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

Interaktívne hry

Ing. Peter Kapec, PhD.

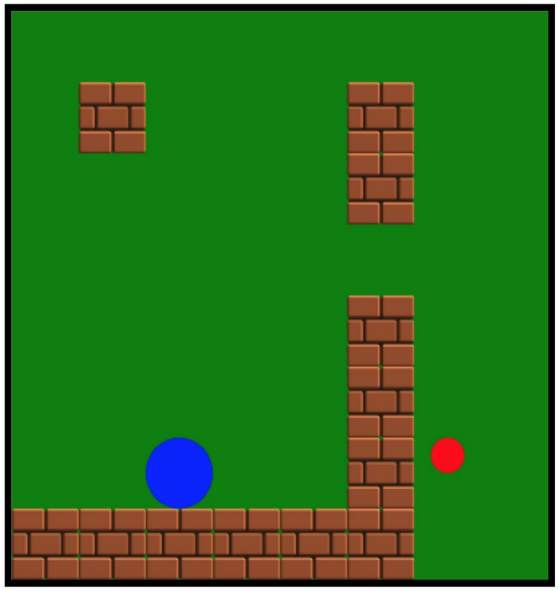
LS 2020-21

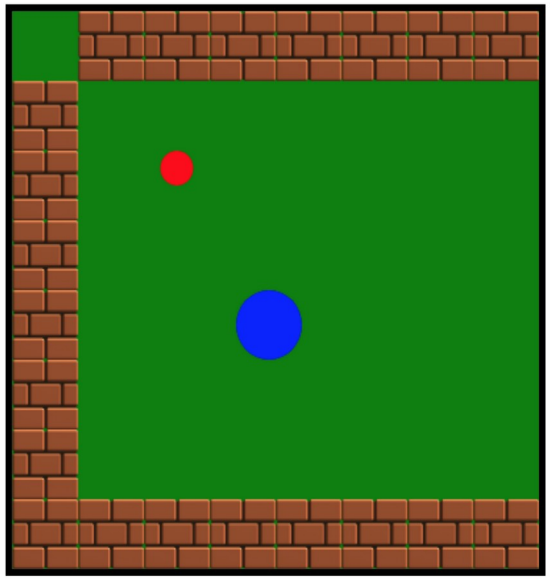
Obsah

- Interaktívne hry
  - Kolízie objektov
  - Dynamické pridávanie objektov
  - Zvuk

Kolízie objektov

- Cieľ:
  - Pridať do scény hráča
    - Ovládaný klávesnicou
  - Vytvoriť 2 levely
    - Rôzne mapy
    - Prechod z jednej mapy do druhej (zmena scény)
  - Detekovať kolíziu hráča
    - So stenami
    - S objektom





HTML

```
<!DOCTYPE html>
<html>

  <head>
    <title>Collisions</title>
    <meta charset="utf-8" />
    <link href="css/game.css" rel="stylesheet" />
    <script src="js/gameobject.js"></script>
    <script src="js/background.js"></script>
    <script src="js/block.js"></script>
    <script src="js/player.js"></script>
    <script src="js/finish.js"></script>
    <script src="js/level1.js"></script>
    <script src="js/level2.js"></script>
    <script src="js/game.js"></script>
    <script src="js/main.js"></script>
  </head>

  <body>
    <p>
      <canvas id="canvas" height="400" width="400">No Canvas :(</canvas>
    </p>
    <img id="block" src="img/block.png" hidden="true" />
  </body>

</html>
```

CSS

- pred-načítanie obrázkov

```
#canvas {
  border-style: solid;
  border-width: 5px;
}
#preload-block { background: url(img/block.png) no-repeat -9999px -9999px; }
```

Gameobject.js

- základné atribúty a metódy

```
class GameObject {
  constructor(x, y, size) {
    // Constructor, generates a new GameObject
    this.x = x;
    this.y = y;
    this.size = size;
    this.physical = true;
  }

 // Move self
  move(game) {}

  // Draw self
  draw(game) { }

  // continue next slide
```

Gameobject.js

- detekcia kolízií medzi *gameobject* “hrubou” silou
  - objekt vs všetky ostatné objekty v scéne

```
  // Check object collision
  checkCollision(scene) {
    // Test collision
    for (var i in scene) {
      var obj = scene[i];
      // Object is not physical
      if (obj == this || !obj.physical) continue;
      // test boundaries
      var test =
        this.x >= obj.x + obj.size ||
        this.x + this.size <= obj.x ||
        this.y >= obj.y + obj.size ||
        this.y + this.size <= obj.y;

      // if collision, then return the hit object
      if (!test) {
        return obj;
      }
    }
    return false;
  }
} // end class
```

Backround.js

- *physical* - nebude testovaný na kolíziu
- *move()* prázdny, *draw()* vykreslí celé pozadie

```
class Background extends GameObject {
  constructor() {
    super(0, 0, 0);
    this.physical = false;
  }

  move(game) {}

  draw(game) {
    game.context.fillStyle = "green";

    // fill whole canvas
    game.context.fillRect(0, 0, game.canvas.width, game.canvas.height);
  }
}
```

Block.js

- reprezentuje jednu stenu

```
class Block extends GameObject {
  constructor(x, y) {
    var size = 50;
    super(x * size, y * size, size);
    this.img = document.getElementById("block");
  }

  draw(game) {
    game.context.drawImage(this.img, this.x, this.y, this.size, this.size);
  }
}
```

Player.js

- *move()* si odloží pozíciu, posunie sa podla vstupu, skontroluje novú pozíciu na kolízie, v prípade kolízie nastaví odloženú pozíciu

```
class Player extends GameObject {
  constructor(x, y) {
    var size = 50;
    super(x * size, y * size, size);
  }

  move(game) {
    var last_x = this.x;
    var last_y = this.y;
    // Move according to pressed keys
    if (game.keys[37]) this.x -= 5;
    if (game.keys[39]) this.x += 5;
    if (game.keys[38]) this.y -= 5;
    if (game.keys[40]) this.y += 5;
    // Reset position if collision occurs
    if (this.checkCollision(game.scene)) {
      this.x = last_x;
      this.y = last_y;
    }
  }
```

Player.js

- *draw()* - iba vykreslenie

```
  draw(game) {
    var ctx = game.context;
    ctx.fillStyle = "blue";
    ctx.beginPath();
    ctx.arc(this.x + this.size/2, this.y + this.size/2, this.size/2, 0, Math.PI * 2);
    ctx.closePath();
    ctx.fill();
  }
}
```

Finish.js

- testuje kolíziu s hráčom (neoptimálne riešenie !!!)
- nastaví scénu na ďalší level

```
class Finish extends GameObject {
  constructor(x, y) {
    var size = 50;
    super(x * size, y * size, size);
    this.physical = false;
  }

  move(game) {
    // If the player reaches the finish
    var obj = this.checkCollision(game.scene);

    // check if obj is Player
    if(obj instanceof Player) {
      // load next level
      game.level++;
      // dynamically generate function name and call it
      // and change scene
      game.scene = eval("level"+game.level+"();");
    }
  }

  draw(game) { ... }
}
```

Level1.js

- vytvorí jednotlivé objekty scény, vloží ich do poľa a vráti ich

```
level1 = function() {
  return [
      new Background(), // !!! the order is important !!!
      new Block(0, 7),  // objects should be displayed
      new Block(1, 7),  // from back to front
      new Block(2, 7),
      new Block(3, 7),
      new Block(4, 7),
      new Block(5, 7),
      new Block(5, 7),
      new Block(5, 6),
      new Block(5, 5),
      new Block(5, 4),
      new Block(5, 2),
      new Block(5, 1),
      new Block(1, 1),
      new Player(2, 6),
      new Finish(6, 6)
  ];
};
```

Level2.js

- vytvorí jednotlivé objekty scény, vloží ich do poľa a vráti ich

```
level2 = function() {
  return [
      new Background(),
      new Block(1, 7),
      new Block(2, 7),
      new Block(3, 7),
      new Block(4, 7),
      new Block(5, 7),
      new Block(6, 7),
      new Block(7, 7),
      new Block(1, 0),
      new Block(2, 0),
      new Block(3, 0),
      new Block(4, 0),
      new Block(5, 0),
      new Block(6, 0),
 ...
```

```
 ...
      new Block(7, 0),
      new Block(0, 1),
      new Block(0, 2),
      new Block(0, 3),
      new Block(0, 4),
      new Block(0, 5),
      new Block(0, 6),
      new Block(0, 7),
      new Block(8, 1),
      new Block(8, 2),
      new Block(8, 3),
      new Block(8, 4),
      new Block(8, 5),
      new Block(8, 6),
      new Block(8, 7),
      new Player(4, 4),
      new Finish(2, 2)
  ];
};
```

Game.js

- *Game* reprezentuje celú aplikáciu
- Model
  - scéna, vytvorená v konštruktore
- Controller
  - sprac. vstupov
- Event loop
  - move()
  - draw()

```
class Game {
  constructor(canvasName) {
    this.canvas = document.getElementById(canvasName);
    this.context = canvas.getContext("2d");

    // Model
    this.level = 1;
    this.keys = [];
    this.scene = level1(); // create first level and set it
  }

  // Controller
  onkeydown(event) { this.keys[event.keyCode] = true; }

  onkeyup(event) { this.keys[event.keyCode] = false; }

  loop() {
    // Just move all the objects
    for (var i in this.scene) { this.scene[i].move(this); }

    // Render the scene
    for (i in this.scene) { this.scene[i].draw(this); }

    // Loop animation
    requestAnimationFrame( this.loop.bind(this) );
  }
}
```

Main.js

- inicializácia aplikácie
- zaregistrovanie vstupov

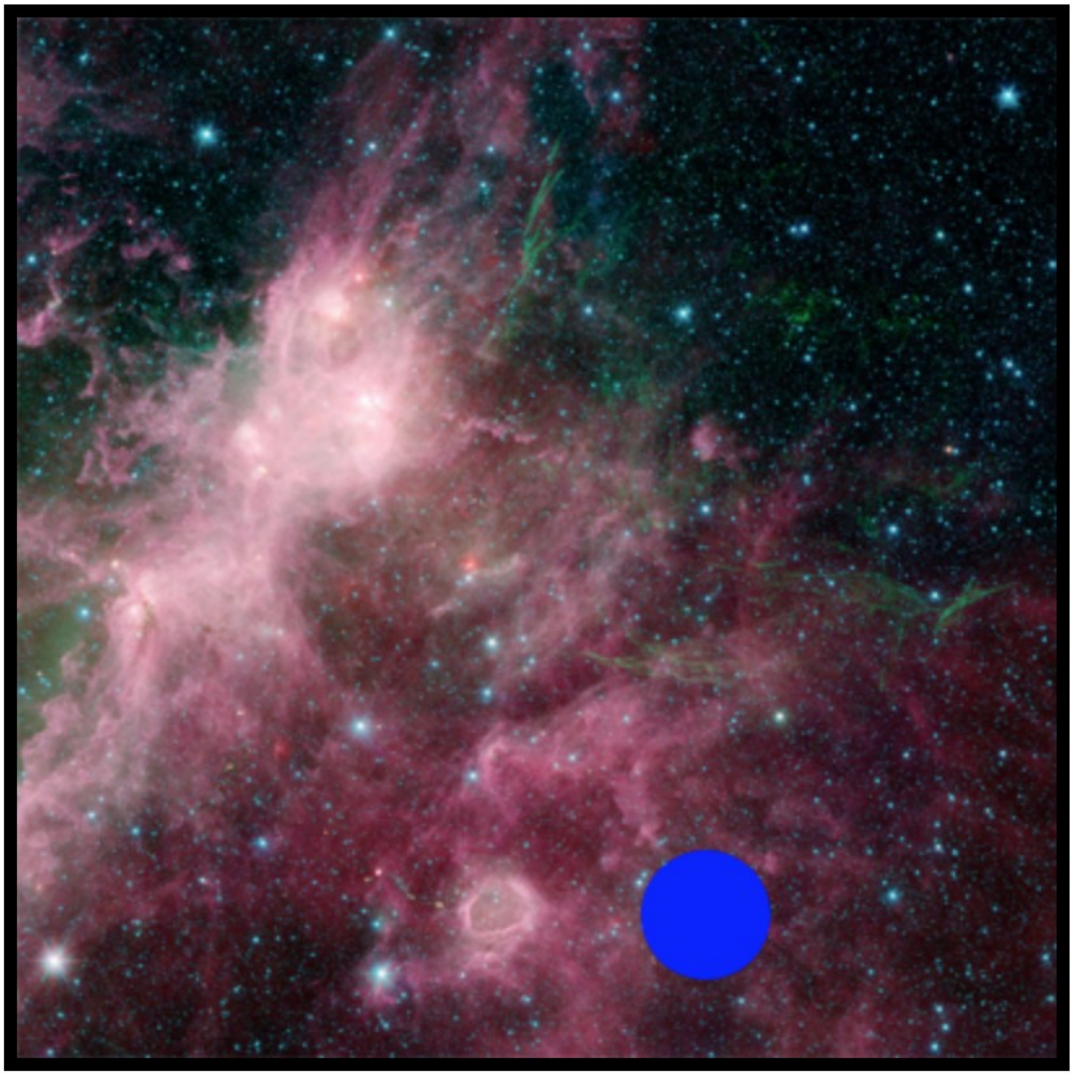
```
var game;

// Just start up our game
window.onload = function () {
  game = new Game("canvas");
  game.loop();
};

window.onkeydown = function (event) {
  game.onkeydown(event);
};

window.onkeyup = function (event) {
  game.onkeyup(event);
};
```

Kolízie objektov

- Vid. zdrojový kóv v: *1_collisions*

Dynamická scéna

- Cieľ:
  - Pridať do scény hráča, pozadie
  - Hráč strieľa animované projektily, ktoré istý čas existujú



HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>Projectiles</title>
  <meta charset="utf-8" />
  <link href="css/game.css" rel="stylesheet" />
  <script src="js/node.js"></script>
  <script src="js/gameobject.js"></script>
  <script src="js/background.js"></script>
  <script src="js/projectile.js"></script>
  <script src="js/player.js"></script>
  <script src="js/game.js"></script>
</head>
<body>
  <p>
    <canvas id="canvas" height="400" width="400">No Canvas :(</canvas>
  </p>
  <img id="projectile" src="img/projectile.png" hidden="true"/>
  <img id="background" src="img/background.jpg" hidden="true"/>
</body>
```

Trieda *Node*

- Observer pattern
- zároveň *Subject* aj *Observer*

```
// Observer pattern implementation
class Node {
    // Create new node
    constructor() { this.nodes = [] }

    // Add a new node
    // @node - node to add to observer list
    add(node) {
        if(!node) return false
        return this.nodes.push(node)
    }

    // Remove a node
    // @node - node to remove from observer list
    remove(node) {
        var index = this.nodes.indexOf(node)
        if (index >= 0) this.nodes.splice(index, 1)
    }
```

Trieda *Node*

- *notify()* - zaslanie správy (volaním metódy) objektu, umožňuje volať:
  - ľubovoľnú metódu s ľub. počtom argumentov

```
    // Notify observers
    // @event - name of the function to call on the observers
    // @... - additional arguments to pass to the event call
    notify( /*event , args ... */ ) {
        var event = arguments[0]
        var args = Array.prototype.slice.call(arguments, 1)

        // Call all observers that can receive the call
        for (var index in this.nodes) {
            var node = this.nodes[index]
            if (node && typeof(node[event]) == "function")
                node[event].apply(node, args)
        }
    }
}
```

Základný *GameObject*

```
class GameObject extends Node {
  constructor(game, x, y, width, height) {
    super()
    this.game = game // each gameobject can access the game object
    this.x = x
    this.y = y
    this.width = width
    this.height = height
  }

  move(dt) {
    // Trigger onmove event
    this.onmove(dt)
    // Notify all children
    this.notify("move", dt)
  }

  draw(ctx) {
    // Trigger ondraw event
    this.ondraw(ctx)
    // Notify all children
    this.notify("draw", ctx)
  }
  // Default event handlers
  onmove(dt) {}
  ondraw(ctx) {}
}
```

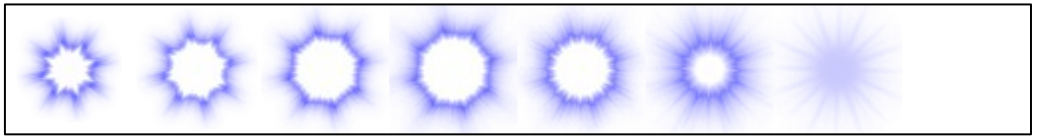
Background.js

- iba vykreslenie pozadia

```
class Background extends GameObject {
  constructor(game) {
    super(game, 0, 0, game.canvas.width, game.canvas.height)
    this.img = document.getElementById("background")
  }

  ondraw(ctx) {
    ctx.drawImage(this.img, this.x, this.y, this.width, this.height)
  }
}
```

Projectile.js

- vykreslenie je riadené časom
  - výber pod-obrázka
  - rotácia



```
class Projectile extends GameObject {
  constructor(game, x, y) {
    super(game, x, y, 60, 60)
    this.dx = 0
    this.dy = -25
    this.maxLife = 8
    this.life = 0
    this.img = document.getElementById("projectile")
  }

  ondraw(ctx) {
    var frame = Math.round((this.life/this.maxLife)*8)
    ctx.save()
    ctx.translate(this.x, this.y)
    ctx.rotate(this.life)
    // use frame to select part of an image
    ctx.drawImage(this.img, 64*frame, 0, 64, 64, -20, -30, this.width, this.height)
    ctx.restore()
  }
```

Projectile.js

- *onmove()* rieši
  - pohyb v priestore
  - aktualizáciu uplynutého času života prijektilu a jeho zánik

```
  onmove(dt) {
    this.x += this.dx * dt
    this.y += this.dy * dt

    // Collision logic here
    // TODO:

    // increase life
    this.life += dt

    // Die when reaching maxLife
    if (this.life > this.maxLife) this.game.remove(this)
  }
}
```

Player.js

- *onmove()* - pohyb podľa klávesov, streľba

```
class Player extends GameObject {
  constructor(game, x, y) {
    super(game, x, y, 50, 50)

    this.rof = 90 // rate of fire
    this.fireTimer = false
  }

  onmove(dt) {
    // Move object
    var keys = this.game.keys
    if ( keys[37] ) this.x-=5;
    if ( keys[39] ) this.x+=5;
    if ( keys[38] ) this.y-=5;
    if ( keys[40] ) this.y+=5;
    if ( keys[32] ) {  // space key
      this.startFiring()  // if pressed start firing
    } else {
      this.stopFiring()
    }
  }

  ondraw(ctx) { ... }
// continues on next slide
```

Player.js

- *start/stopFiring()* - nastavenie časovača, ktorý bude volať *fire()*
- *fire()* - vytvorí nový projektil a vloží ho scény

```
  startFiring() {
    // if timer is set, then skip
    if (this.fireTimer) return

    var player = this
    this.fireTimer = setInterval( function() { player.fire() }, this.rof )
  }

  stopFiring() {
    // if timer is no set, then skip
    if (!this.fireTimer) return

    clearInterval(this.fireTimer)
    this.fireTimer = false
  }

  fire() {
    var projectile = new Projectile(this.game, this.x+this.width/2, this.y)
    // add the new projectile to game object
    this.game.add(projectile)
  }
} // class end
```

Game reprezentuje celú aplikáciu

```
class Game extends GameObject {
     constructor(canvas) {
          // Init game object
          super(undefined, 0, 0, canvas.width, canvas.height)

          this.canvas = canvas
          this.ctx = canvas.getContext("2d")

          // Model
          this.keys = []
          this.time = Date.now()

          // Event handlers
          var game = this
          window.addEventListener("keydown", function(event) {
               game.keys[event.keyCode] = true
          })

          window.addEventListener("keyup", function(event) {
               game.keys[event.keyCode] = false
          })

          // Loop callback
          this.loop = function() { game.onloop() }

          // Load level1
          this.level1()
     }
```

Game.js

- *onloop()* - implementuje hlavný *event loop*
- *level1()* - vytorí nový level (vymaže koreň stromu v *nodes* a pridá pozadie a hráča ako 2 potomkov do stromu)

```
    onloop() {
        // Get time delta
        var now = Date.now()
        var dt = (now - this.time) / 100
        this.time = now

        // Animate
        this.move(dt)

        // Draw
        this.draw(this.ctx)

        // Loop animation
        requestAnimationFrame(this.loop)
    }

    // create a level
    level1() {
        this.nodes = []
        this.add(new Background(this))
        this.add(new Player(this, 0, 120))
    }
} // class end
```

main.js

- inicializácia aplikácie

```
// Init
window.onload = function() {
    var game = new Game(document.getElementById("canvas"))
    game.loop()
}
```

Dynamická scéna

- Vid. zdrojový kóv v: *2_projectiles*

Porovnanie

1_collisions

- organizácia všetkých objektov v jednom poli
  - jednoduchšie
- šírenie správ
  - iba volania *move(), draw()*

2_projectiles

- organizácia objektov v stromovej štruktúre
  - zložitejšie
  - flexibilnejšie
- šírenie správ riešené dvojicou
  - *move() - onmove()*
  - ľahko rozšíritelné o ďalšie správy

Zvuk

```
function Sound(file) {
  this.sound = document.createElement("audio");
  this.sound.src = file;
  this.sound.setAttribute("preload", "auto");
  this.sound.setAttribute("controls", "none");
  this.sound.style.display = "none";

  document.body.appendChild(this.sound);

  this.play = function(){
    this.sound.play();
  }
  this.stop = function(){
    this.sound.pause();
  }
}
```

```
var mySound;
mySound = new Sound("some_sound.mp3");
mySound.play();
```

Nabudúce

- Sieťové aplikácie
  - Základný prehľad a mechanizmy

Ďakujem za pozornosť