

Základy tvorby interaktívnych aplikácií

- Objektovo orientované programovanie v jazyku JavaScript
- Ing. Jaroslav Erdelyi
- LS 2021-2022

Obsah

- Čo je to objekt
- Základy tvorby objektov
- Pokročilejšie techniky v OOP
- HTML Canvas

Objekt

- entita, ktorá má **vlastnosti** (ktoré definujú jej stav) a **správanie** (meni jej stav, alebo stav iných objektov)
- Vlastnosti nazývame **atribúty** objektu
- Správanie
 - je definované cez **metódy** objektu, ktoré sú implementované ako funkcie vložené do objektu
- *Objektovo orientovaný návrh* - opísanie a dekompozícia problému pomocou spolupracujúcich objektov
- *Objektovo orientované programovanie* - implementácia pomocou objektov v konkr. OO program. jazyku

Objekt v JavaScript jazyku

- Objekt definujeme:
 - `Const nazovObjektu = {};`
- Vlastnosti (tzv. atribúty) objektu zapisujeme v tvare
 - *member: value*

```
const objectName = {  
  member1Name: member1Value,  
  member2Name: member2Value,  
  member3Name: member3Value,  
  ...,  
  memberXName: memberXValue,  
};
```

Objekt

```
const person = {  
  name:  
  age:  
  gender:  
  interests:  
};
```

```
const person = {  
  name: ['Bob', 'Smith'],  
  age: 32,  
  gender: 'male',  
  interests: ['music', 'skiing'],  
};
```

Objekt a správanie

```
const person = {  
  name: ['Bob', 'Smith'],  
  age: 32,  
  gender: 'male',  
  interests: ['music', 'skiing'],  
  bio: function() {  
  },  
  greeting: function() {  
  }  
};
```

Objekt a správanie

```
const person = {  
  name: ['Bob', 'Smith'],  
  age: 32,  
  gender: 'male',  
  interests: ['music', 'skiing'],  
  bio: function() {  
    alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age +  
      ' years old. He likes ' + this.interests[0] + ' and ' +  
      this.interests[1] + '.');  
  },  
  greeting: function() {  
    alert('Hi! I\'m ' + this.name[0] + '.');  
  }  
};
```

Vnorený objekt

```
const person = {  
  name : {  
    first: 'Bob',  
    last: 'Smith'  
  },  
  age: 32,  
  gender: 'male',  
  interests: ['music', 'skiing'],  
}
```


Prístup k vlastnostiam a správaniu

```
// pristup k atributom:  
person.age  
person.name.first  
// alternativna notacia pre pristup k atributom  
person['age']  
person['name']['first']
```

```
// zapis do atributov  
person.age = 45;  
person['name']['last'] = 'Cratchit';  
person['eyes'] = 'hazel';  
person.bye = function() { alert("Bye everybody!"); }
```

Viac objektov

```
const person1 = {  
  name: 'Chris',  
  greeting: function() {  
    alert('Hi! I\'m ' + this.name + '.');  
  }  
}
```

```
const person2 = {  
  name: 'Deepti',  
  greeting: function() {  
    alert('Hi! I\'m ' + this.name + '.');  
  }  
}
```

Viac objektov

```
function createNewPerson(name) {  
  const obj = {};  
  obj.name = name;  
  obj.greeting = function() {  
    alert('Hi! I\'m ' + obj.name + '.');  
  };  
  return obj;  
}
```

```
const John = createNewPerson('John');  
John.name;  
John.greeting();
```

Vytvorenie objektu pomocou funkcie

```
function Person(name) {  
  this.name = name;  
  this.greeting = function() {  
    alert('Hi! I\'m ' + this.name + '.');  
  };  
}
```

```
let person1 = new Person('Bob');  
let person2 = new Person('Sarah');
```

```
function Person(first, last, age, gender, interests) {
  this.name = {
    first : first,
    last : last
  };
  this.age = age;
  this.gender = gender;
  this.interests = interests;
  this.bio = function() {
    alert(this.name.first + ' ' + this.name.last + ' is ' + this.age +
      ' years old. He likes ' + this.interests[0] + ' and ' + this.interests[1]
+ '.');
  };
  this.greeting = function() {
    alert('Hi! I\'m ' + this.name.first + '.');
  };
}
```

```
let person1 = new Person('Bob', 'Smith', 32, 'male', ['music', 'skiing']);
```

Klíčové slovo *this*

- *This* ukazuje na aktuální objekt
- V `person1` ukazuje na `person1` a v objektu `person2` ukazuje na `person2`

```
const person1 = {
  name: 'Chris',
  greeting: function() {
    alert('Hi! I\'m ' + this.name +
    '.');
  }
}

const person2 = {
  name: 'Deepti',
  greeting: function() {
    alert('Hi! I\'m ' + this.name +
    '.');
  }
}
```

Triedy, inštancie objekty

Class-based OOP

- napr. C++, Java, ...
- Trieda je šablóna pre objekt
- Objekt je inštanciou triedy, po kompilácii a v run-time sú nemenné

```
class MyClass {          // The class
public:                  // Access specifier
    // Attribute (int variable)
    int myNum;
    // Attribute (string variable)
    string myString;
};

// Create an object of MyClass
MyClass myObj;
// Access attributes and set values
myObj.myNum = 15;
myObj.myString = "Some text";
```

Prototyp-based OOP

- napr. JavaScript, ...
- objekt *Object* je prototyp
- Objekt je inštanciou prototypu, v run-time možno meniť

```
let person1 = new Object(
{
    name: 'Chris',
    age: 38,
    greeting: function() {
        alert('Hi! I\'m ' + this.name + '.');
    }
});
```

Triedy v JavaScript jazyku

```
class Person {  
  constructor(first, last, age, gender, interests) {  
    this.name = {  
      first,  
      last  
    };  
    this.age = age;  
    this.gender = gender;  
    this.interests = interests;  
  }  
  
  greeting() {  
    console.log(`Hi! I'm ${this.name.first}`);  
  };  
  
  farewell() {  
    console.log(`${this.name.first} has left the building. Bye for now!`);  
  };  
}
```

nastavenie atribútov
pomocou
konštruktora

Vytvorenie inštancií

```
let han = new Person('Han', 'Solo', 25, 'male', ['Smuggling']);  
han.greeting();  
// Hi! I'm Han  
  
let leia = new Person('Leia', 'Organa', 19, 'female',  
['Government']);  
leia.farewell();  
// Leia has left the building. Bye for now
```

```
let person1 = new Person('Bob');  
let person2 = new Person('Sarah');
```

Pokročilé OOP prístupy

- Zapuzdrenie
- Dedenie
- Polymorfizmus
- Asociácia, Agregácia, Kompozícia objektov

Zapuzdrenie (angl. Encapsulation)

```
class Person {  
    constructor(first, last, age, gender, interests) {  
        this.name = {  
            first : first,  
            last : last  
        };  
        this._age = age;  
        this.gender = gender;  
        this.interests = interests;  
    }  
  
    get age() {  
        return this._age + 5;  
    }  
  
    set age(newAge) {  
        this._age = newAge;  
    }  
}
```

Dedenie (ang. Inheritance)

- Objekt získava dedením atribúty a metódy od iného objektu (platí rovnako aj pre triedy)
- **Dedenie slúži na konkretizáciu, špecializáciu objektov:** z abstraktného objektu, ktorý ma nejaké vlastnosti, pridávaním ďalších vlastností dedený objekt konkretizujeme, napr.:
 - trieda *Person* je šablóna pre ľubovoľnú osobu (obsahuje atribúty, t.j. vlastnosti, ktoré sú spoločne pre všetky osoby)
 - trieda *Teacher* je špecializáciou triedy *Person*, t.j. obsahuje tie iste vlastnosti ako *Person*, ALE navyše obsahuje aj vlastnosti špecifické pre učiteľa (napr. aký predmet vyučuje)

Dedenie (ang. Inheritance)

- vytvoríme triedu *Teacher*, ktorá dedí z triedy *Person* nasl. zápisom:
 - *Teacher* obsahuje všetky atribúty (first name, last name, age, gender, interests) a metódy (greeting, farewell) z triedy *Person*
 - pridáme ďalšie atribúty špecifické pre *Teacher*: *subject*, *grade*

```
class Teacher extends Person {  
    constructor(subject, grade) {  
        super(); // calling the parent constructor - 'this' is  
        initialized  
        this.subject = subject;  
        this.grade = grade;  
    }  
}
```

Dedenie (ang. Inheritance)

- musíme zavolať konštruktor pôvodnej triedy volaním metódy *super()*

```
class Teacher extends Person {
  constructor(first, last, age, gender, interests, subject, grade)
  {
    super(first, last, age, gender, interests);

    // subject and grade are specific to Teacher
    this.subject = subject;
    this.grade = grade;
  }
}

let Snow = new Teacher('John', 'Snow', 58, 'male', ['Potions'], 'Dark arts',
5);
Snow.greeting(); // Hi! I'm John.
Snow.farewell(); // John has left the building. Bye for now.
Snow.age // 58
Snow.subject; // Dark arts
```

Polymorfizmus

- zdedene triedy obsahujú rovnako pomenovane metódy ako v povodnej triede, ale metody v zdedenych triedach maju odlišne telo (prípadne aj iné parametre)
- príklad:
 - trieda *Shape* reprezentuje genericky geometricky objekt
 - trieda *Shape* ma metódu *area()* na výpočet obsahu plochy geometrického objektu
 - triedy *Circle*, *Triangle*, *Rectangle* dedia z triedy *Shape*, ale majú vlastne implementácie metódy *area()* na výpočet obsahu plochy
- **Prínos:** môžeme nad rôznymi objektmi volať rovnako nazvanú metódu, pričom každý objekt ma vlastnú implementáciu

Polymorfizmus

```
class Shape {  
    area() {  
        return 0;  
    }  
}
```

```
class Circle extends Shape {  
    constructor(r) {  
        super();  
        this.radius = r;  
    }  
  
    area() {  
        return Math.PI * this.radius ** 2;  
    }  
}
```


Polymorfizmus

```
class Rectangle extends Shape {  
    constructor(w, h) {  
        super();  
        this.width = w;  
        this.height = h;  
    }  
    area() {  
        return this.width * this.height;  
    }  
}
```

```
class Triangle extends Shape {  
    constructor(b, h) {  
        super();  
        this.base = b;  
        this.height = h;  
    }  
    area() {  
        return this.base * this.height / 2;  
    }  
}
```

Asociácia, Agregácia, Kompozícia objektov

- **Agregácia** - vzťah medzi 2 objektami, pričom *potomok môže* existovať bez *rodiča*

```
function Book(title, author) {  
  this.title = title;  
  this.author = author;  
}  
const book1 = new Book ('Hippie', 'Paulo Coelho');  
const book2 = new Book ('The Alchemist', 'Paulo Coelho');  
let publication = {  
  "name": "new publication Inc",  
  "books": []  
}  
publication.books.push(book1);  
publication.books.push(book2);
```

- **Kompozícia** - vzťah medzi 2 objektami, pričom *potomok nemôže* existovať bez *rodiča*

```
let Book = {  
  "title": "The Alchemist",  
  "author": "Paulo Coelho",  
  "publication": {  
    "name": "new publication Inc",  
    "address": "chennai"  
  }  
}
```

- **Asociácia** - vyjadruje, že medzi 2 objektami **je vzťah**, napr. objekt1 pristupuje k atribútu objektu2, alebo vola metodu objektu2

Návrh interaktívnej hry

Tvorba návrhu

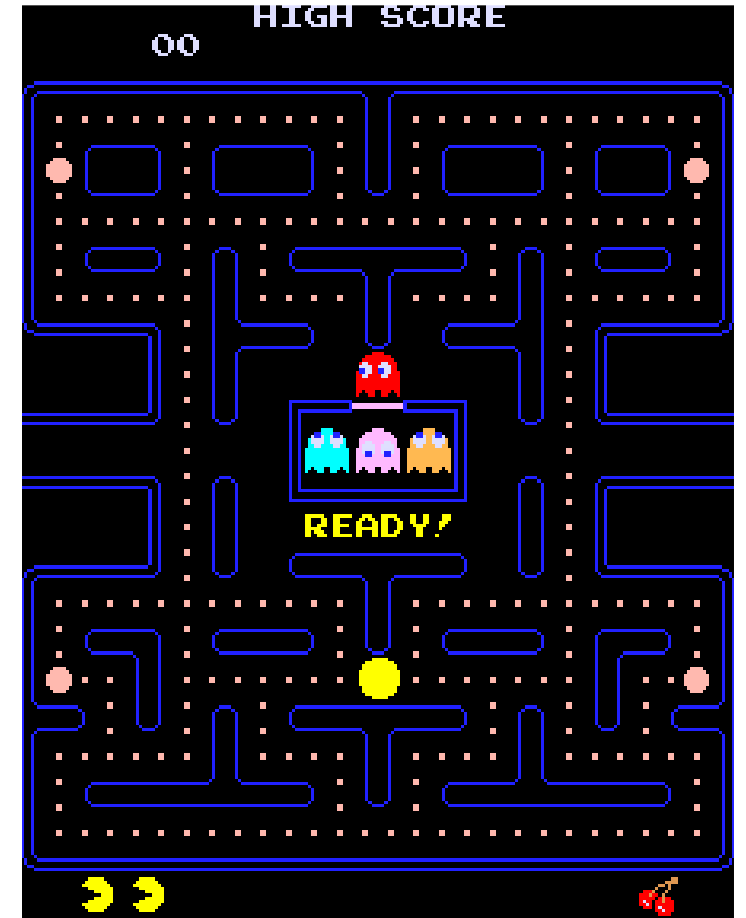
- Definuj GUI rozhranie
 - analýza predošlých podobných rozhraní
- Identifikuj **objekty**
- Definuj **udalosti** (ako reakcie na **akcie** používateľa), ktoré menia **stav** riešenia problematiky
- Možné situácie over na prototype resp. modeli

Objekty a akcie

- Ako identifikovať objekty a akcie?
 - Základ *objektovo-orientovanej dekompozície problému a objektovo-orientovaného návrhu a implementácie*
- **Objekt**
 - Entita, ktorá má **vlastnosti** (ktoré definujú jej stav) a **správanie** (ktoré mení jej stav, alebo stav iných objektov)
- **Akcie**
 - Zvyčajne príkaz na vykonanie správania objektu (môže iniciovať používateľ, alebo aj sám systém)

Objekty a akcie

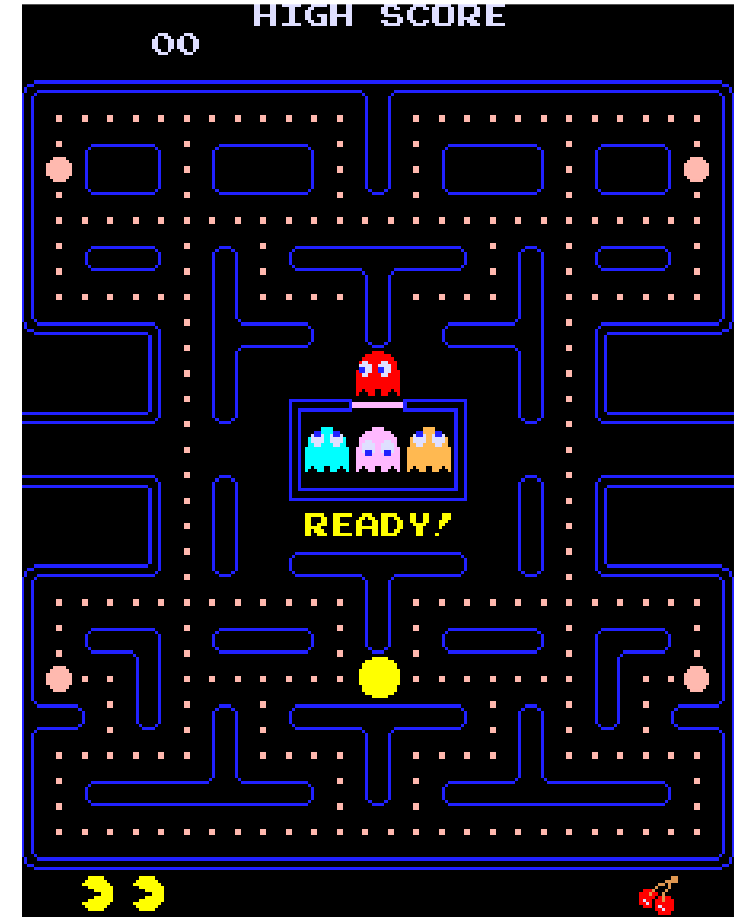
- Objekty v Pac Manovi



Objekty a akcie

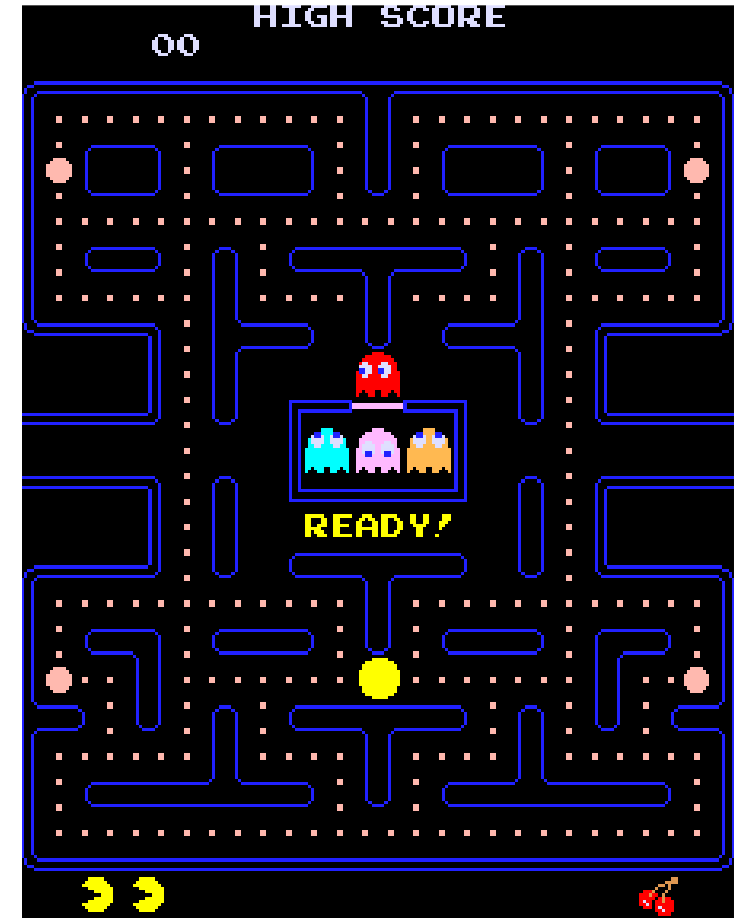
- Objekty v Pac Manovi

- Hráč
- Duch
- Potrava
- PowerUp
- Stena
- Level (mapa)
- Skóre, text
- Počet životov



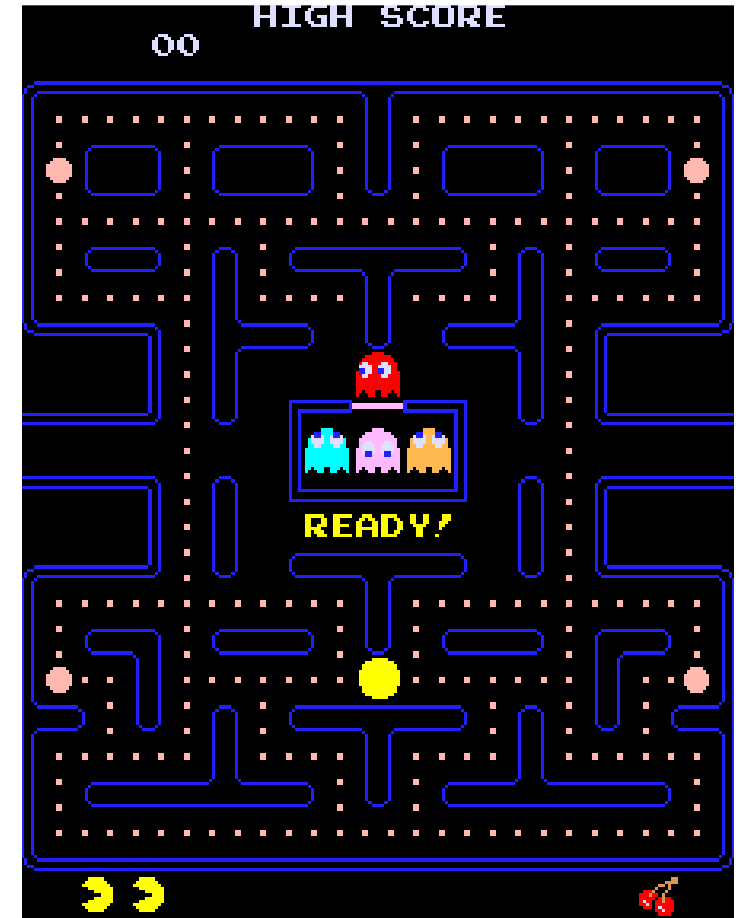
Objekty a akcie

- Akcie v Pac Manovi



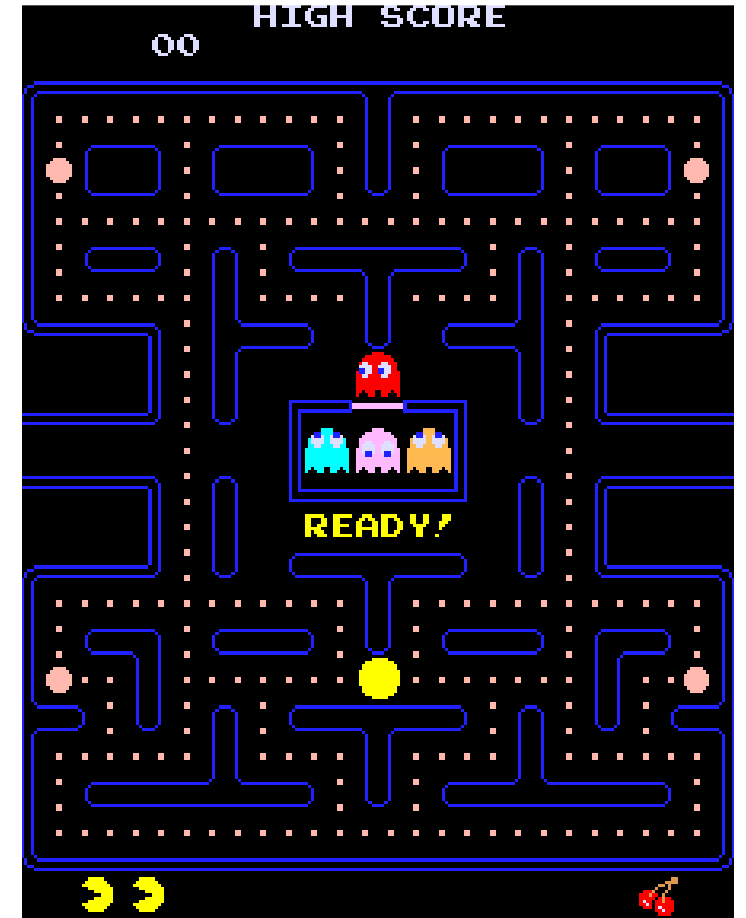
Objekty a akcie

- Akcie v Pac Manovi
 - Z pohľadu hráča je to ovládanie hry



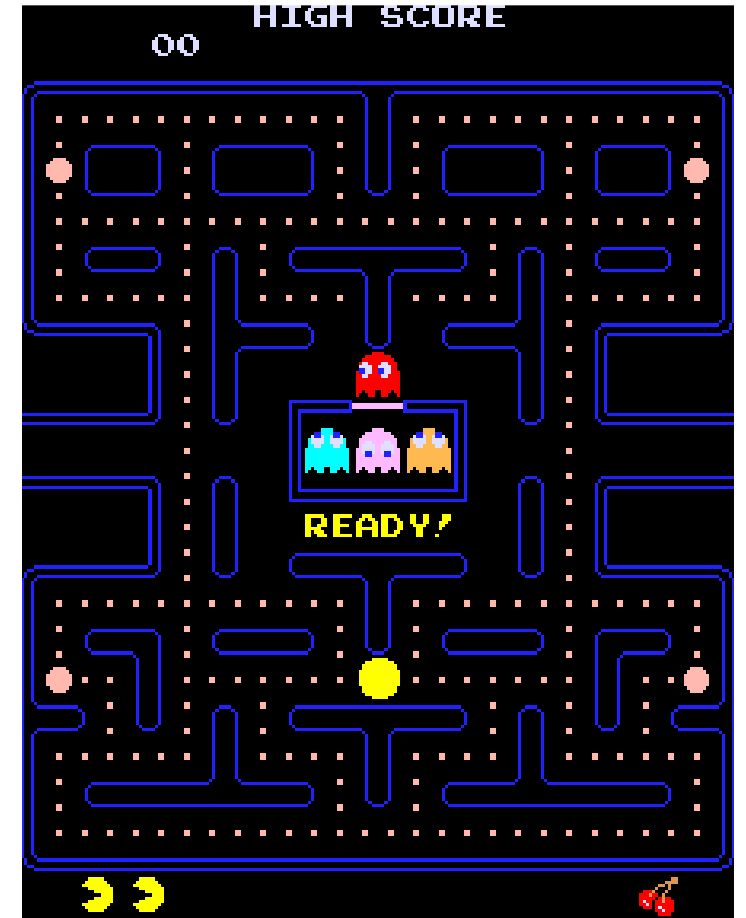
Objekty a akcie

- Udalosti a stavy v Pac Manovi



Objekty a akcie

- Udalosti a stavy v Pac Manovi
 - Zmena pozície – hráč aj duch
 - Zmena textu
 - Zmena skóre a počtu životov
 - Zobrazenie potravy



Objekty a akcie

- Objekty v hře Super Mario



Objekty a akcie

- Objekty v hre Super Mario
 - Hráč
 - Nepriatelka
 - Kocky
 - Pozadie
 - Level
 - Minca
 - Rúra
 - Texty



Objekty a akcie

- Akcie v hře Super Mario



Objekty a akcie

- Akcie v hre Super Mario
 - Z pohľadu hráča je
 - to ovládanie hry



Objekty a akcie

- Udalosti a stavy v hre Super Mario



Objekty a akcie

- Udalosti a stavy v hre Super Mario
 - Zmena pozície
 - Zmena textu
 - Zmena skóre
 - Zmena životov



Celkový pohľad

- **Stav** aplikácie (Model) je reprezentovaný **stavom objektov**
 - **zmena stavu** objektov sa dosiahne **volaním** (t.j. vykonaním) metód objektov
- **Udalosti** v aplikácii
 - sú inicializovane **akciou** používateľa (napr. reakcia na vstup)
 - sú inicializovane samotnou aplikáciu (napr. autonómny pohyb nepriateľa)
 - menia **stav** aplikácie volaním metód objektov

Počítačová grafika a HTML Canvas

2D Grafika

- Rastrová a vektorová reprezentácia
- Založene na vykresľovaní
 - základných geom. útvarov
 - rastrových obrázkov
- farebný model RGBA
 - kompozícia na základe priesvitnosti

3D Grafika

- Založená na vykresľovaní povrchovej reprezentácie (najčastejšie polygóny a trojuholníky)
- Objekty sú osvetlene a otextúrovane
- Oveľa výkonovo náročnejšia
- Viac na predmete *Princípy počítačovej grafiky a spracovania obrazu*
- V HTML5 možno použiť *Canvas* a *WebGL*
 - knižnica *three.js*

HTML Canvas

- Štandardný HTML5 element, ktorý reprezentuje voľne manipulovateľný obraz (2D frame buffer)
- Poskytuje kontext, cez ktorý možno kresliť do plochy priamo z JavaScriptu
- Objekt *Context* poskytuje základne funkcie pre kreslenie
- **Nie je nutné manipulovať DOM**
- Rýchlejšie a vhodnejšie pre tvorbu hier

HTML Canvas

- Poskytuje:
 - kreslenie textu
 - kreslenie geometrických telies
 - kreslenie obrázkov
- Umožňuje:
 - animácie
 - interakciu

Canvas element

- Vytvorenie *Canvas* elementu

```
<canvas id="canvas" height="100" width="100">  
Sorry no Canvas :(  
</canvas>
```

- Práca s *Canvas* elementom

```
var canvas = document.getElementById("canvas")  
var context = canvas.getContext("2d")
```

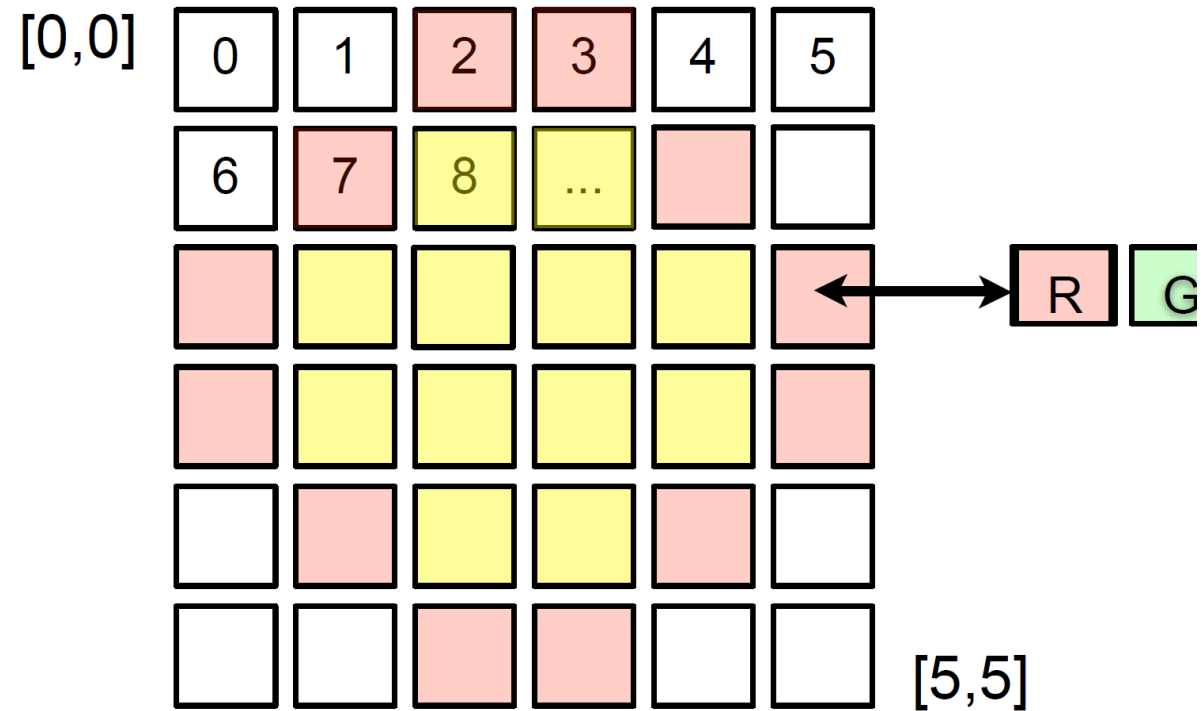

Vytvorenie kontextu

- Na prácu so samotným *Canvas* potrebuje získať jeho *Context* objekt
- *Context* objekt poskytuje
 - atribúty - nastavenie farby, typ čiary, výplne, ...
 - metódy - vykreslenie čiary, kruhu, krivky, obrázka,...
- Obráz sa nemení pokiaľ ho neupravíme

```
var canvas = document.getElementById("canvas")  
var context = canvas.getContext("2d")
```

Canvas, framebuffer, pixely

- Canvas poskytuje framebuffer - 2D pole pixelov
 - ľavý horný roh [0,0], pravý dolný roh [height, width]
- Každý pixel možno priamo meniť
 - súradnica [x,y], alebo index v rozsahu 0 až *img.data.length*



Prístup k pixelom

- Priamy prístup

```
// nacitanie obrazu 100x100 pixelov zo suradnice [0,0]
var img = canvas.getImageData(0,0,100,100)
img.data[0] = 255; // R
img.data[1] = 0; // G
img.data[2] = 0; // B
img.data[3] = 0; // A
canvas.putImageData(img, 0, 0) // zapisanie obrazu 100x100
                                // pixelov na
                                // suradnicu [0,0]
```

Kreslenie textu

```
var x = canvas.width / 2;  
var y = canvas.height / 2;  
context.font = '30pt Calibri';  
context.textAlign = 'center';  
context.fillStyle = 'blue';  
context.fillText('Hello World!', x, y);
```

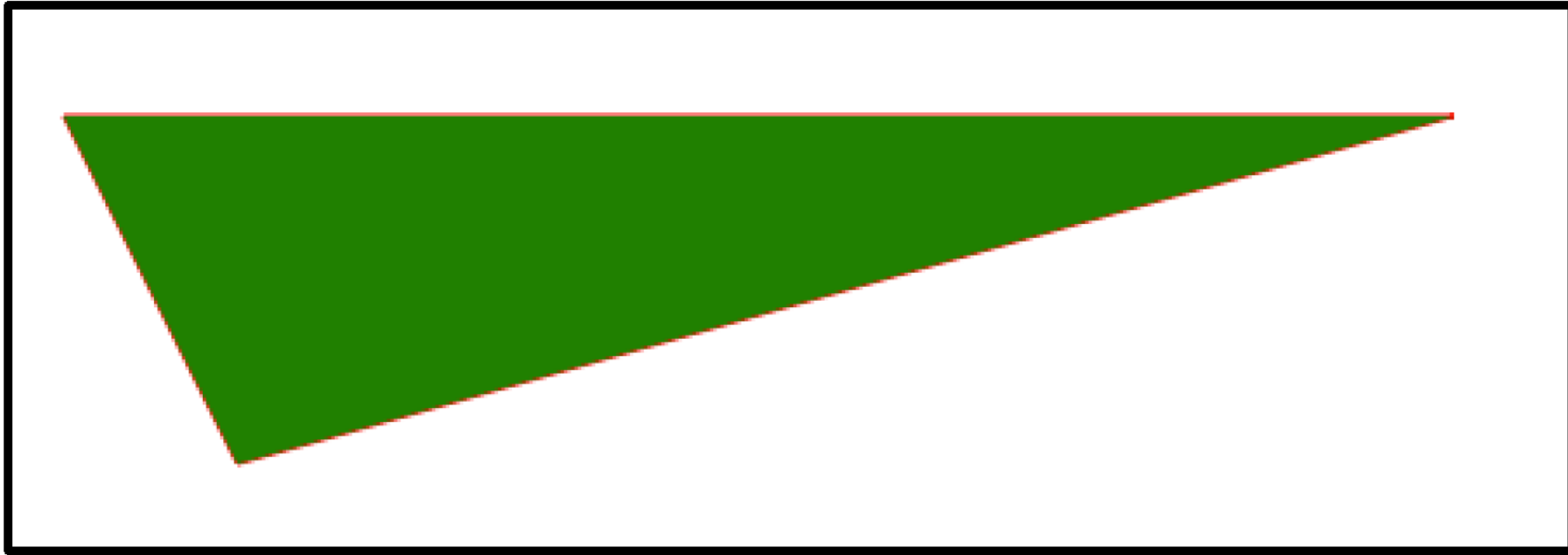


Hello World!

Kreslenie čiar a tvarov

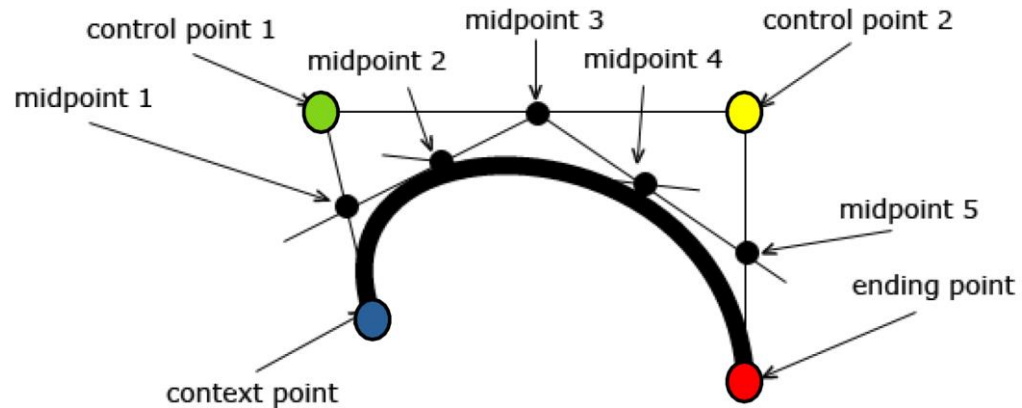
```
context.beginPath();           // začni kresliť tvar
context.strokeStyle = "red";   // nastav farbu čiary
context.fillStyle = "green";   // nastav farbu vyplne
context.moveTo(100, 150);      // presun sa na pozíciu
context.lineTo(450, 50);       // nakresli čiaru na pozíciu
context.lineTo(50, 50);
context.lineTo(100, 150);
context.closePath();           // spoj posledný bod s prvým
context.stroke();               // vykresli
```

Kreslenie čiar a tvarov



Kreslenie kriviek

- Napr. vykreslenie Bezierovej krivky



```
context.beginPath();  
context.moveTo(188, 130);  
context.bezierCurveTo(140, 10, 388, 10, 388, 170);  
context.lineWidth = 10;  
context.strokeStyle = "black";  
context.stroke();
```

Obrázky

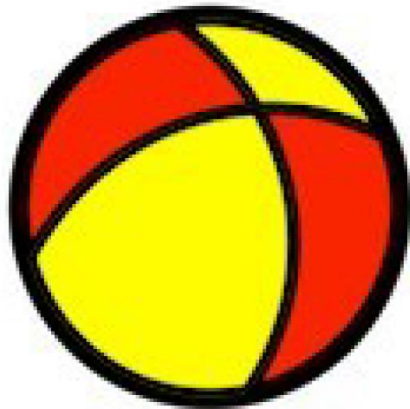
- Obrázky je možné kresliť pomocou objektu *Image*
- Je možné použiť `` element z DOM

```
var imageObj = new Image();
imageObj.src = "ball.png"

// Obrázok sa vykreslí až po nabití
imageObj.onload = function () {
    context.drawImage(imageObj, x, y, width, height);
}
```


Obrázky

- Obrázky je možné kresliť pomocou objektu *Image*
- Je možné použiť `` element z DOM



Manažment stavov

- *Context* dokáže uchovať a obnoviť svoj stav
 - užitočne pri kreslení viacerých nezávislých objektov
 - stav sa ukladá/vyberá zo zásobníka

```
// vykresli objekt1
context.save()                // uloz akt. stav
// vykonaj transformacie nad objekt1
context.drawImage(imageObj1,0,0)// vykresli transformovany obr.

context.restore()             // obnov stav

// vykresli objekt2
context.save()                // uloz akt. stav
// vykonaj transformacie nad objekt2
context.drawImage(imageObj2,0,0)// vykresli transformovany obr.

context.restore()             // obnov stav
```

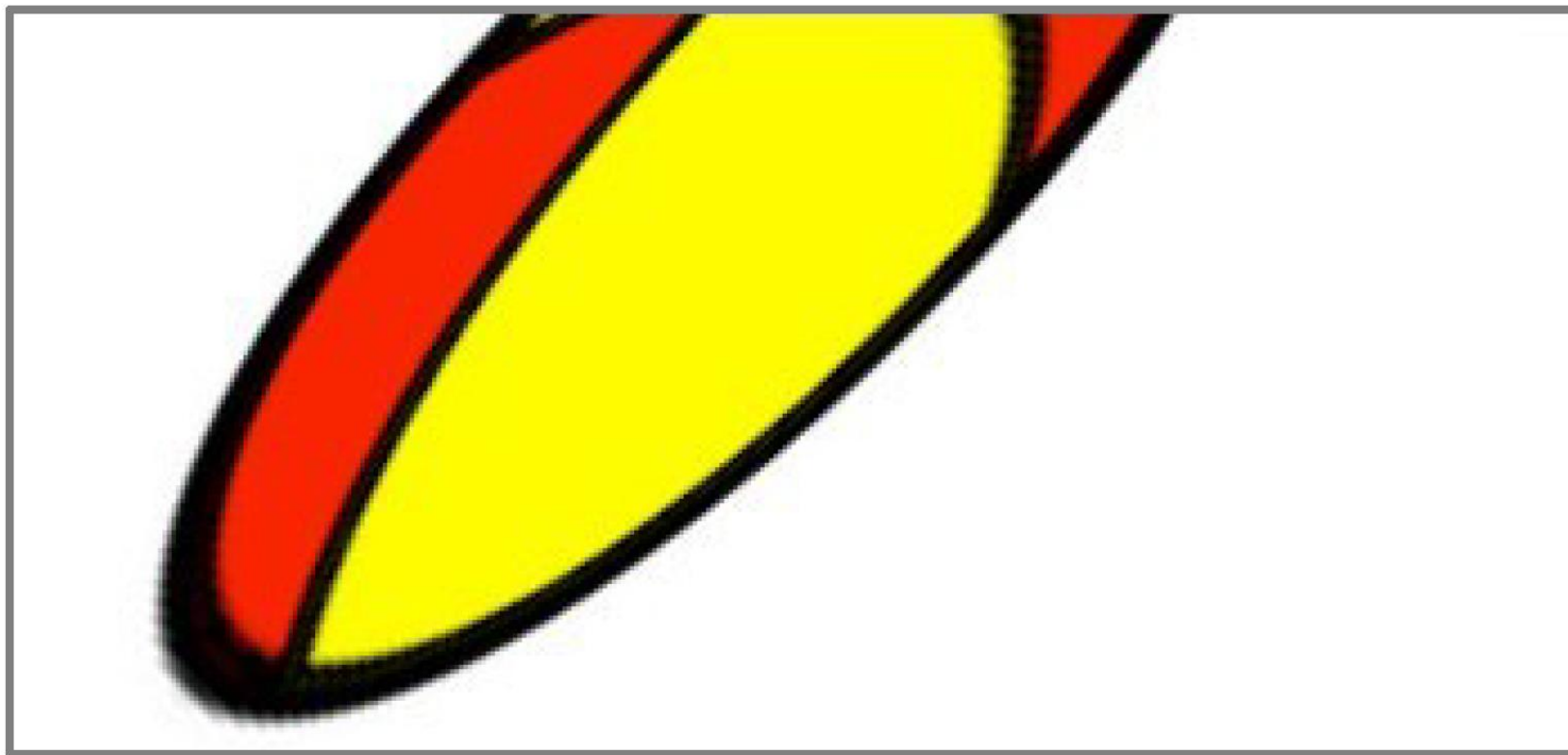
Transformácie

- Možná aplikácia základných geometrických transformácií

```
// kresli objekt
context.save()                // ulozenie akt. stavu
context.translate(40,120)     // posun
context.rotate(-45*Math.PI/180.0) // rotacia
context.scale(3.0,1.0)        // skalovanie
context.drawImage(imageObj,0,0) // vykreslenie
                                // transformovaneho obr.
context.restore()             // obnovenie stavu
```

Transformácie

- Možná aplikácia základných geometrických transformácií



Vytvorenie celého obrazu

Stree fighter



Earthworm Jim



Základný princíp

- Každý obraz vytvárame celý z prázdneho obrazu
- Každý obraz kreslime od pozadia k poprediu ako kompozíciu viacerých častí
- Ak vykresľujeme aspoň 24 obrázkov za sekundu tak vytvárame ilúziu plynulej animácie
- Animáciu vytvárame zmenou parametrov **Modelu** a jeho následným vykreslením cez **View**

Kompozícia obrazu



Zdroje

- <http://www.html5canvastutorials.com/>
- https://www.w3schools.com/graphics/canvas_reference.asp
 - podrobný opis všetkých atribútov a metód

Zhrnutie

- Objektovo-orientovaná paradigma
 - Čo sú triedy, objekty, inštancie
 - Čo sú atribúty a metódy objektu
 - Kompozícia objektov, dedenie
- Ako pristupovať k objektovo-orientovanému návrhu
 - Ako rozpoznať objekty, ich atribúty a metódy
- HTML Canvas
 - Prístup k framebuffer-u a modifikácia pixel-ov
 - Kreslenie základných geom. Telies
 - Zobrazenie obrázka
 - Vykresľujeme scénu od pozadia k poprediu