

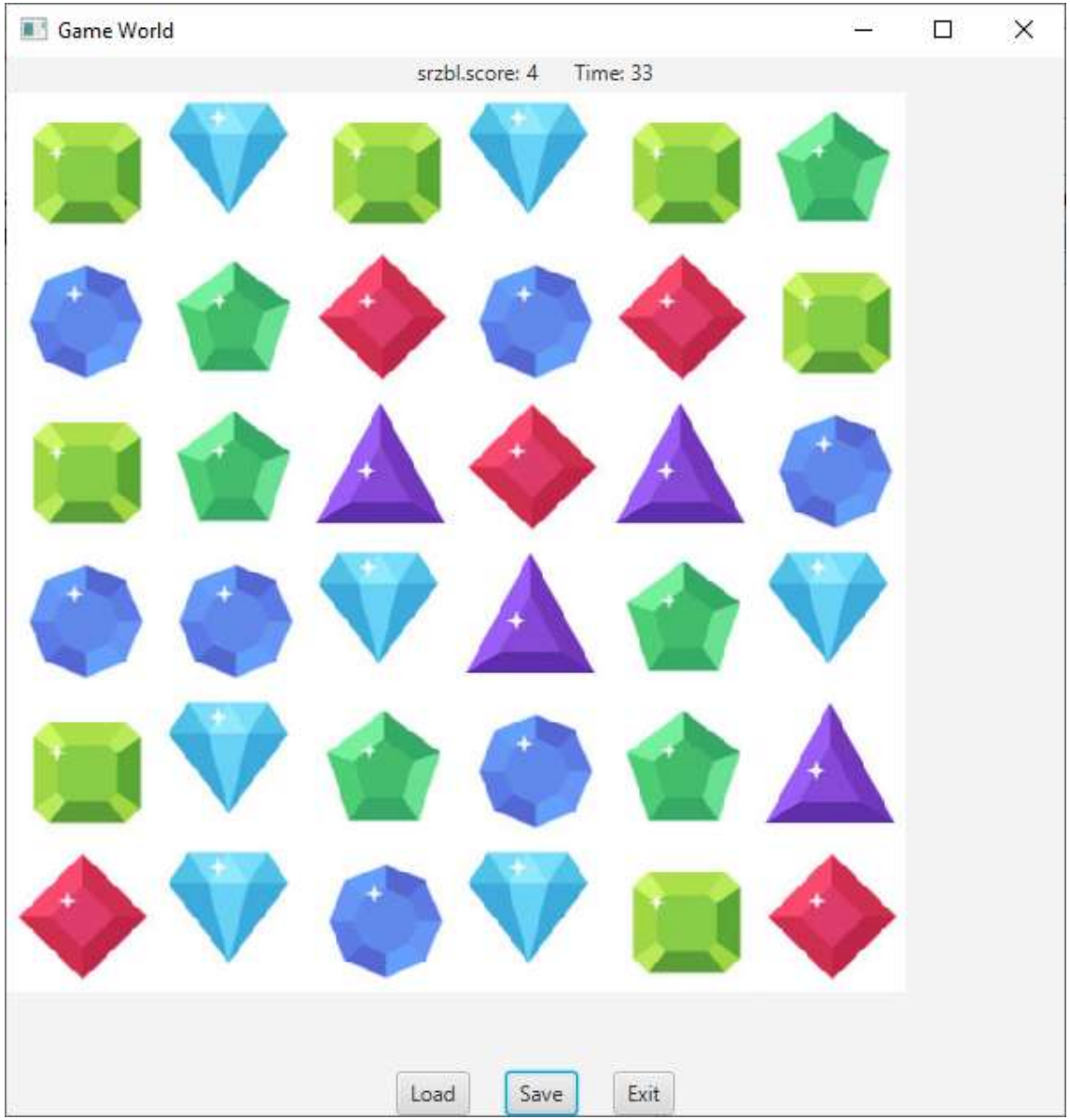
  V niektorých štvorcoch sa nachádzajú zajace, v niektorých kapusty a v ostatných šípky smerujúce jedným zo štyroch smerov. Počet zajacov, kapusty a šípiek môže byť vzhľadom na N rôzny. Niektoré šípky sú červené - tie smerujú stále rovnakým smerom, niektoré sú modré a tie sa pri kliknutí myšou otáčajú o 90^{o}. Príklad hernej situácie je na obrázku:





- Cieľom hráča je pootáčať modré šípky tak, aby sa všetky zajace mohli podľa šípiek 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 …

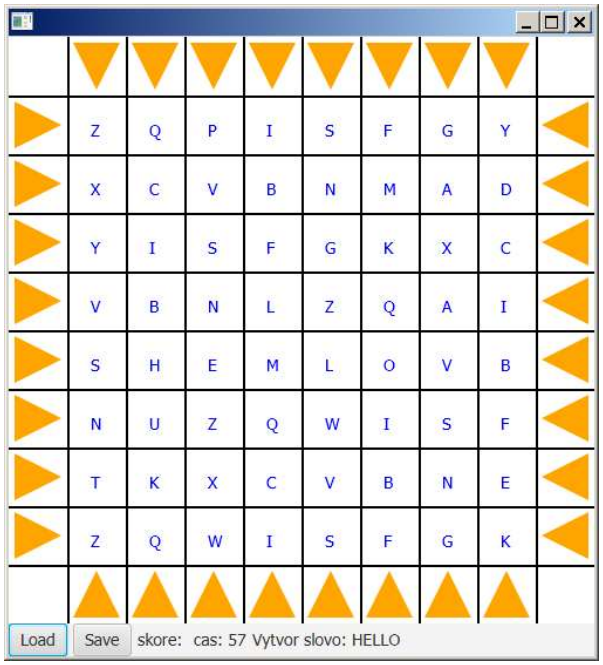
Bypass Excellencie

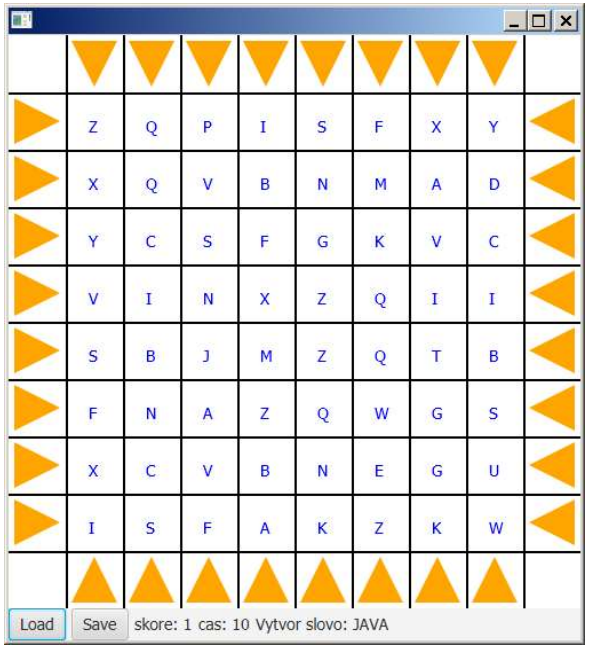


Niečo novšie

(**Písmenkovica**)

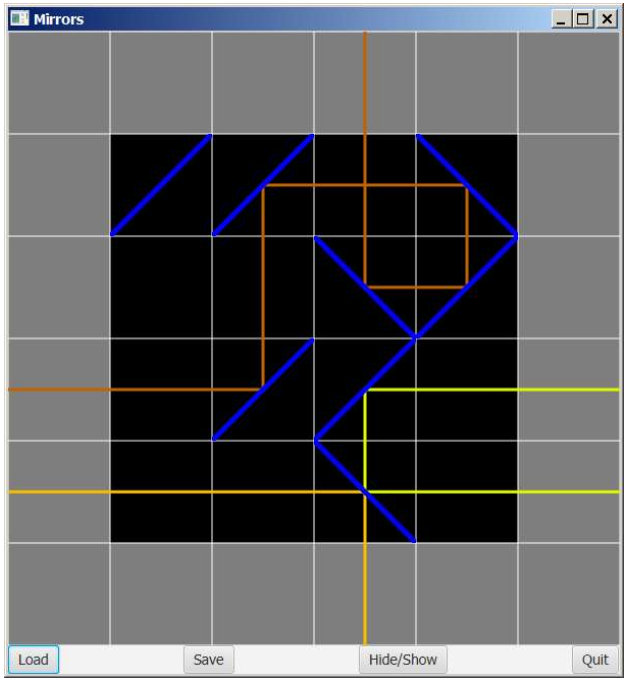
- V hre Písmenkovica sú v štvorcovej mriežke rozmiestnené písmená anglickej abecedy. Na okrajoch všetkých strán štvorca sú šípky. Ich stlačením dojde k otočeniu riadka alebo stĺpca o jedno písmenko podľa smeru šípky. Niekde v okne je zobrazené slovo, ktoré treba zo susedných písmen v mriežke vytvoriť: 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 nahradzujú písmená z vytvorených slov, sú uložené v súbore a na začiatku hry sa z neho načítajú. Formát súboru je nasledujúci …





Niečo novšie

(**Zrkadlová sieň**)

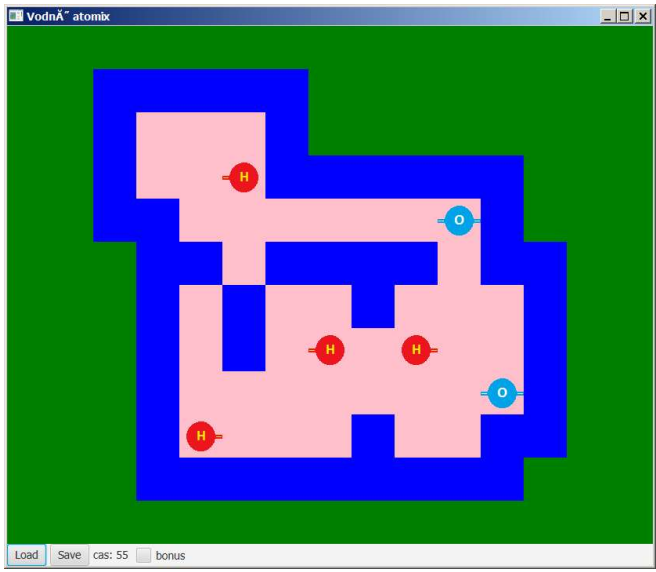


- V štvorcovej sále s rozmermi NxN sú v niektorých políčkach umiestnené diagonalne zrkadlá, v ilustráciach sú zobrazené modrou farbou. Môžu byť dvoch typov, / alebo \. Na kraji štvorcovej sály sú políčka, ktoré obsahujú zdroje svetla rôznych farieb. Krajné ľavé a pravé políčka (N+N) obsahujú vodorovný zdroj svetla, krajné horné a dolné políčka (N+N) obsahujú zvislý zdroj svetla. Rožné políčka (4) nemajú žiadnu funkciu. Pre jednoduchosť znázornenia scény plochu kreslíme do štvorcovej mriežky s rozmermi (N+2)x(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 (premyslite si...). Niektorým políčkom lúč opustí hraciu plochu, vy znázorňujete jeho cestu. Svetelné zdroje sú rôznych farieb, a farieb mate dosť (stačí si ich nejako 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.