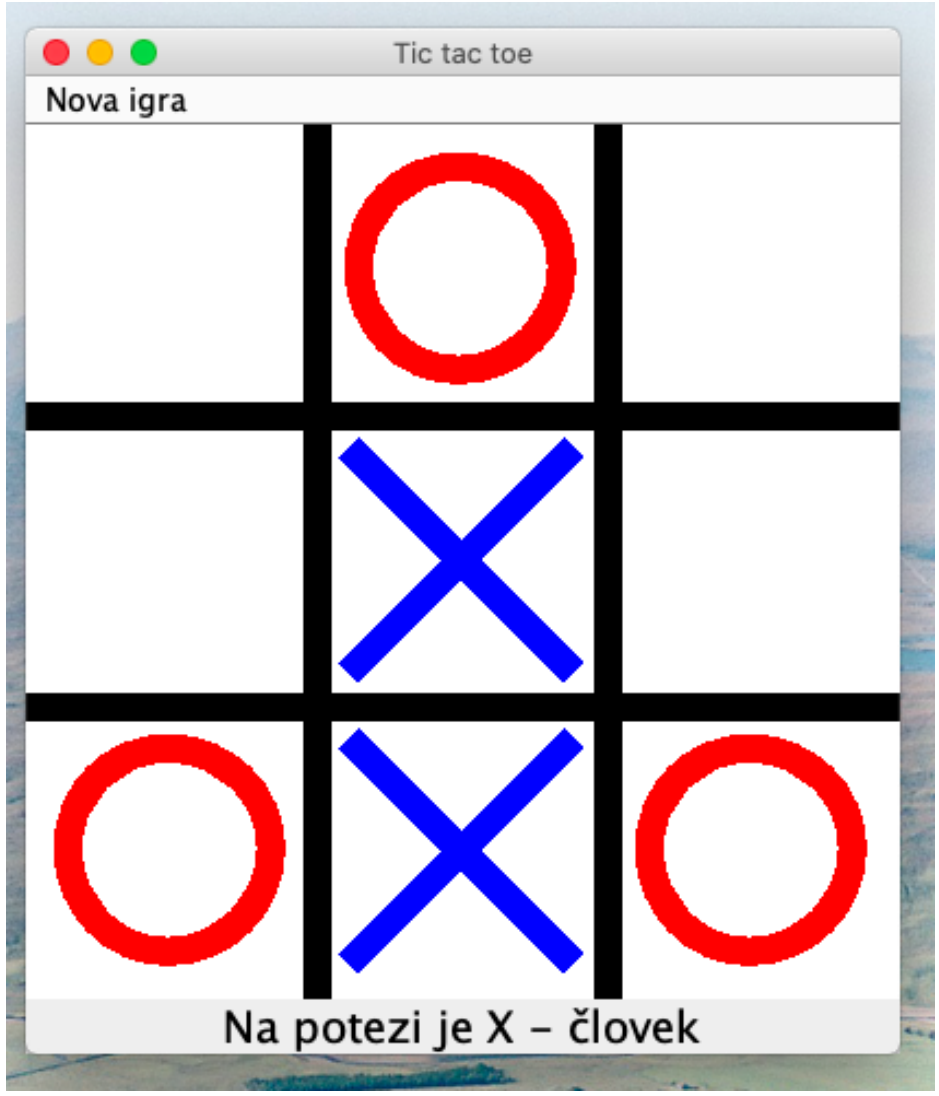
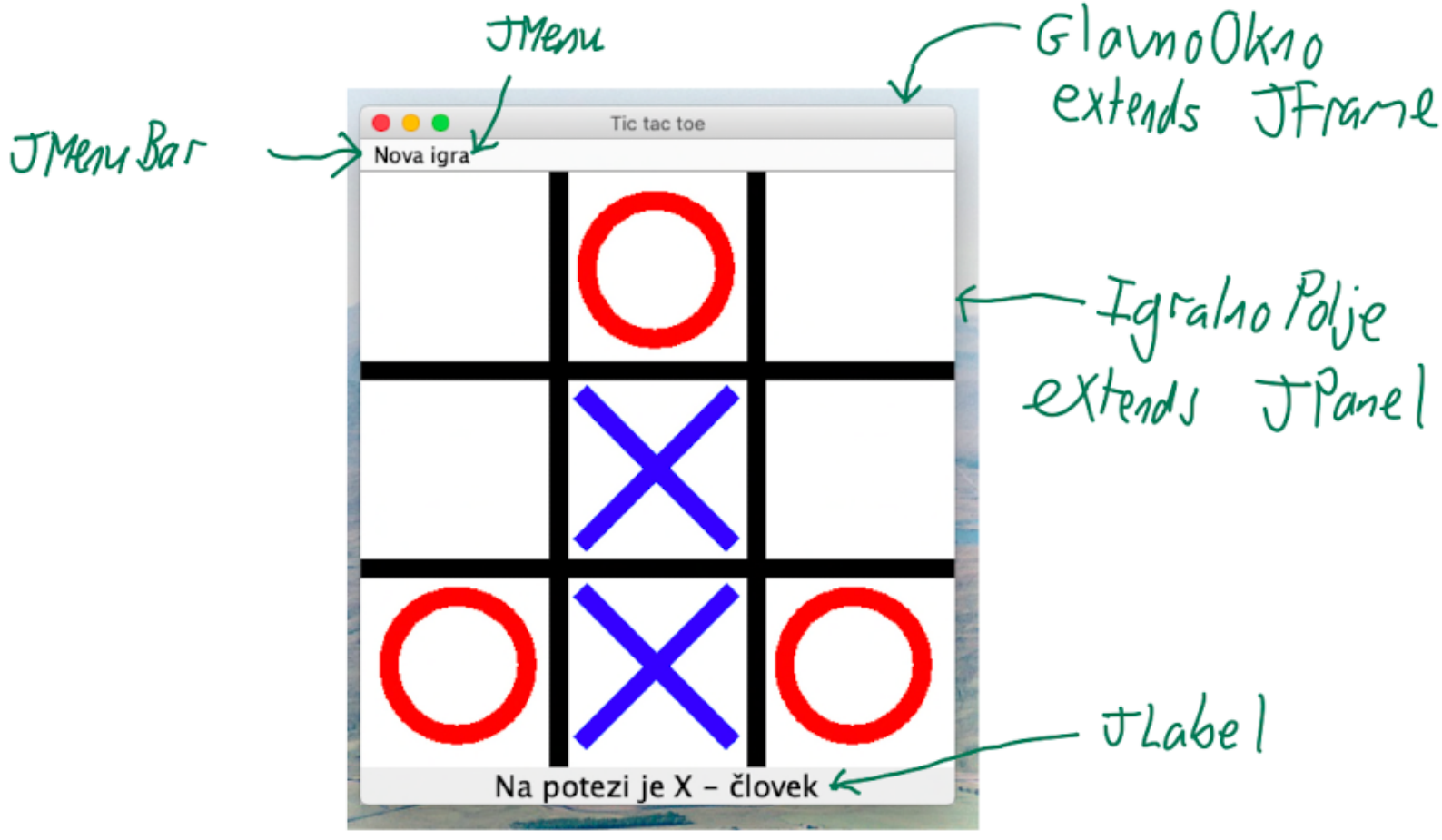
Igra z grafičnim vmesnikom

Aplikacija `TicTacToe`




JMenu
GlavnoOkno
extends JFrame
JMenuBar
Tic tac toe
Nova igra
IgralnoPolje
extends JPanel
JLabel
Na potezi je X – človek

Konstruktor razreda `GlavnoOkno` (1/3)

```
public GlavnoOkno() {

    this.setTitle("Tic tac toe");
    this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    this.setLayout(new GridBagLayout());

    // menu

    JMenuBar menu_bar = new JMenuBar();
    this.setJMenuBar(menu_bar);
    JMenu igra_menu = new JMenu("Nova igra");
    menu_bar.add(igra_menu);

    igraClovekRacunalnik = new JMenuItem("Človek - racunalnik");
    igra_menu.add(igraClovekRacunalnik);
    igraClovekRacunalnik.addActionListener(this);
    // podobno za druge elemente na meniju

    ...
```

- Uporabljamo `GridBagLayout`, ki je fleksibilen vpravitelj postavitve (layout manager) v Swingu.
- Vsak element na meniju mora imeti svoj akcijski poslušalec (action listener).

Konstruktor razreda `GlavnoOkno` (2/3)

```
...

// igralno polje
polje = new IgralnoPolje();

GridBagConstraints polje_layout = new GridBagConstraints();
polje_layout.gridx = 0;
polje_layout.gridy = 0;
polje_layout.fill = GridBagConstraints.BOTH;
polje_layout.weightx = 1.0;
polje_layout.weighty = 1.0;
getContentPane().add(polje, polje_layout);

...
```

- Damo `polje` po `GridBagLayoutu` na koordinati `gridx = 0, gridy = 0`.
- Vrednosti za `fill`, `weightx` in `weighty` pomenijo, da, če spremenimo velikost glavnega okna, spremenimo tudi velikost igralnega polja
- `getContentPane().add(component, constraints)` doda del `component` na okno z omejitvami `constraints`

Konstruktor razreda `GlavnoOkno` (3/3)

```
        ...

        // statusna vrstica za sporočila

        status = new JLabel();
        GridBagConstraints status_layout = new GridBagConstraints();
        status_layout.gridx = 0;
        status_layout.gridy = 1;
        status_layout.anchor = GridBagConstraints.CENTER;
        getContentPane().add(status, status_layout);

        status.setText("Izberite igro!");
    }
```

- Damo `polje` po `GridBagLayoutu` na koordinati `gridx = 0, gridy = 1`.
- Vrednost `CENTER` za `anchor` pomeni, da bo besedilo v centru statusne vrtice.

Razred GlavnoOkno

```
public class GlavnoOkno extends JFrame implements ActionListener {

    private IgralnoPolje polje;

    private JLabel status;

    private JMenuItem igraClovekRacunalnik;
    private JMenuItem igraRacunalnikClovek;
    private JMenuItem igraClovekClovek;
    private JMenuItem igraRacunalnikRacunalnik;

    public GlavnoOkno() { ... }

    @Override
    public void actionPerformed(ActionEvent e) { ... }

    public void osveziGUI() { ... }
}
```

Razred IgralnoPolje

```
public class IgralnoPolje extends JPanel implements MouseListener {

    public IgralnoPolje() {
        setBackground(Color.WHITE);
        this.addMouseListener(this);    }

    @Override
    public Dimension getPreferredSize() {
        return new Dimension(400, 400);    }


    @Override
    protected void paintComponent(Graphics g) { ... }

    @Override
    public void mouseClicked(MouseEvent e) { ... }


    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}
```

Metoda paintComponent (1/2)

```
private final static double LINE_WIDTH = 0.08; // relativna širina črte

// Sirina enega kvadratka
private double squareWidth() {
    return Math.min(getWidth(), getHeight()) / Igra.N; }

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D)g;
    double w = squareWidth();

    // crte
    g2.setColor(Color.BLACK);
    g2.setStroke(new BasicStroke((float) (w * LINE_WIDTH)));
    for (int i = 1; i < Igra.N; i++) {
        g2.drawLine((int)(i * w), (int)(0),
                    (int)(i * w), (int)(Igra.N * w));
        g2.drawLine((int)(0), (int)(i * w),
                    (int)(Igra.N * w), (int)(i * w));
    }
    ...
}
```

Metoda paintComponent (2/2)

```
private final static double PADDING = 0.18; // relativni prostor okoli X/O
private void paintO(Graphics2D g2, int i, int j) {
    double w = squareWidth();
    double d = w * (1.0 - LINE_WIDTH - 2.0 * PADDING); // premer O
    double x = w * (i + 0.5 * LINE_WIDTH + PADDING);
    double y = w * (j + 0.5 * LINE_WIDTH + PADDING);
    g2.setColor(Color.RED);
    g2.setStroke(new BasicStroke((float) (w * LINE_WIDTH)));
    g2.drawOval((int)x, (int)y, (int)d , (int)d);
}

protected void paintComponent(Graphics g) {
    ...
    // krizci in krožci
    Polje[][] plosca;;
    if (Vodja.igra != null) {
        plosca = Vodja.igra.getPlosca();
        for (int i = 0; i < Igra.N; i++) {
            for (int j = 0; j < Igra.N; j++) {
                switch(plosca[i][j]) {
                case X: paintX(g2, i, j); break;
                case O: paintO(g2, i, j); break;
                default: break;
}    }    }    }    }
```

Komentarji o metodi `paintComponent`

- `squareWidth` je metoda, ker mora biti vrednost dinamično izračunana, če spremenimo velikost okna.
- Izpustil sem kodo, ki pobarva ozadje zmagovalne terice.
- `N` je statično polje razreda `Igra`. Igramo igro velikosti `N`×`N`.
- Razred `Vodja` ima statično polje `igra` tipa `Igra`. Polje `igra` je igra, ki jo igramo. Uporabljamo njegovo metodo `getPlosca`, da bi videli trenutno stanje v igri.

  (Razred `Igra` je podoben razredu prejšnjega tedna, vendar ima nekaj novih metod, na primer `getPlosca`.)

Paketi v aplikaciji in njihovi razredi

- default package

  `TicTacToe` (ima metodo `main`)

- `vodja`

  `Vodja`, `VrstaIgralca`

- `gui`

  `GlavnoOkno`, `IgralnoPolje`

- `logika`

  `Igra`, `Igralec`, `Koordinati`, `Polje`, `Stanje`, `Vrsta`

Razred Igra

Konstruktor, ki ustvari novo igro.

```
Igra ()
```

Javni polja in metode

```
static final int N = 3 // velikost igralne plošče je NxN

Igralec naPotezi()

public Polje[][] getPlosca() // vrne tabelo igralne plošče

List<Koordinati> poteze() // vrne seznam možnih potez

Vrsta zmagovalnaVrsta() // vrne zmagovalno vrsto, če obstaja

Stanje stanje()

boolean odigraj(Koordinati p)
```

Metoda `odigraj(p)` odigra potezo `p` v igri in vrne `true`, če je poteza možna, sicer vrne `false`.

Razred Vodja (1/4) in metoda actionPerformed

V razredu Vodja:

```
public class Vodja {

    public static Map<Igralec,VrstaIgralca> vrstaIgralca;
    public static GlavnoOkno okno;
    public static Igra igra = null; // igra, ki jo trenutno igramo
    public static boolean clovekNaVrsti = false;
    ...
```

V razredu GlavnoOkno:

```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == igraClovekRacunalnik) {
        Vodja.vrstaIgralca =
            new EnumMap<Igralec,VrstaIgralca>(Igralec.class);
        Vodja.vrstaIgralca.put(Igralec.O, VrstaIgralca.C);
        Vodja.vrstaIgralca.put(Igralec.X, VrstaIgralca.R);
        Vodja.igramoNovoIgro();
    } else if (e.getSource() == igraRacunalnikClovek) { ...
    } else if (e.getSource() == igraClovekClovek) { ...
    } else if (e.getSource() == igraRacunalnikRacunalnik) { ... }
}
```

Razred Vodja (2/4)

```
public static void igramoNovoIgro () {
    igra = new Igra (); igramo ();        }

public static void igramo () {
    okno.osveziGUI();
    switch (igra.stanje()) {
    case ZMAGA_O:
    case ZMAGA_X:
    case NEODLOCENO:
        return;  // odhajamo iz metode igramo
    case V_TEKU:
        Igralec igralec = igra.naPotezi();
        VrstaIgralca vrstaNaPotezi = vrstaIgralca.get(igralec);
        switch (vrstaNaPotezi) {
        case C:
            clovekNaVrsti = true;
            break;
        case R:
            igrajRacunalnikovoPotezo ();
            break;
        }
    }
}
...
```

Metoda `osveziGUI`

V razredu `GlavnoOkno`:

```
public void osveziGUI() {
    if (Vodja.igra == null) { status.setText("Igra ni v teku."); }
    else {
        switch(Vodja.igra.stanje()) {
        case NEODLOCENO: status.setText("Neodločeno!"); break;
        case V_TEKU:
            status.setText("Na potezi je " + Vodja.igra.naPotezi + " - " +
                Vodja.vrstaIgralca.get(Vodja.igra.naPotezi));
            break;
        case ZMAGA_O:
            status.setText("Zmagal je O - " +
                Vodja.vrstaIgralca.get(Igralec.O));
            break;
        case ZMAGA_X:
            status.setText("Zmagal je X - " +
                Vodja.vrstaIgralca.get(Igralec.X));
            break;
        }
    }
    polje.repaint();
}
```

Razred `Vodja` (3/4)

```
...

private static Random random = new Random ();

public static void igrajRacunalnikovoPotezo() {
    List<Koordinati> moznePoteze = igra.poteze();
    int randomIndex = random.nextInt(moznePoteze.size());
    Koordinati poteza = moznePoteze.get(randomIndex);
    igra.odigraj(poteza);
    igramo ();
}

...
```

Računalnik igra brez inteligence!

Metoda mouseClicked in razred Vodja (4/4)

V razredu IgralnoPolje:

```
@Override
public void mouseClicked(MouseEvent e) {
    if (Vodja.clovekNaVrsti) {
        int x = e.getX(); int y = e.getY();
        int w = (int)(squareWidth());
        int i = x / w ; int j = y / w ;
        double di = (x % w) / squareWidth() ;
        double dj = (y % w) / squareWidth() ;
        if (0 <= i && i < Igra.N &&
                0.5 * LINE_WIDTH < di && di < 1.0 - 0.5 * LINE_WIDTH &&
                0 <= j && j < Igra.N &&
                0.5 * LINE_WIDTH < dj && dj < 1.0 - 0.5 * LINE_WIDTH) {
            Vodja.igrajClovekovoPotezo (new Koordinati(i, j));
}    }    }
```

V razredu Vodja:

```
    public static void igrajClovekovoPotezo(Koordinati poteza) {
        if (igra.odigraj(poteza)) clovekNaVrsti = false;
        igramo ();
    }
}
```

Komentarji o strukturi programa

- Vsa polja in metode razreda `Vodja` so statična.
  - Razred `Vodja` vodi samo eno igro v edinem oknu.
- Razred `Igra` ima konstruktor in dinamičnimi medotami.
  - Uporabljamo nov objekt `igra` razreda `Igra`, ko začnemo novo igro.
  - Ko pišemo kodo za računalnikovo inteligenco, bo bistveno, da lahko ustvarimo mnoge objekte razreda `Igra`.
- Paket `logika` je samostojni.
  - Paket `logika` se tiče samo logike igre. Razredom v paketu `igra` ni treba uporabljati razredov v paketih `vodja` in `gui`.

Razred TicTacToe

```
public class TicTacToe {

    public static void main(String[] args) {
        GlavnoOkno glavno_okno = new GlavnoOkno();
        glavno_okno.pack();
        glavno_okno.setVisible(true);
        Vodja.okno = glavno_okno;
    }

}
```

Niti v `Swingu`

- Metoda `main` se izvaja v začetni niti (initial thread).

  Ta nit samo inicializira okno.

- Potem program se izvaja pod vodjo `Swinga` v niti event dispatch thread (EDT).

- Če izvajanje programa potrebuje čas, ne bo EDT normalno funkcionirala, medtem ko program izračuna. Zato je pomembno, da izvajamo kodo, ki potrebuje čas, v niti v ozadju.

- Java ima dobro podporo za sočasno programiranje z nitmi. Na žalost, `Swing` ni varna z nitimi.

- Abstraktni razred `SwingWorker` podpora programiranje z `Swingom` z nitmi.

Niti z uporabo razreda SwingWorker

```
import javax.swing.SwingWorker;

SwingWorker<t, Void> worker =
        new SwingWorker<t, Void> () {
    @Override
    protected t doInBackground() {
```

Ta koda se izvaja v niti v ozadju. Mora vrniti vrednost tipa `t`.

```
    }
    @Override
    protected void done () {
```

Ta koda se izvaja, ko je koda v ozadju končana.
Metoda `get()` nam da vrednost tipa `t`, ki jo je vrnila koda v ozadju.
Metoda `get()` lahko vrži `ExecutionException` in `InterruptedException`.

```
    }
};
worker.execute();
```

Metoda igrajRacunalnikovoPotezo

```
public static void igrajRacunalnikovoPotezo() {
    Igra zacetkaIgra = igra;
    SwingWorker<Koordinati, Void> worker =
            new SwingWorker<Koordinati, Void> () {
        @Override
        protected Koordinati doInBackground() {
            try {TimeUnit.SECONDS.sleep(2);} catch (Exception e) {};
            List<Koordinati> moznePoteze = igra.poteze();
            int randomIndex = random.nextInt(moznePoteze.size());
            return moznePoteze.get(randomIndex);
        }
        @Override
        protected void done () {
            Koordinati poteza = null;
            try {poteza = get();} catch (Exception e) {};
            if (igra == zacetkaIgra && poteza != null) {
                igra.odigraj(poteza);
                igramo ();
            }
        }
    };
    worker.execute();
}
```

Bolj elegantna rešitev (brez uporabe `get()`)

```
public static void igrajRacunalnikovoPotezo() {
    Igra zacetkaIgra = igra;
    SwingWorker<Void, Void> worker = new SwingWorker<Void, Void> () {
        @Override
        protected Void doInBackground() {
            try {TimeUnit.SECONDS.sleep(2);} catch (Exception e) {};
            return null;
        }
        @Override
        protected void done () {
            if (igra != zacetkaIgra) return;
            List<Koordinati> moznePoteze = igra.poteze();
            int randomIndex = random.nextInt(moznePoteze.size());
            Koordinati poteza = moznePoteze.get(randomIndex);
            igra.odigraj(poteza);
            igramo ();
        }
    };
    worker.execute();
}
```