Ondřej Žára

# JavaScript

## Programátorské techniky a webové technologie

- Poutavý výklad na příkladu hry
- Moderní postupy vývoje funkční aplikace
- Od základů po pokročilé techniky

computer press

Ondřej Žára

# JavaScript
## Programátorské techniky a webové technologie

Computer Press
Brno
2015

# JavaScript
## Programátorské techniky a webové technologie

**Ondřej Žára**

**Obálka:** Martin Sodomka
**Odpovědný redaktor:** Martin Herodek
**Technický redaktor:** Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4573-9

Vydalo nakladatelství Computer Press v Brně roku 2015 ve společnosti Albatros Media a. s.
se sídlem Na Pankráci 30, Praha 4. Číslo publikace 19 278.


1. vydání

ALBATROS MEDIA a.s.

# Obsah

# Úvod

Jen těžko se lze pohybovat v oblasti tvorby webu a nepovšimnout si, jakému zájmu se JavaScript těší. Spolu s oblíbeností a rozšířeností logicky roste také poptávka po kvalifikovaných vývojářích. Najít zkušeného JavaScriptového programátora přitom vůbec není jednoduché; základy si na Internetu najde každý, ale skutečně hlubokého pochopení dojde jen málokdo. Je k tomu zapotřebí mnoho praxe, zvídavosti i obecného algoritmického rozhledu.

Na oblíbeném programátorském webu *stackoverflow.com* je položeno přes tři čtvrtě milionu otázek na téma JavaScript. To je dobrým dokladem zájmu i nedostatečného vzdělání mezi webovými vývojáři. Ambicí této knihy je poskytnout podklady pro systematické pochopení JavaScriptu a jeho praktického používání.

## Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

*Computer Press*
*Albatros Media a.s., pobočka Brno*
*IBC*
*Příkop 4*
*602 00 Brno*

nebo

*sefredaktor.pc@albatrosmedia.cz*

**Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.**

## Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/K2209* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy. Najdete je na serveru GitHub na adrese *https://github.com/ondras/javascript*.

# Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2209* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA 1

# Začínáme

**V této kapitole:**
- Motivace
- Java-co?
- Metodika

## Motivace

Předvídání budoucnosti je vždy ošemetná věc, u informačních technologií tomu není jinak. S ohledem na vývoj posledních zhruba deseti let se však zdá, že málokterá výpočetní oblast zažívá takový opravdový boom jako World Wide Web. Důvodů je mnoho: dostupný a mobilní Internet, miniaturizace chytrých zařízení, překotný rozvoj webových prohlížečů, snadno přístupné jazyky a rozhraní. Troufám si proto tvrdit, že sázka na webové technologie se rozhodně vyplatí – a to nejen těm, kdo potřebují nabízet své produkty elektronickou cestou, ale i vývojářům všeho druhu, technologickým fandům i zájemcům o hlubší setkání s výpočetní technikou.

Konkrétní nástroje, postupy a knihovny přicházejí a s různorodým odstupem zase mizí. Základní koncepty a jazyky WWW však zůstávají a postupným vylepšováním upevňují web na předním místě každodenních životních činností. Práce s HTML stránkami, jejich design a postupné oživování pomocí skriptů jsou tak více a více považovány za základní IT vzdělání; mnoho vývojářů se s prvními algoritmy a jazyky potká právě v rámci tvorby webu.

S rostoucí nabídkou je na místě dodávat i užitečné studijní podklady. Zde přichází ke slovu tato kniha: poskytuje krátký exkurz do světa pod pokličkou HTML stránek, ukazuje principy vývoje a seznamuje s typickými postupy. Na své by si tak měli přijít jak začátečníci, tak i zkušenější vývojáři z ostatních oborů, kteří by rádi obhlédli, jak že se to pracuje s tím HTML, CSS a JavaScriptem.

## Java-co?

Specifické typografické znaménko v nadpisu (tzv. *interrobang*, kombinace vykřičníku a otazníku) koresponduje s údivem, který tento jazyk – včetně svého jména – nezřídkakdy vyvolává ve vývojářích. Oprávněně očekávají klientskou verzi jazyka Java, jenže

ouha. Jak říká tradiční anglické úsloví, „Java is to JavaScript as ham is to hamster" (slovní hříčka přeložitelná možná jako „Java a JavaScript, to je jako párek a kašpárek"), tedy podobnost mezi těmito jazyky je jen v názvu.

Pozoruhodnou souhrou historických událostí se stalo, že jazyk JavaScript, navržený právě pro skriptování v prostředí webového prohlížeče, dostal do vínku nezvyklou kombinaci vlastností. Jeho syntaxe vychází z jazyka C, některá standardní rozhraní (vestavěné funkce, objekty, proměnné) se podobají Javě, objektový a funkcionální model je inspirován jazyky Scheme a Self. Je dynamicky typovaný (tj. o datových typech proměnných se rozhoduje až za běhu programu) a funkce jsou v něm k dispozici jako běžný datový typ. JavaScript staví na takzvané *prototypové dědičnosti* (budeme se jí věnovat v jedenácté kapitole), která bývá velmi obtížně uchopitelná pro programátory vyškolené v tradičním *třídním* objektovém modelu.

Během několika prvních let života dokázal JavaScript odsunout na vedlejší kolej svého jediného konkurenta, VBScript, navržený společností Microsoft jako alternativní jazyk pro skriptování webových stránek, a stal se tak de facto jediným skutečným programovacím jazykem použitelným v rámci HTML dokumentů. Implementace JavaScriptu je dnes nedílnou součástí všech webových prohlížečů a bez nadsázky lze říci, že se jedná o jednu z nejrozšířenějších technologií vůbec.

**Odbočka (historická)**

**Vývoj a verze JavaScriptu**

Archeologičtí nadšenci se mohou detailnější informace dozvědět mimo jiné na stránkách *https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript* a *http://en.wikipedia.org/wiki/JavaScript#History* – tato odbočka je zkrácenou kombinací textů z několika zdrojů, zejména však těch dvou odkazovaných.

První verze jazyka JavaScript byla navržena a naimplementována během pouhých deseti dní (!) Brendanem Eichem ve společnosti Netscape. Psal se rok 1995 (mimo jiné též rok vzniku jazyka PHP) a Eich dostal za úkol v této kruté časové lhůtě navrhnout *menšího a hloupějšího sourozence Javy* – jako konkurenční krok v tvrdém souboji s prohlížečem Internet Explorer. Tato verze jazyka se jmenovala Mocha a záměrně, aby nevypadala příliš jako Java, nabízela namísto tříd prototypy.

Ještě v roce 1995 došlo k přejmenování na LiveScript v rámci vydání v prohlížeči Netscape Navigator 2. Brzy poté přibyla do Netscape Navigatoru podpora pro spouštění appletů Java a zároveň s tím dostal klientský skriptovací jazyk své dnešní jméno.

V roce 1996 se objevuje první konkurenční implementace – JScript v Internet Exploreru 3. Zároveň dochází k formální standardizaci jazyka, tentokrát pod názvem ECMAScript (ECMA je název standardizační organizace *European Computer Manufacturers Association*, zodpovědné mimo jiné za specifikace formátů dat na CD či jazyka C#).

Poslední zajímavá zmínka ve dvacátém století pochází z konce roku 1999, kdy světlo světa spatřuje třetí verze specifikace jazyka, ECMA-262. Ta je na dlouhou dobu poslední, a JavaScript se tak v této podobě během následujících deseti let rozšiřuje do podvědomí webových vývojářů.

Pro svůj vlastní prohlížeč Chrome se společnost Google rozhodla vytvořit zbrusu novou a výkonnou implementaci JavaScriptu. V roce 2008 se objevuje Chrome 1.0 a implementace dostává název V8. Tím dochází k oživení myšlenky, že jazyk jako takový nemá důvod být používán jen v rámci klientského skriptování a lze jej využít pro programování serverového kódu. Tyto úvahy v roce 2009 ústí v založení neformální organizace *CommonJS*, která si za cíl stanovila prozkoumat a standardizovat chybějící serverová JavaScriptová rozhraní.

Ještě v roce 2009 se ale objevuje projekt Node.js: sada rozhraní a knihoven, obalující implementaci V8, a dovolující tak jednoduše spouštět JavaScriptový kód i mimo webový prohlížeč. Node.js si velmi rychle získal silnou uživatelskou základnu, zastínil CommonJS, a stal se tak de facto standardním řešením pro serverové vykonávání JavaScriptu.

Koncem roku 2009 konečně vychází nová, pátá verze specifikace jazyka ECMAScript (ES5). Obsahuje podporu pro tzv. *striktní režim*, funkce pro práci s datovým formátem JSON a další.

Dnes, v době psaní knihy (počátek roku 2015), je ECMAScript v pátém vydání nejrozšířenější verzí, a proto se v knize využívá právě tento standard. Organizace ECMA však završuje práci na šesté verzi, která přinese razantní novinky jak syntaktické, tak funkční. Tato šestá verze se zpravidla označuje ES6 či ES2015. Do budoucna je pak v plánu verze sedmá, o které toho zatím není mnoho známo.

# Metodika

O jazyku JavaScript bylo napsáno mnoho knih; běžný popis či referenční příručka by tak byla jen nošením dříví do lesa. V této knize není cílem čtenáře seznámit se všemi aspekty jazyka, protože v praxi se vývojář jen málokdy potká s nutností ovládat JavaScript na tak komplexní úrovni. Namísto toho chceme na praktických úlohách naznačit, kudy by mohlo vést řešení a jaké konstrukce (syntaktické a algoritmické) se nabízejí a hodí.

V některých vývojářských komunitách se čas od času konají události nazvané *Code Retreat*: jedná se o krátká jednodenní školení či tréninky, při kterých se účastníci zdokonalují ve svých dovednostech opakovaným řešením té samé úlohy. Za den ji zvládnou vyřešit třeba desetkrát. Pokaždé s trochu jinými postupy, někdy s ohledem na rychlost, jindy bez použití té či oné vlastnosti jazyka a podobně.

Lehce podobný přístup je k dispozici čtenáři v této knize. Ústředním tématem je konkrétní úloha, která je v průběhu kapitol studována, rozvíjena a vylepšována. Po několika prvních kapitolách je hotové základní řešení; s ním se ale nespokojíme a postupným přidáváním funkcionality nahlédneme i do dalších partií webových technologií, jako je testování, hledání a odstraňování chyb a podobně. Čtenář se proto setká s postupem, který je velmi blízký skutečnému vývoji v praxi – s tím rozdílem, že jednotlivá architektonická a implementační rozhodnutí dopředu zvolil autor knihy, aby na nich demonstroval různé formy vývoje a zajímavé prostředky jazyka.

Stejně tak jako u opravdového vývojářského úkolu ani zde nebudou probrány a ozkoušeny veškeré dostupné vlastnosti jazyka a souvisejících technologií. Naše úloha však byla zvolena tak, aby pokryla co možná nejširší spektrum běžných programátorských činností, a měla by tak být přínosná pro všechny zájemce o JavaScript.

KAPITOLA 2

# Pravidla hry

**V této kapitole:**
- Herní strategie
- Varianty pravidel
- Existující implementace

V této kapitole si popíšeme úlohu, jejíž řešení je náplní knihy. Jedná se o implementaci poměrně málo známé deskové hry, která nemá jednoznačný název – objevuje se pod anglickými jmény *Chain reaction*, *Atoms*, *Atomic reaction* a dalšími. My budeme hře říkat prostě Atomy.

Koncept hry coby programátorské úlohy je zvolen záměrně. Úkoly z kurzů programovacích jazyků jsou často nudné a nezáživné; jde o algoritmizační či matematické hříčky a problémy, které dokážou zaujmout nadšence, ale ne široké publikum. Bývají totiž zpravidla buď příliš abstraktní („napište program hledající miliontou cifru desetinného rozvoje Ludolfova čísla“), nebo neužitečné či příliš specifické („napište program řešící úlohu obchodního cestujícího“). Naše herní úloha má jasně definované a dobře uchopitelné zadání; zároveň se jedná o užitečný úkol, protože jeho výstupem je fungující aplikace. Lze ji hrát ve dvou i více hráčích, experimentovat s pravidly, hledat dobrou strategii a podobně. V pozdějších kapitolách knihy se též objeví téma umělé inteligence, a tak i možnost naprogramovat si vlastního protivníka (a buď jej se slávou porazit, nebo mu – s nemenší radostí – podlehnout).

Rigorózní pravidla pro Atomy neexistují, v průběhu let vzniklo nespočet variant a žádná kanonická verze není k dispozici. My se budeme držet těchto základních stavebních kamenů hry:

1. Hry se účastní alespoň dva hráči, kteří se pravidelně střídají v tazích.
2. Hrací plocha má podobu čtvercové mřížky velikosti $6 \times 6$ polí (buněk).
3. Buňka hracího pole může být buď prázdná (tak je tomu u všech buněk na začátku hry), nebo může obsahovat několik *atomů* (kuliček). Každý atom má barvu podle hráče, kterému přísluší. **V buňce smí být jen atomy stejné barvy.**
4. Hráč, který je na tahu, musí do jedné buňky vložit tzv. *atom* (kuličku) své barvy. Atom lze vložit buď do prázdné buňky, nebo do buňky obsahující atomy hráče na tahu. Nelze tedy atom přidat do buňky s atomy jiné barvy.

5. Pokud hráč nemůže táhnout (tj. na hrací ploše neexistují žádné prázdné buňky a v žádné buňce hráč na tahu nemá své atomy), prohrál. Při hře dvou hráčů to značí vítězství oponenta; u více hráčů se pokračuje do té doby, než zbude jediný.
6. Pro každou buňku hracího pole existuje takzvané *kritické množství* atomů, které je definováno jako počet sousedů (v čtyřsousednosti). Čtyři rohové buňky mají tedy kritické množství 2, šestnáct buněk po straně hrací plochy má kritické množství 3, ostatní (vnitřní) buňky pak 4.
7. Pokud v rámci tahu hráč vložením atomu překoná kritické množství atomů v buňce, dojde okamžitě k jejímu *rozpadu*: do každé sousední buňky přeskočí jeden atom z té rozpadající se. V původní buňce tak (většinou) zůstane jeden atom. Během rozpadu se může stát, že sousední buňka již obsahuje atomy – stejné či odlišné barvy. Pokud je barva odlišná, v rámci přeskoku atomu dojde k *přebarvení* všech atomů v sousední buňce na barvu hráče na tahu. Znamená to, že tento rozpad je jediným způsobem, jak může hráč získat buňku, která patří jeho soupeři.
8. Po dokončení rozpadu buňky mohou na herní ploše zůstat další nadkritické buňky (nadkritická množství atomů se v nich objevila v rámci přeskoku). I tyto buňky se nyní musí rozpadnout (je jedno, v jakém pořadí); tento proces se opakuje do té doby, než všechny buňky obsahují nejvýše kritické množství atomů.

# Herní strategie

Autorovi knihy není znám žádný výzkum, který by se věnoval otázce pravidel hry a její rozhodnutelnosti, výherní strategie a podobně. Cílem hry je přirozeně zabrání všech buněk hracího pole, ale není patrné, jestli by se tak mělo postupovat už od začátku (tj. nejprve vložit do co nejvíce buněk po jednom atomu), či naopak nejprve co nejlépe zaplnit menší počet buněk a expandovat až v rámci (řetězových) reakcí.

Stejně tak se nabízí otázka, jestli je lepší oddalovat interakci s dalšími hráči až do explozivního finále hry, nebo vyhledávat časté šarvátky již v jejím průběhu. Pravidla hry totiž dovolují pokládat atomy zcela nesouvisle do vzdálených (prázdných) buněk a pokrýt tak značnou část hrací plochy.

Další prostor pro herní taktiku nabízí odlišná kritická množství atomů v buňkách. Zabíráním rohových polí se dá docílit rozpadu již po třech tazích a obecně pole u krajů hrací plochy dovolují rychlejší *zbrojení* a přípravu na případnou reakci v porovnání s buňkami *ve vnitrozemí*.

Atomy jistě neobsahují tolik taktické hloubky jako některé jiné hry, například dáma či šachy. I tak je k dispozici dostatek možností pro budování strategie a hra ani po mnoha partiích nepůsobí *ohraným* dojmem – zejména zvážíme-li možnost účasti více hráčů.

# Varianty pravidel

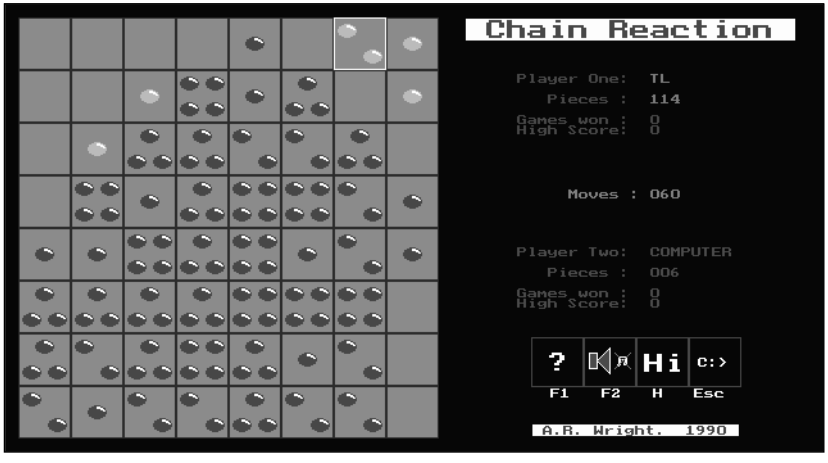
Kromě základních pravidel výše se nabízejí i další drobné varianty, které mohou hru zpestřit či upravit. V knize se na ně nebudeme soustředit, ale zvídavý čtenář si je může zkusit – povětšinou velmi snadno – naimplementovat *za domácí úkol*.

- Hrací plocha může mít libovolné rozměry (i obdélníkové); často se objevuje velikost $8 \times 8$. Při minimální hrací ploše $2 \times 2$ mají všechny buňky stejné kritické množství a tento *degenerovaný* případ se může hodit pro testovací a ladicí účely.
- V některých implementacích se objevují kritická množství o jedničku nižší. Při rozpadu buňky se tak většinou stane, že centrální pole zůstane prázdné (tj. opět volné pro zabrání libovolným hráčem).
- Podobně je též možné uvažovat osmisousednost, tedy i rozpad do diagonálních směrů. Kritická množství pak adekvátně narostou.
- Kromě vkládání nových atomů lze v rámci tahu povolit i přesun atomu z jedné buňky do jiné. Tuto formu tahu můžeme omezit například jen na přesun do sousední buňky.
- Při aplikaci předchozího pravidla je možné hráče vybavit pevným (konečným) množstvím atomů. Jakmile jim atomy dojdou, jsou odkázáni jen na přesouvání po hrací ploše.

# Existující implementace

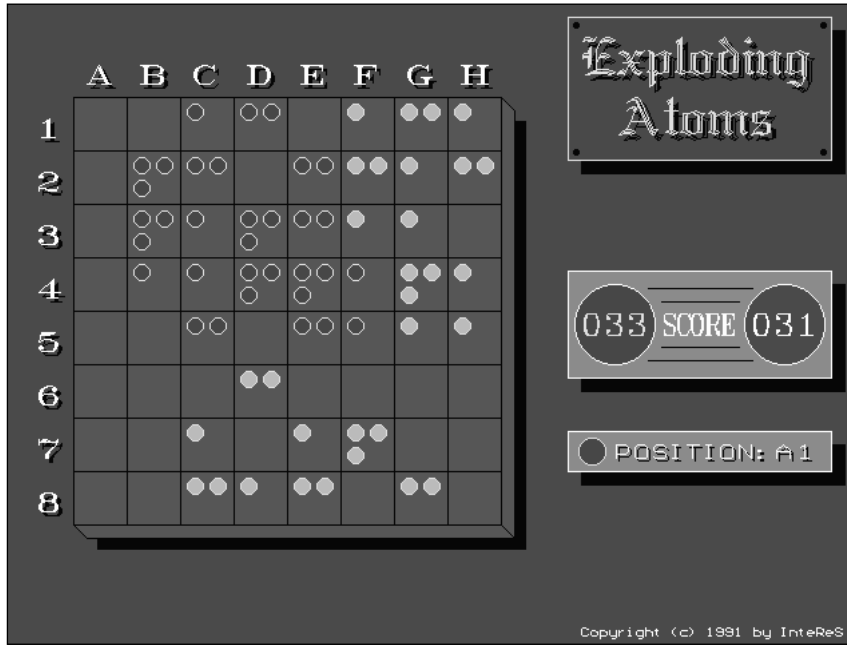
Pojďme se na konec této kapitoly v rychlosti podívat na některé existující implementace konceptu hry Atomy. Jedná se zpravidla o starší programy, protože elektronické verze deskových her zažívaly svůj boom spíše v dřevních dobách počítačové éry.

- **Chain reaction** (autor A. R. Wright) z roku 1990 je patrně první existující verzí pro počítače PC. Program je k dohledání ke stažení na Internetu.



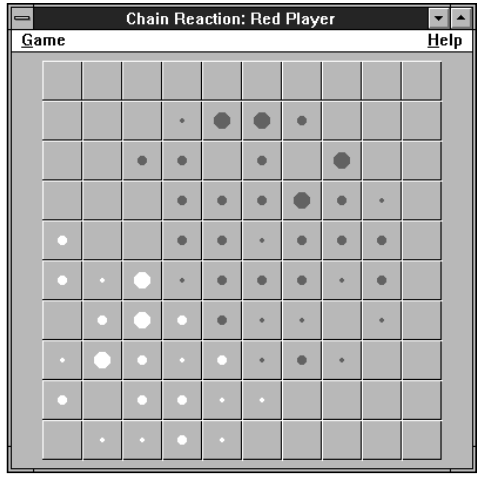

**Obrázek 2.1** Chain reaction, A. R. Wright

- **Exploding Atoms** je dílem českého programátora Pavla Poly. Hru napsal v letech 1991–92 ve svých šestnácti letech.



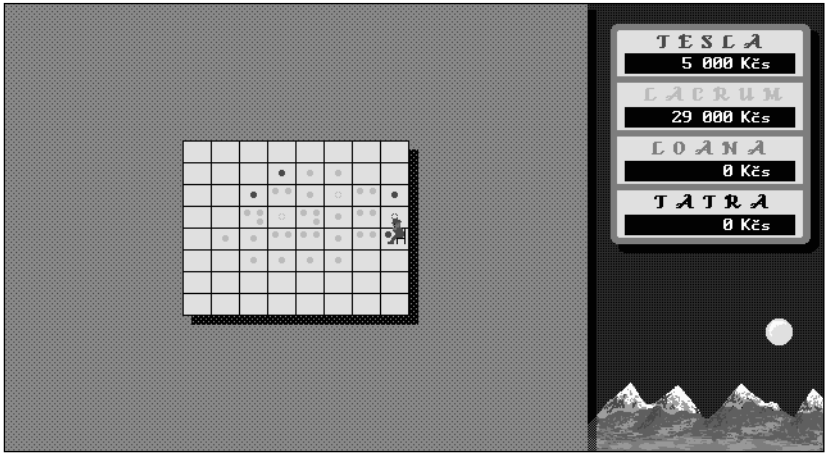

**Obrázek 2.2** Exploding Atoms, Pavel Pola

- **Chain reaction** (autor Jonas Olsson) pochází ze Švédska. Hra byla napsána pro Windows 3.1 a využívá velkou hrací plochu. Namísto více atomů v buňce je vykresleno kolečko, jehož velikost je úměrná počtu atomů.



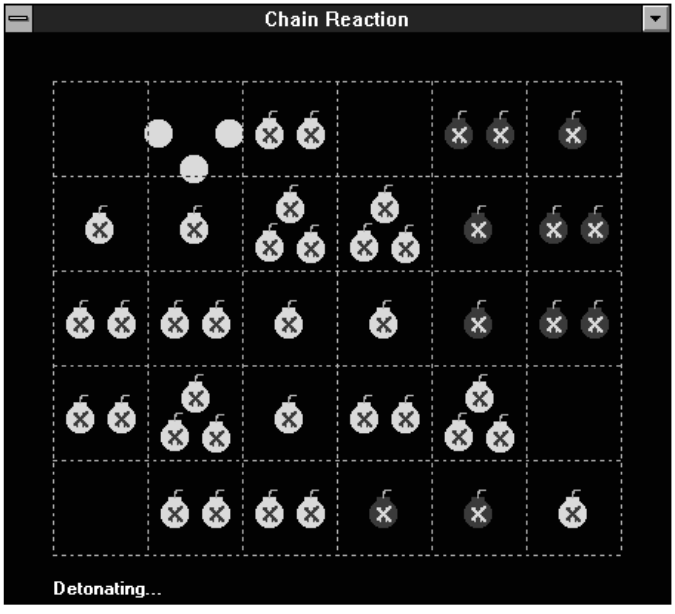

**Obrázek 2.3** Chain reaction, Jonas Olsson

- **Velká privatizace** (1992) je kuriozitou, kdy je koncept hry Atomy realizován v rámci privatizačního procesu (herní políčka jsou jednotlivé privatizované podniky). Autorem je známý programátor Miroslav Němeček, autor proslulé hry Vlak.




**Obrázek 2.4** Velká privatizace, Miroslav Němeček

- **Chain reaction** (autor Neil Fraser) patří mezi novější verze – vznikla v roce 2003. Nabízí poměrně malou herní plochu a je spustitelná ve všech verzích Windows počínaje 3.1. Autor v roce 2006 přepsal kód do Javy, a hru tak učinil hratelnou i v rámci webového prohlížeče (pomocí techniky Java appletu).




**Obrázek 2.5** Chain reaction, Neil Fraser

KAPITOLA 3

# Než se dáme do práce

**V této kapitole:**
- Jak pracovat s knihou
- Terminologie
- Opáčko základů JavaScriptu
- Pár slov o prohlížečích
- Coding style
- Test na závěr kapitoly

Před každým projektem, větším i menším, je dobrý nápad na chvíli zpomalit a obhlédnout *bitevní pole*. Bez dobré přípravy je zbrklá akce jen ztrátou času. Pojďme se podívat, jakým způsobem začneme souboj s JavaScriptem a souvisejícími vývojářskými technikami. Skutečný kód proto napíšeme až v další kapitole – ale dobře připraveni.

## Jak pracovat s knihou

Cílem této knihy je nabídnout zvídavému čtenáři pohled do zákulisí práce s webovými technologiemi a naučit ho, jak převést myšlenku do podoby JavaScriptového kódu. Pro tento účel je kniha členěna do kapitol, které odpovídají jednotlivým fázím vývoje softwarového produktu – konkrétně hry Atomy. Ke každé kapitole tak přirozeně patří celá sada zdrojových kódů odpovídajících stavu projektu během zmiňované kapitoly.

Tyto zdrojové kódy jsou k dispozici hned na několika místech: jednak v archivu ke stažení na webu nakladatelství (na stránce věnované této knize), druhak ve veřejném úložišti zdrojového kódu na serveru GitHub na adrese *https://github.com/ondras/javascript*. Tam k nim lze přistupovat jak běžným webovým prohlížečem, tak i pomocí specializovaného programu `git`, sloužícího právě pro správu zdrojových kódů. Vposled je valná většina kódu též po částech zkopírována do knihy samotné, aby se dalo snadno odkazovat na probírané techniky.

Na čtenářově rozhodnutí zůstává, jestli se rozhodne nejprve kapitoly celé pročítat, nebo rovnou studovat související soubory a zkoušet je otevírat v prohlížeči, vylepšovat a dále upravovat. Tak či onak je pro kompletní pochopení všech ukázek rozhodně dobrý nápad využít počítač (a v něm webový prohlížeč) a věci si v něm, dříve či později, vyzkoušet. Jen čtením knih se, jak známo, ještě nikdo programovat nenaučil.

# Terminologie

Při popisování úryvků kódu (což je v průběhu dalších kapitol poměrně častá činnost) je dobré při pojmenování dodržovat jisté konvence a postupovat konzistentně. Abychom předešli případnému zmatení, pojďme vyjasnit tato potenciálně problematická místa:

- Od čtenáře se očekává znalost alespoň těch nejzákladnějších termínů z algoritmizace a základů programování. Běžně tak bez upřesnění používáme termíny jako *proměnná*, *pole*, *funkce* a podobné.
- Jazyk HTML je postaven na *značkách* (píšou se mezi většítka a menšítka, `<takto>`). Když následně mluvíme o *prvcích*, máme na mysli konkrétní realizace značek v paměti prohlížeče. V zápisu zdrojového kódu je proto kupříkladu *prvek* odstavce zapisován pomocí *značky* `<p>`.
- Jazyky s klasickým (třídním) objektovým modelem jednoznačně definují pojmy *funkce* a *metoda*. V JavaScriptu je ale situace jiná (vysvětleno to bude v šesté kapitole), a tak tato označení používáme dle aktuálního kontextu. To, co je jednou funkce, je jindy metoda a naopak.
- Termín *signatura funkce* označuje počet a typy parametrů, někdy zahrnuje i typ návratové hodnoty.

# Opáčko základů JavaScriptu

Většinu JavaScriptových konstrukcí v ukázkách kódu popíšeme a dostatečně vysvětlíme. Pojďme si rychle projít základní vlastnosti jazyka, ať se s nimi později nemusíme zdržovat:

- Proměnné jsou buď globální (tj. dostupné odkudkoliv), nebo lokální (definované v rámci funkce). Lokální jsou k dispozici jen uvnitř funkce, kde jsou definovány.

```
var a = 3;

function f() {
  var b = 5;
  return a+b;
}

f(); // 8
alert(b); // Chyba! Proměnná "b" je k dispozici jen uvnitř funkce
```

- Jednoduché datové typy (číslo, řetězec, bool, undefined a null) jsou do funkcí *předávány hodnotou*. To znamená, že když je použijeme jako parametr, tak je při volání jejich hodnota zkopírována a uvnitř funkce se pracuje s touto kopií. Složitější datové typy (ty ostatní – pole, objekty, funkce a další) jsou *předávány odkazem*; pokud je jako parametry uvnitř funkce změníme, projeví se to i na hodnotě mimo funkci. Toto chování nelze ovlivnit.

```
function add(array, value) {
  array.push(value);
  value += 1;
}

var array = [1, 2, 3];
var value = 4;
add(array, value);

alert(array); // [1, 2, 3, 4]
alert(value); // 4
```

- Funkce je datový typ. Funkce tak můžeme ukládat do proměnných a předávat jako parametry.

```
var f = function() {
  alert("Hi");
}

setTimeout(f, 1000); // vykonat f za 1000 msec
```

- Definice funkcí do sebe můžeme zanořovat. Lokální proměnné z vnější funkce jsou k dispozici i ve funkci vnitřní (dochází k takzvané *uzávěře*).

```
function outer(a) {
  function inner(b) {
    return a+b;
  }
  return inner;
}

var f = outer(3);
f(5); // 8
```

Pro čtenáře tápající i v těchto krátkých ukázkách je na webu k dispozici nepřeberné množství textů a začátečnických JavaScriptových lekcí. Autor knihy může jako velmi kvalitní zdroj informací doporučit (anglický) web *https://developer.mozilla.org/*.

# Pár slov o prohlížečích

Dnes (počátek roku 2015) jsou uživatelům k dispozici webové prohlížeče na skutečně vysoké technologické úrovni. Většina výrobců prohlížečů svůj software aktualizuje velmi často (mluvíme pak o tzv. *evergreen* prohlížečích), respektované standardizační instituce (WhatWG, W3.org) dohlíží na vznik a specifikaci nových rozhraní. Nebudeme si proto s otázkou tzv. *cross-browser* compatibility lámat hlavu; namísto toho se spokojíme s konstatováním, že v aktuálních verzích moderních prohlížečů (Mozilla Firefox, Google Chrome, Apple Safari, Microsoft Internet Explorer) bude veškerý kód v knize fungovat bez problémů.

Ve výčtu absentuje prohlížeč Opera – to proto, že jeho aktuální verze jsou z větších částí postaveny na komponentách Google Chrome, a proto s tímto prohlížečem téměř identicky sdílí funkcionalitu.

Připomeňme však, že historicky situace vždy tak jednoduchá nebyla. V dřevních dobách webových aplikací patřila otázka kompatibility napříč prohlížeči k těm nejdůležitějším. Kód napsaný a otestovaný v prohlížeči **X** se mohl klidně ukázat jako kompletně nefunkční v jiném prohlížeči **Y**; něco takového samosebou vývoj brzdilo a notně komplikovalo. Důvodů pro takové chování existovalo hned několik:

- Malá míra centralizované standardizace webových rozhraní; HTML značek, CSS selektorů a vlastností, JavaScriptových API. Když tak chtěl výrobce prohlížeče nabrat konkurenční výhodu, navrhl si pro vlastní potřeby API nové, nekompatibilní.
- Nízká frekvence aktualizací verzí prohlížečů. Díky tomu mohlo trvat třeba i několik let, než se naprogramovaná funkcionalita dostala mezi uživatele.
- Bouřlivý rozmach dynamických webových stránek v kontrastu s malou množinou existujících rozhraní. Pro pokročilejší problémy a techniky tak neexistovaly jednotné postupy k jejich řešení, což vedlo ke vzniku mnoha řešení navzájem nekompatibilních.

Tyto problémy s časem buď vymizely, nebo se nenápadně transformovaly do otázky kompatibility v desítkách a desítkách rozmanitých mobilních zařízení. Pro naše potřeby se naštěstí orientujeme jen na plnohodnotné prohlížeče desktopových počítačů.

A ještě pozor – v závěrečných kapitolách knihy se objevují dvě velmi moderní (místy až experimentální) rozhraní, WebGL a Web Audio API. U nich je podpora v prohlížečích slabší, a tak se může stát, že kód ve starší verzi prohlížeče (nebo v závislosti na hardwaru a jeho ovladačích) fungovat nebude.

# Coding style

Zdrojové kódy v knize drží společnou štábní kulturu, takzvaný *coding style*. Bylo by pěkné, kdyby existovala jednotná pravidla pro zápis kódu – usnadnilo by to jeho čitelnost a spolupráci napříč vývojovými týmy. Něco takového však není možné, protože syntaktická pravidla programovacích jazyků (JavaScript nevyjímaje) většinou dovolují notnou dávku flexibility. Ve výsledku se pak často stává, že si každý vývojář navykne na nějakou sadu vlastních pravidel zápisu. Ta se snaží konzistentně dodržovat v celém projektu, ale takovýchto stylů je pak skoro stejně tolik, jako je vývojářů.

Situaci komplikuje skutečnost, že k dobře napsanému zdrojovému kódu vývojáře často váže takřka mateřská hrdost; málokdo je pak ochoten přistoupit na změnu stylu zápisu jen kvůli konzistenci (třeba s ostatními členy týmu). Čtenář knihy má v této otázce naštěstí volnou ruku, a pokud se rozhodne samostatně přistoupit k implementaci, je

pánem svého kódu. Při čtení knihy a souvisejících souborů však nezbude než respektovat styl autorův.

Pojďme se podívat, jaká pravidla zápisu zdrojového kódu platí v této knize.

- Řádky kódu jsou zleva odsazeny jedním tabulátorem za každou úroveň zanoření. Ve většině programovacích jazyků lze jako odsazení použít tabulátory nebo mezery; JavaScript je jedním z mála, ve kterém ještě užívání sudého počtu mezer zcela nepřeválcovalo tabulátory. Ne ve všech jazycích je tato benevolence: někdy je používání tabulátorů nutnost (Makefile), někdy to naopak nelze (YAML).
- Otevírací složené závorky jsou na konci řádku před otevřeným blokem kódu či definicí objektu. To je dobrá praxe s ohledem na ASI (Automatic Semicolon Insertion; vlastnost jazyka JavaScript způsobující automatické vkládání středníků na konce řádků). Tento kód by totiž v JavaScriptu napáchal pěkný nepořádek:

```
function f() {
  return
    {
      hello: "world"
    }
}
```

  Díky ASI by došlo k vložení středníku za klíčové slovo `return`, a funkce by tak vrátila `undefined`.

- Složené závorky jsou použity i pro ohraničení bloku kódu s jediným příkazem; tento se ale zpravidla vejde na jeden řádek i s oběma závorkami.
- Názvy proměnných začínají malými písmeny, víceslovné názvy se spojují pomocí camelCase (odstranit mezery, od druhého slova dál s prvním velkým písmenem). Názvy konstant jsou velkými písmeny, První Velké Písmeno se používá pro globální objekty a funkce, ze kterých lze vytvářet instance objektů.
- Pole a *objekty* (jedná se o lehce matoucí pojmenování, ve skutečnosti jde o struktury typu klíč-hodnota) jsou definovány pomocí znaků `[]` a `{}`. Zápis `new Array()`, resp. `new Object()`, je funkčně identický a zbytečně zdlouhavý.
- Pole jsou iterována tradičním způsobem pomocí cyklu `for`. V JavaScriptu lze používat též *funkcionální iteraci* (metody `map`, `forEach`, `filter` a další); tato technika je zajímavou alternativou, ale nepřináší nic koncepčně nového, a proto se budeme držet klasického přístupu.
- V kódu je jen velmi málo komentářů. Dobře komentovaný kód je vždy pěkná vizitka programátora, ale v případě této knihy je kód více než zevrubně popsán v knize samotné. Další komentáře v kódu by tak spíše překážely.
- Pokud se v kódu přece jen čas od času komentáře objeví, jsou psány česky a s diakritikou. To není příliš obvyklá praxe, ale našinci to usnadní čtení.

- Názvy proměnných, funkcí, objektů a jejich vlastností jsou anglicky. Působí totiž velmi nezvykle, když dochází k míchání (anglických) klíčových slov typu `function`, `replace` či `length` a českých názvů proměnných.

# Test na závěr kapitoly

Znalost jazyka (a zároveň připravenost pro další čtení) je nyní možno ověřit na krátkém příkladu. Ideálně bychom měli přesně chápat každý řádek této funkce, určené k výpočtu n-té hodnoty Fibonacciho posloupnosti (hodnota je definována jako součet dvou předchozích) s mezipamětí, do které si pro urychlení ukládáme mezivýsledky:

```
/* Výpočet Fibonacciho posloupnosti s ukládáním mezivýsledků */
var fib = function(x, cache) {
  /* známe z předchozích výpočtů? */
  if (x < cache.length) { return cache[x]; }

  /* cache se předá odkazem */
  var result = fib(x-1, cache) + fib(x-2, cache);

  /* přidat mezivýsledek */
  cache[x] = result;

  return result;
}

/* posloupnost inicializujme hodnotami 0 a 1 */
var start = [0, 1];

alert(
  /* desátá hodnota */
  fib(10, start)
);
```

Generování hodnot Fibonacciho posloupnosti samosebou není hlavním úkolem této kapitoly ani knihy. Čtenář by teď již ale měl být připraven na výpisy zdrojových kódů a popisy algoritmů, protože to vše začneme používat hned od příští kapitoly.