STU FIIT
SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
VGG
Vision & Graphics Group
Institute of Computer Engineering and Applied Informatics

Základy tvorby interaktívnych aplikácií

Princípy interaktívnych aplikácií

Ing. Peter Kapec, PhD.

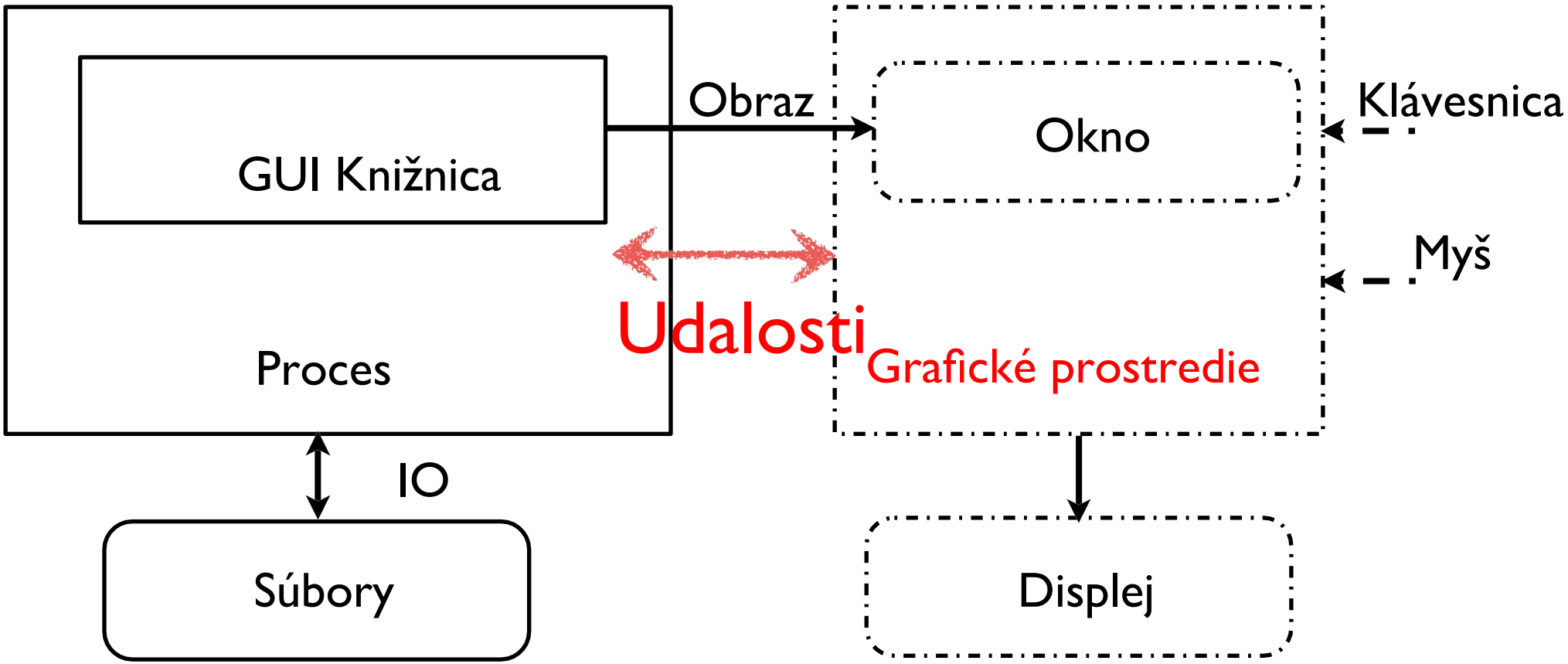
LS 2020-21

Obsah

- GUI aplikácie a interakcia
- Udalosti a ich spracovanie
- JavaScript spracovanie vstupov
- Spracovanie udalostí v desktopových GUI knižniciach

Udalosti a ich spracovanie

Typická GUI aplikácia



Aké Udalosti ?

- Spracovanie udalostí zo systému
  - Klávesnica, myš
  - Zatvorenie okna
  - Zmena veľkosti okna
  - ...
- Udalosti => spracovanie správ

Správy a ich spracovanie

- Správy sú zasielané aplikácii systémom
- Prichádzajúce správy sú zoradené vo fronte
- Aplikácia obsahuje mechanizmus spracovania
- Vyberá správu z fronty a spracuje ju
- Zvyčajne spracúva správy sekvenčne v cykle
- Komplexnejšie aplikácie spracúvajú udalosti v samostatnom vlákne

Správy a ich spracovanie

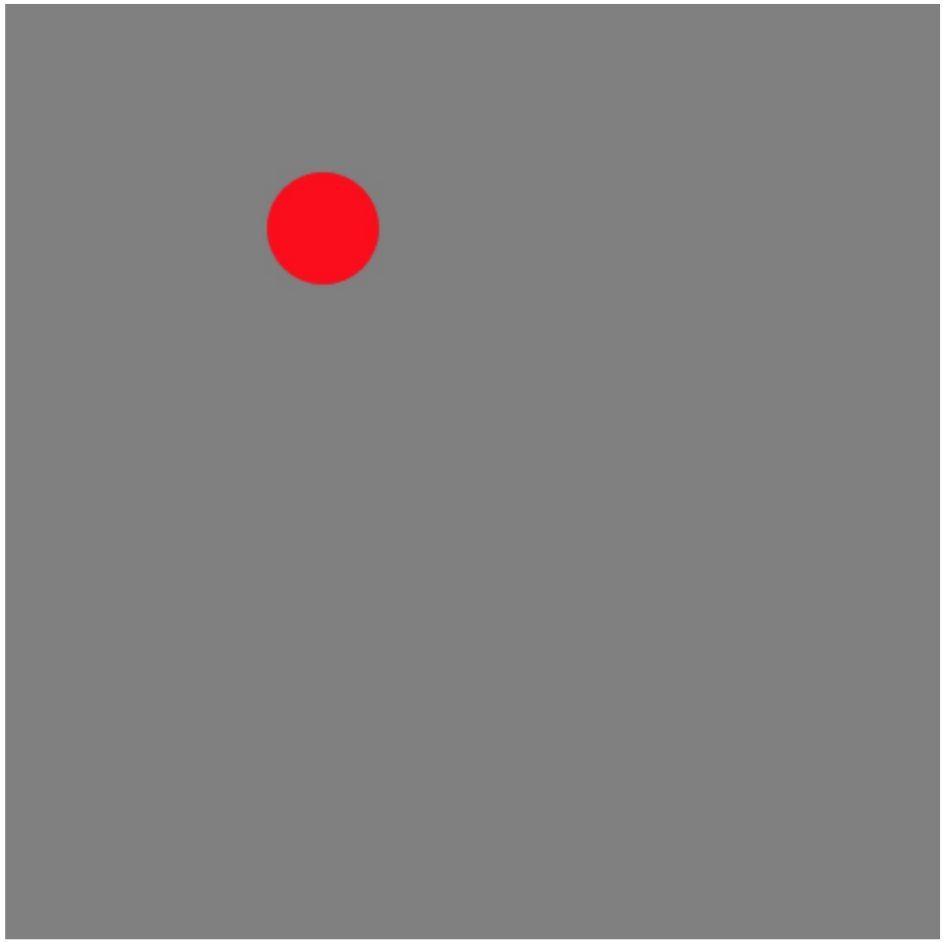
- Spracovanie udalostí v cykle (rôzne názvy angl. *event loop, message dispatcher, message loop, main event loop*):

```
function main
    initialize()
    while message != quit
        message := get_next_message()
        process_message(message)
    end while
end function
```

JavaScript spracovanie vstupov z klávesnice a myši

Ako spracovať vstup z klávesnice

- Zachytenie udalostí stlačenia kláves
  - Zápis stlačeného klávesu do zoznamu
  - Čítanie stavu v hlavnom cykle aplikácie
- Zachytenie udalosti pustenia kláves
  - Odstránenie záznamu zo zoznamu



HTML

- definuje *Canvas*, využíva *keyboard.js*

```
<html>

  <head>
    <title>Keyboard Ball</title>
    <script src="keyboard.js"></script>
    <style>
      #canvas {
        border: 0 solid;
      }
    </style>
  </head>

  <body>
    <p>
      <canvas id="canvas" height="500" width="500"></canvas>
    </p>
  </body>

</html>
```

Globálne premenné, inicializácia

- Model: objekt pre stlačené klávesy, objekt pre loptu

```
var canvas;
var ctx;
var tick = 0;

// Model
var keys = {};
var ball = { x: 50, y: 50 };

// View / Controller

// Initialization
window.onload = function() {
  // Setup global variables for easy access
  button = document.getElementById("button");
  canvas = document.getElementById("canvas");
  ctx = canvas.getContext("2d");

  requestAnimationFrame(mainLoop);
};
// Input
```

Spracovanie vstupov z klávesnice

- zadefinujeme funkcie zachytávajúce
  - stlačenie klávesy - udalosť *onkeydown*
  - pustenie klávesy - udalosť *onkeyup*
- kód stlačenej klávesy vložíme ako atribút do objektu *keys* (aj viacero súčasne stlačených)

```
// Input
// Handle keyboard events
window.onkeydown = function(event) {
  keys[event.keyCode] = true;
  console.log(keys);
};
window.onkeyup = function(event) {
  keys[event.keyCode] = false;
};
```

View

- zobrazí červený kruh na šedom pozadí

```
// View
ball.draw = function() {
  // Clear canvas
  ctx.fillStyle = "gray";
  ctx.fillRect(0, 0, canvas.width, canvas.height);

  // Render a circle
  ctx.fillStyle = "red";
  ctx.beginPath();
  ctx.arc(ball.x, ball.y, 30, 0, Math.PI * 2, true);
  ctx.closePath();
  ctx.fill();
};
```

Controller

- hlavná slučka programu: posunutie a vykreslenie
- *ball.move()* podľa stlačenej klávesy aktualizuje pozíciu objektu (prečo nie *if...else if… ?*)

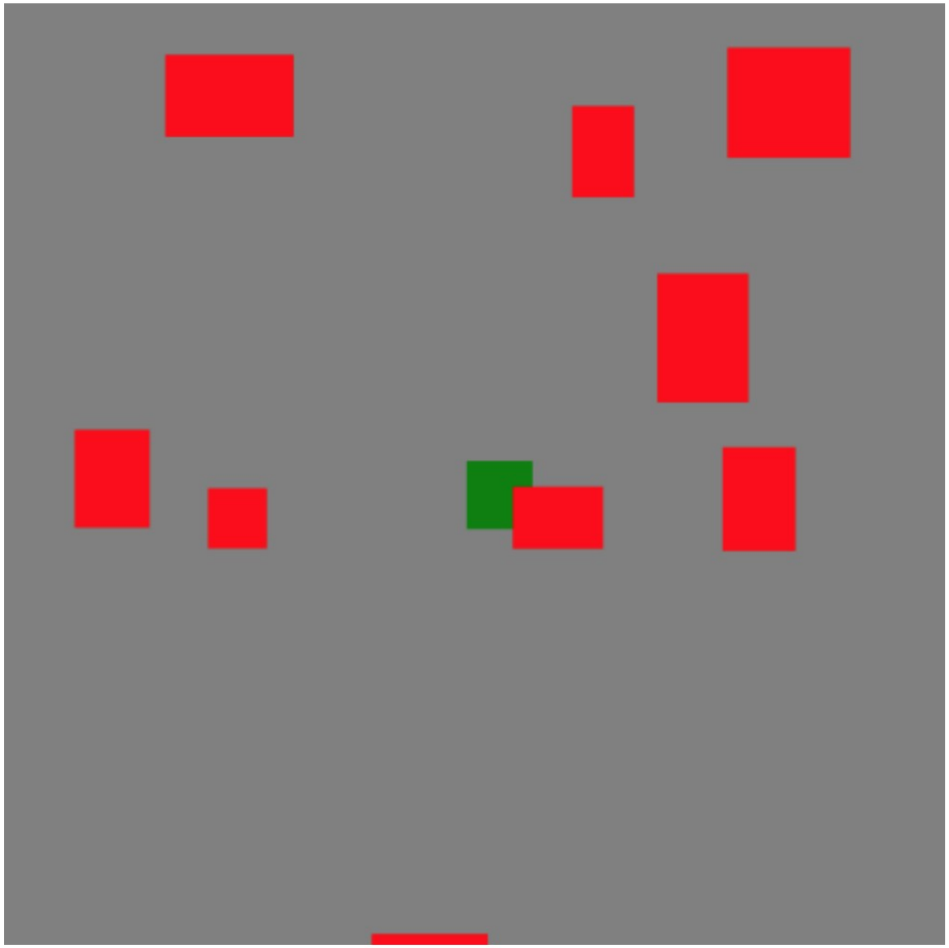
```
// Controller
function mainLoop() {
  tick++;
  ball.move();
  ball.draw();
  requestAnimationFrame(mainLoop);
}

ball.move = function() {
  if (keys[37]) ball.x -= 5;  // sipka vlavo
  if (keys[39]) ball.x += 5;  // sipka vpravo
  if (keys[38]) ball.y -= 5;  // sipka hore
  if (keys[40]) ball.y += 5;  // sipka dole
};
```

Spracovanie vstupov z klávesnice

- Pozri implementáciu v súboroch keyboard.html a keyboard.js

Ako spracovať myš

- Objektom v scéne možno definovať *onclick* metódu
- Pri stlačení myši na elemente canvas zachytíme X a Y koordináty
- Prejdeme objektami v scéne a zistíme, ktoré boli vybraté
- Zavoláme metódu *onclick* daného objektu



HTML a JavaScript

```
<html>

  <head>
    <title>Click the Squares</title>
    <script src="click.js"></script>
    <style>
      #canvas {
        border-style: solid;
        border-width: 0px;
      }
    </style>
  </head>

  <body>
    <p>
      <canvas id="canvas" height="500" width="500"></canvas>
    </p>
  </body>

</html>
```

Globálne premenné a inicializácia

```
var canvas
var ctx
var timer
var image
var tick = 0

// Model / View / Controller

// Initialization
window.onload = function() {
  // Get canvas and context
  canvas = document.getElementById("canvas")
  canvas.onclick = mouseClick     // Handle onclick event via mouseClick()

  ctx = canvas.getContext("2d")

  // Create 10 squares
  for (i = 0; i < 10; i++) {
    squares.push( new Square() )
  }
  requestAnimationFrame(mainLoop)
}
```

Model

- globálne pole *squares* pre objekty *Square*
- konštruktor nastaví náhod. pozíciu, smer pohybu, šírku a výšku obdĺžnika, farbu

```
// Model
var squares = []

// Constructor for the square object
function Square() {
  this.x = Math.random() * canvas.width;
  this.y = Math.random() * canvas.height;
  this.dx = Math.random() * 10 - 5;
  this.dy = Math.random() * 10 - 5;
  this.width = Math.random() * 40 + 30;
  this.height = Math.random() * 40 + 30;
  this.color = "red";
}
```

Metódy *Square* objektu

- Pohyb

```
// Common properties for all squares
Square.prototype = {
  move: function() {
    // Movement logic
    if (this.x >= canvas.width || this.x <= 0) {
      this.dx *= -1
    }
    if (this.y >= canvas.height || this.y <= 0) {
      this.dy *= -1
    }

    // Posun
    this.x = this.x + this.dx
    this.y = this.y + this.dy
  },
  // continue on next slide
```

Metódy *Square* objektu

- Vykreslenie
- *onclick()* - zmení farbu (metódu zavolá Controller pri spracovaní udalosti stlačenie tlačítka na myši)

```
  // Draw itself on canvas
  draw: function() {
    ctx.fillStyle = this.color
    ctx.beginPath()
    ctx.fillRect(this.x, this.y, this.width, this.height)
    ctx.closePath()
  },
  // Function to call when clicked
  onclick: function() {
    this.color = "green"
  }
}
```

View

- Štandardné zobrazenie všetkých objektov v poli *squares*

```
// View
function display() {
  // Clear canvas
  ctx.fillStyle = "gray"
  ctx.fillRect(0, 0, canvas.width, canvas.height)
  // Draw all squares
  for (i in squares) {
    squares[i].draw()
  }
}
```

Controller

- Posúva všetky objekty v poli *squares* volaním metódy *move()* príslušného objektu
- Hlavná slučka

```
// Controller
function move() {
  for (var i in squares) {
    squares[i].move()
  }
}

function mainLoop() {
  tick++
  move()
  display()
  requestAnimationFrame(mainLoop)
}
```

Spracovanie kliku myšou

- Získanie reálnej pozície [x,y] v Canvas-e
- V cykle prejdeme cez všetky objekty v poli *squares*, ak je pozícia myši vo vnútri *Square* objektu, zavoláme jeho *onclick()* metódu

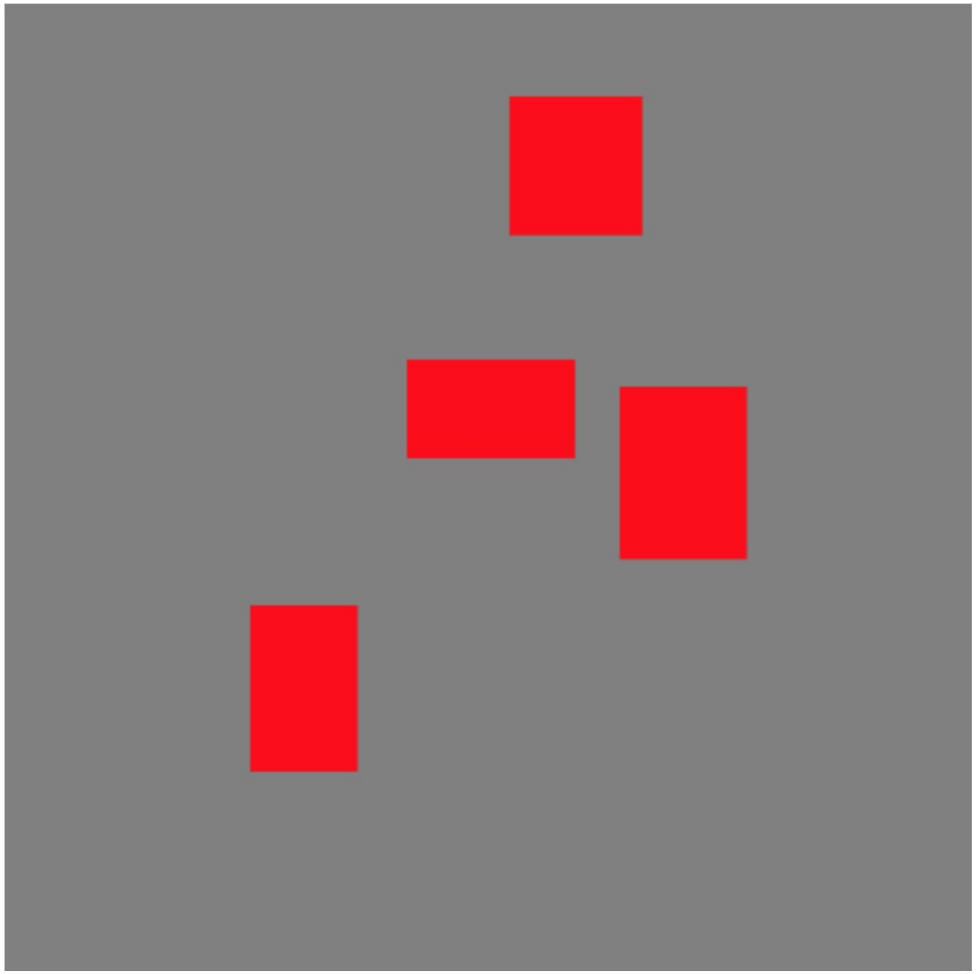
```
function mouseClick(event) {
  var x = event.pageX - canvas.offsetLeft
  var y = event.pageY - canvas.offsetTop

  // Test each square for click
  for (var i in squares) {
    var square = squares[i]
    if (x > square.x && x < square.x + square.width &&
        y > square.y && y < square.y + square.height) {
        square.onclick()
    }
  }
}
```

Spracovanie vstupov z myši

- Pozri implementáciu v súboroch click.html a click.js

Kombinácia ovládania

- Spracujeme udalosť *onclick* pre výber objektu
- Prejdeme objektami v scéne a zistíme, ktorý bol vybraný
- Zachytávame X a Y koordináty myši počas pohybu myši a aktualizujeme pozíciu vybraného objektu



HTML a JavaScript

```
<html>

  <head>
    <title>Click and Drag the Squares</title>
    <script src="drag.js"></script>
    <style>
      #canvas {
        border-style: solid;
        border-width: 0px;
      }
    </style>
  </head>

  <body>
    <p>
      <canvas id="canvas" height="500" width="500"></canvas>
    </p>
  </body>

</html>
```

Inicializácia, spracovanie udalostí

```
var canvas
var ctx
var timer
var image
var tick = 0

// Model / View / Controller

// Initialization
window.onload = function() {
  // Setup global variables
  canvas = document.getElementById("canvas")
  canvas.onmousedown = mousedown  // Handle mousedown event
  canvas.onmouseup = mouseup      // Handle mouseup event
  canvas.onmousemove = mousemove  // Handle mousemove event

  ctx = canvas.getContext("2d")

  scene.push( new Window() )
  scene.push( new Window() )
  scene.push( new Window() )
  scene.push( new Window() )
  requestAnimationFrame(mainLoop);
}
```

Model

- objekt *mouse*: pozícia kurzora, či je na myši stlačené tlačidlo, či je vybraný objekt
- pole *scene* pre objekty *Window* (obdĺžniky)

```
// Model
var mouse = { x: 0, y: 0, pressed: false, selected: false}
var scene = []

function Window() {
  // Constructor for rectangular area
  this.x = Math.random() * canvas.width;
  this.y = Math.random() * canvas.height;
  this.width = Math.random() * 40 + 50;
  this.height = Math.random() * 40 + 50;
  this.color = "red";
}
```

Model

- Metódy *Window* objektu pre
  - *draw()* - vykreslenie
  - *setPosition()* - nastavenie pozície

```
Window.prototype = {
  // Draw self using a rectangle
  draw: function() {
    ctx.fillStyle = this.color
    ctx.beginPath()
    ctx.fillRect(this.x, this.y, this.width, this.height)
    ctx.closePath()
  },
  setPosition: function(x, y) {
    this.x = x
    this.y = y
  }
}
```

View a Controller

- štandardné vykreslenie všetkých objektov

```
// View
function display() {
  // Clear the canvas
  ctx.fillStyle = "gray"
  ctx.fillRect(0, 0, canvas.width, canvas.height)

  // Render all objects in scene
  for (i in scene) {
    scene[i].draw()
  }
}
```

- iba vykreslenie objektov, keďže sa sami nehýbu

```
// Controller
function mainLoop() {
  tick++
  display()
  requestAnimationFrame(mainLoop);
}
```

Spracovanie vstupu z myši

- pri stlačení myši sa nájde *Window* podľa pozície kurzora a veľkosti objektu *Window*
- poznačí sa, ktorý objekt bol vybraný

```
// Handle mouse interaction
function mousedown(event) {
  mouse.pressed = true
  for(i in scene) {
    var window = scene[i]
    if(window.x < mouse.x && window.x + window.width > mouse.x &&
      window.y < mouse.y && window.y + window.height > mouse.y) {
        // store the selected window and change its color
        mouse.selected = window
        mouse.selected.color = "green"
        break
      }
  }
}
```

Spracovanie vstupu z myši

- pri pohybe sa nastaví pozícia objektu cez *setPosisiton()*
- pri pustení tlačidla myši sa resetuje výber objektu

```
// Handle mouse movement
function mousemove(event) {
    mouse.x = event.pageX - canvas.offsetLeft
    mouse.y = event.pageY - canvas.offsetTop
    // mouse.selected contains now a Window object,
    // so we can call this object's method setPossition()
    if(mouse.selected) mouse.selected.setPosition(mouse.x, mouse.y)
}

// Handle mouse release
function mouseup(event) {
  mouse.pressed = false
  if(mouse.selected) mouse.selected.color = "red"
  mouse.selected = false    // remove the selected window
}
```

Spracovanie vstupov z myši

- Pozri implementáciu v súboroch drag.html a drag.js

Spracovanie udalostí v desktopových GUI knižniciach

GUI Knižnice

- Zjednodušujú obsluhu systémovo špecifických správ, ktoré musí aplikácia spracovať
- Poskytujú sadu objektov pomocou ktorých možno vytvoriť grafické rozhranie
- Poskytujú základnú implementáciu ovládacích prvkov aplikácie
- Zväčša umožňujú špecifikovať rozhrania mimo kódu za pomoci vizuálneho editora

Príklad: Win32 Okno

- Minimálne okno
- Nastav parametre aplikácie
- Nastav parametre okna
- Nastav funkciu spracovania udalostí
- V cykle spracúvaj správy
- **VAROVANIE**: Nasleduje Win32 C kód ...

Win32 - vytvorenie okna, event loop

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
  MSG msg = { 0 };
  WNDCLASS wc = { 0 };
  wc.lpfnWndProc = WndProc; // Funkcia spracovania spravy
  wc.hInstance = hInstance;
  wc.hbrBackground = (HBRUSH)(COLOR_BACKGROUND);
  wc.lpszClassName = "minwindowsappi";
  if (!RegisterClass(&wc))
    return 1;
  if (!CreateWindow(wc.lpszClassName,
          "Win32 Application",
          WS_OVERLAPPEDWINDOW | WS_VISIBLE,
          100, 100, 320, 320, 0, 0, hInstance, NULL))
    return 2;

  // Cyklus obsluhujuci spravy
  while (GetMessage(&msg, NULL, 0, 0) > 0)
    DispatchMessage(&msg);
  return 0;
}
```

Win32 - event handling

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message) {

  case WM_LBUTTONDOWN:
    int x = GET_X_LPARAM(lParam);
    int y = GET_Y_LPARAM(lParam);
    printf("MouseEvent: button=left x=%d y=%d\n", x, y);
    break;
  case WM_KEYDOWN:
    int key = wParam;
    printf("KeyboardEvent: key=%d\n", key);
    if(key == VK_ESCAPE) PostQuitMessage(0);
    break;
  case WM_CLOSE:
    PostQuitMessage(0);
    break;
  default:
    return DefWindowProc(hWnd, message, wParam, lParam);
  }
  return 0;
}
```

GLUT

- Vytvorenie okna
- Zaregistrovanie tzv. *call-back* funckií, ktoré event loop volá pre dané udalosti

```
int main(int argc, char* argv[]) {
  glutInit(&argc, argv);
  glutCreateWindow("Glut Window");
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(320, 320);

  glutMouseFunc(mouseEventHandler);
  glutKeyboardFunc(keyboardEventHandler);
  glutDisplayFunc(displayEventHandler);

  glutMainLoop();
}
```

GLUT

- Ukážky implementácie *call-back* funkcií

```
void mouseEventHandler(int button, int state, int x, int y) {
  printf("MouseEvent: button=%d state=%d x=%d y=%d\n", button, state, x, y);
}

void keyboardEventHandler(unsigned char key, int x, int y) {
  printf("KeyboardEvent: key=%d, x=%d, y=%d\n", key, x, y);
  if(key == 27) exit(0);
}

void displayEventHandler() {}
```

GTK

- Vytvorenie okna
- Zadefinovanie *signálov* a *call-back* funkcií

```
int main(int argc, char* argv[]) {
  gtk_init(&argc, &argv);

  GtkWidget *window;
  window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
  gtk_window_set_title(GTK_WINDOW(window), "GTK Window");
  gtk_window_move(GTK_WINDOW(window), 100, 100);
  gtk_window_set_default_size(GTK_WINDOW(window), 320, 320);
  gtk_widget_add_events(window, GDK_BUTTON_PRESS_MASK);

  g_signal_connect(window, "key-press-event", G_CALLBACK(keyEventHandler), NULL);
  g_signal_connect(window, "button-press-event", G_CALLBACK(mouseEventHandler), NULL);

  gtk_widget_show(window);
  gtk_main();
  return 0;
}
```

GTK

- Ukážky implementácie *call-back* funkcií

```
int mouseEventHandler(GtkWidget *window, GdkEventButton *event, GdkWindowEdge edge)
{
  printf("MouseEvent: button=%d, x=%f, y=%f\n", event->button, event->x, event->y);
  return 0;
}

int keyEventHandler(GtkWidget *widget, GdkEventKey *event) {
  printf("KeyboardEvent: key=%d\n", event->keyval);
  if(event->keyval == 65307) gtk_main_quit();
  return 0;
}
```

Qt

- Vytvorenie okna aplikácie

```
int main(int argc, char *argv[]) {
  QApplication app(argc, argv);

  MyWindow window;

  window.show();
  return app.exec();
}
```

Qt

Implementácia okna ako triedy, definovanie spracovanie udalostí (*event*)

```
class MyWindow : public QMainWindow {
  Q_OBJECT

public:
  MyWindow() : QMainWindow() {
    setWindowTitle("Qt Window");
    move(100, 100);
    setFixedSize(320, 320);
  }

protected:
  void mousePressEvent(QMouseEvent *event) {
    std::cout << "MouseEvent button=" << event->button()
              << " x=" << event->x() << " y=" << event->y() << std::endl;
  }
  void keyPressEvent(QKeyEvent *event) {
    std::cout << "KeyboardEvent key=" << event->key() << std::endl;
    if(event->key() == Qt::Key_Escape) close();
  }
};
```

Zhrnutie

- Klúčové poznatky z prednášky
  - Interakcia s GUI aplikáciou je reprezentovaná udalosťami
  - Udalosti sú spracovávané v cykle (angl. *event loop*)
  - Ukážka spracovania vstupov v JavaScript
  - Spracovanie udalostí v desktopových GUI knižniciach
    - Rôzne variácie spracovania udalostí v cykle

Nabudúce

- Návrhový vzor Observer
  - Rozšírenie vzoru na všeobecné použitie
- Šírenie udalostí v komplexných GUI aplikáciách
  - GUI Widgets a ich ukážková implementácia v JS
- Využitie OOP dedenia
- Komplexné GUI knižnice

Ďakujem za pozornosť