Edícia výskumných textov
Informatiky a informačných technológií

Umelá inteligencia a kognitívna veda II

Vydané s podporou firmy Gratex International, a.s.

Táto publikácia vznikla za podpory grantovej agentúry VEGA SR v rámci grantových úloh VG1/0804/08 a VG 1/0243/10

Vladimír Kvasnička
Jiří Pospíchal
Pavol Návrat
Peter Lacko
Peter Trebatický
(editori)

Umelá inteligencia a kognitívna veda II

Slovenská technická univerzita v Bratislave, 2010

PUBLIKÁCIU PODPORILO ZDRUŽENIE

GRATEX IT INŠTITÚT

v rámci fondu GraFIIT www.gratex.com

Príspevky boli recenzované redakčnou radou edície *Umelá inteligencia a kognitívna veda.*

Schválilo vedenie Fakulty informatiky a informačných technológií STU v Bratislave dňa 15.12.2009, uznesenie č. 2009.28.1

ISBN XXX-XX-XXX-XXXX-X

AUTORSKÝ KOLEKTÍV KNIHY

Ing. Štefan Babinec, PhD.
E-mail: stefan.babinec@stuba.sk
Ústav informatizácie, automatizácie a matematiky, Fakulta chemickej a potravinárskej technologie, Slovenská technická univerzita, Radlinského 9, 812 37 Bratislava

Mgr. Michal Bída
E-mail: michal.bida@gmail.com
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

RNDr. Anna Bou Ezzeddine
E-mail: anna.bou.ezzeddine@stuba.sk
Ústav informatiky a softvérového inžinierstva, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4

Mgr. Cyril Brom, PhD.
E-mail: brom@ksvi.mff.cuni.cz
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

Mgr. Vladimír Dziuban
E-mail dziuban@fmph.uniba.sk
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky Univerzita Komenského, Mlynská dolina, 848 48 Bratislava

Mgr. Juraj Frank
E-mail: frank@fmph.uniba.sk
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky Univerzita Komenského, Mlynská dolina, 848 48 Bratislava

Doc. PhDr. Dušan Gálik, CSc.
E-mail: Dusan.Galik@savba.sk
Filozofický ústav Slovenskej akadémie vied, Klemensova 19, 813 64 Bratislava

RNDr. Martin Homola
E-mail: martin.homola@fmph.uniba.sk
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky Univerzita Komenského, Mlynská dolina, 848 48 Bratislava

Mgr. Martin Klíma
ext. spolupracovník Mgr. Broma
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

Mgr. Jakub Gemrot
E-mail: jakub.gemrot@gmail.com
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

Mgr. Rudolf Kadlec
E-mail: rudolf.kadlec@gmail.com
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

Ing. Matej Košík
E-mail: matej.kosik@fiit.stuba.sk
Ústav aplikovanej informatiky, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4

prof. Ing. Štefan Kozák, CSc.
E-mail: kozak@stuba.sk
Ústav aplikovanej informatiky, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4 a

Ústav riadenia a priemyselnej informatiky, Fakulta elektrotechniky a informatiky, Slovenská technická univerzita, Ilkovičova 3, 812 19 Bratislava

prof. Ing. Vladimír Kvasnička, DrSc.
E-mail: kvasnicka@fiit.stuba.sk
Ústav aplikovanej informatiky, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4

doc. RNDr. Ladislav Kvasz, PhD.
E-mail: ladislav.kvasz@ku.sk
Katedra matematiky a didaktiky matematiky, Ped. Fakulta Univerzity Karlovy, 116 39 Praha a

Katedra filozofie, Filozofická Fakulta Katolíckej Univerzity, 03 401 Ružomberok

doc. Ing. Marián Mach, CSc.
E-mail: Marian.Mach@tuke.sk
Katedra kybernetiky a umelej inteligencie, Fakulta elektrotechniky a informatiky, Technická univerzita, Letná 9, 042 00 Košice

doc. RNDr. Mária Markošová, PhD.
E-mail: markosova@fmph.uniba.sk
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky Univerzita Komenského, Mlynská dolina, 848 48 Bratislava

prof. Ing. Pavol Návrat, PhD.
E-mail: navrat@fiit.stuba.sk
Ústav informatiky a softvérového inžinierstva, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4

Bc. Radek Píbil
E-mail: black_hand@centrum.cz
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

Mgr. Tomáš Plch
E-mail: tomas.plch@gmail.com
Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta, Univerzita Karlova, Malostranské nám. 25, 118 00 Praha 1

prof. RNDr. Jiří Pospíchal, DrSc.
E-mail: pospichal@fiit.stuba.sk
Ústav aplikovanej informatiky, Fakulta informatiky a informačných technológií Slovenská technická univerzita Ilkovičova 3, 842 16 Bratislava 4

Mgr. Kristína Rebrová
E-mail: kristina.rebrova@fmph.uniba.sk
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky Univerzita Komenského, Mlynská dolina, 848 48 Bratislava

Doc. Ing. Lukáš Sekanina, PhD.
E-mail: sekanina@fit.vutbr.cz
Ústav počítačových systémů, Fakulta informačních technologií, Vysoké učení technické v Brně, 612 66 Brno

Mgr. Radovan Šikl, PhD.
E-mail: sikl@psu-cas.cz
Psychologický ústav Akademie věd ČR, Veveří 97, 602 00 Brno

Mgr. Michal Šimeček
E-mail: michal.simecek@volny.cz
Psychologický ústav Akademie věd ČR, Veveří 97, 602 00 Brno

Mgr. Daniel Svozil, PhD.
E-mail: daniel.svozil@gmail.com
Laboratoř informatiky a chemie, Vysoká škola chemicko-technologická, 166 28 Praha

Táto kniha je venovaná pamiatke zosnulého
Prof. Ing. Vladimíra Vojteka, CSc. (∗1954 –† 2009)

Predhovor

Umelá inteligencia a kognitívna veda sú už dobre inštitucionalizované vedné odbory, ktoré majú interdisciplinárne až transdisciplinárne zameranie. Dávajú rigorózne zameraným informatikom krídla fantázie, umožňujúce presah ich technickej/ prírodovednej disciplíny smerom k humanitným a behaviorálnym vedám.

Na Slovensku tieto dva odbory majú už svoje študijné programy: umelá inteligencia v Košiciach na FEI TU a kognitívna veda v Bratislave na FMFI UK. Už desať rokov sa na česko-slovenskej, alebo slovensko-českej konferencii *Kognícia a umelý život* schádzajú odborníci z rôznych vedných oblastí (informatici, matematici, filozofi, psychológovia, lekári,...), ktorých zjednocuje záujem o poznanie kognitívnych procesov, prebiehajúcich v ľudskom mozgu a o ich simuláciu metódami modernej umelej inteligencie: neurónových sietí, evolučných algoritmov a multiagentových systémov.

Edícia *Umelá inteligencia a kognitívna veda* sa zameriava na prezentáciu výsledkov tejto komunity, dosiahnutých v umelej inteligencii a v kognitívnej vede, s cieľom pomôcť študentom najmä bakalárskeho, ale aj pokročilejšieho štúdia.

Obsah predkladanej knihy *Umelá inteligencia a kognitívna veda II* dokumentuje, že tieto vedné odbory našli pevné zakotvenie tak vo filozoficky, sociálnovedne, ako aj prírodovedne a informaticky orientovanej komunite pracovníkov, v oblastiach filozofie mysle, kognitívnej psychológie, umelej inteligencie, kognitívnej vedy a umelého života.

Ďakujeme všetkým autorom, ktorí prispeli svojimi príspevkami do tejto knihy, spoločnosti Gratex International, a.s., reprezentovanej Ing. Ivanom Polášekom, PhD., za finančnú podporu vydania tejto knihy a vedeniu Fakulty informatiky a informačných technológií STU v Bratislave, menovite dekanovi fakulty prof. RNDr. Ľudovítovi Molnárovi, DrSc., za podporu vydávania tejto edície a jej zaradenie do fakultného edičného radu „Edícia výskumných textov“.

Túto knihu venujeme pamiatke zosnulého kolegu a priateľa prof. Ing. Vladimíra Vojteka, CSc. (∗1954 - †2009), riaditeľa Ústavu aplikovanej informatiky FIIT STU, ktorý väčšine z nás veľmi priateľsky “šéfoval” a mal veľké pochopenie pre naše aktivity v umelej inteligencii a kognitívnej vede.

V Bratislave, marec 2010.

Vladimír Kvasnička
Jiří Pospíchal
Pavol Návrat
Peter Lacko
Peter Trebatický
(editori)

Obsah

Rekurentné neurónové siete s echo stavmi

Štefan BABINEC [1]

Abstrakt. Adaptácia rekurentných neurónových sietí je zložitý proces, pričom nie je garantované, že dynamika dosiahne požadované správanie. Na druhej strane aj náhodne inicializovaná rekurentná sieť má už pred začiatkom trénovania zmysluplnú dynamiku a je schopná „predspracovať“ vstupné postupnosti a signály. Netrénovanú rekurentnú sieť možno použiť v rámci komplexnejšej architektúry, a tak využiť predspracovanie vstupu na dosiahnutie požadovaného výsledku. Práve tieto poznatky viedli k vytvoreniu špecifického druhu rekurentných neurónových sietí, označovaných ako Echo State neurónové siete, alebo neurónové siete s echo stavmi.

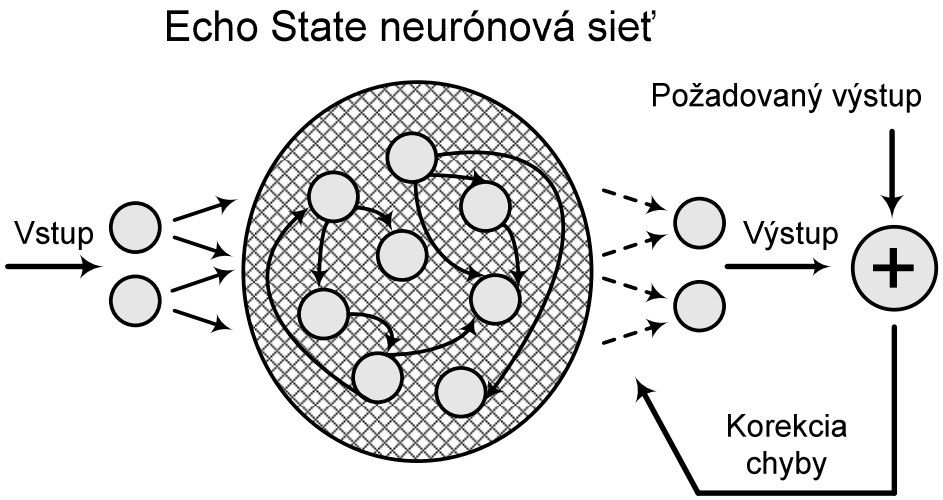
1 Úvod do Echo State neurónových sietí

Echo State neurónová sieť (ESN) je z architektonického hľadiska rovnaká ako klasická rekurentná sieť. Obsahuje vstupné, skryté a výstupné neuróny organizované do vstupnej, skrytej a výstupnej vrstvy. Hlavný rozdiel spočíva v rekurentnej časti siete, ktorá je v prípade ESN tvorená veľkým počtom zvyčajne riedko poprepájaných neurónov s netrénovanými spojeniami (dynamický rezervoár – DR). K adaptácii váh dochádza iba vo výstupnej vrstve, ktorej úlohou je využiť predspracovanie vstupu náhodne inicializovaným dynamickým rezervoárom k produkovaniu požadovaného výstupu (obr. 1).

Pri klasických rekurentných neurónových sieťach (RC ANN – Recurrent Artificial Neural Network, obr. 2) je z hľadiska časovej a výpočtovej náročnosti trénovania najproblematickejšou časťou práve rekurentná vrstva siete. Keďže v prípade ESN táto časť nie je trénovaná, môže obsahovať veľký počet neurónov bez toho, aby sa výrazne zvýšila časová náročnosť trénovania. Dynamický rezervoár teda obsahuje stovky neurónov, zatiaľ čo pri klasických sieťach je veľkosť rekurentnej časti siete z uvedeného dôvodu obmedzená rádovo na desiatky neurónov. Ako bude ukázané neskôr, pri trénovaní váh výstupnej vrstvy nie je potrebné šíriť chybový signál späť v čase, a preto sa na ich trénovanie využíva algoritmus lineárnej regresie. Prínosom je podstatne nižšia výpočtová zložitosť, ako aj fakt, že nevzniká problém lokálnych

[1] Fakulta Chemickej a Potravinárskej Technológie, Slovenská Technická Univerzita, Oddelenie Matematiky, Radlinského 9, Bratislava 812 37, E-mail: stefan.babinec@stuba.sk

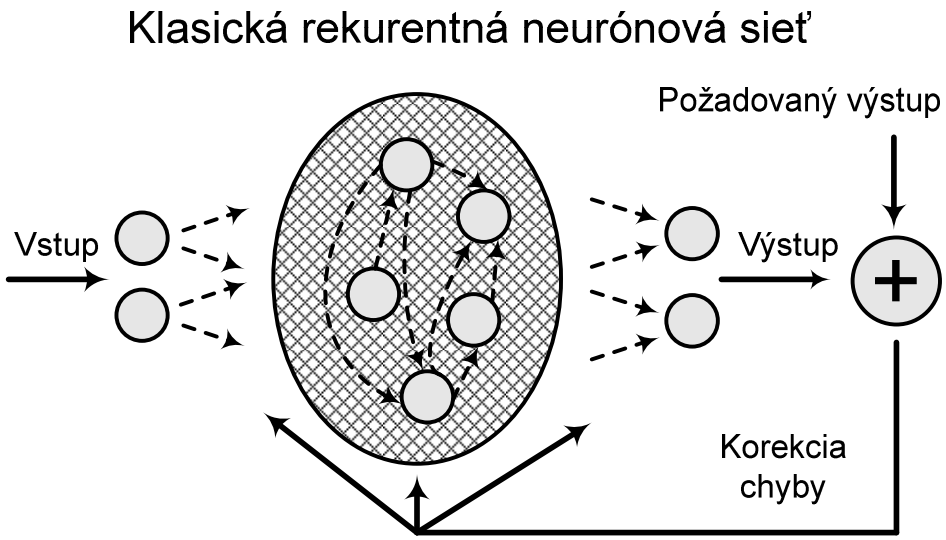
miním, ktoré sú známym obmedzením algoritmov založených na minimalizácii chyby metódou najprudšieho spádu [5].



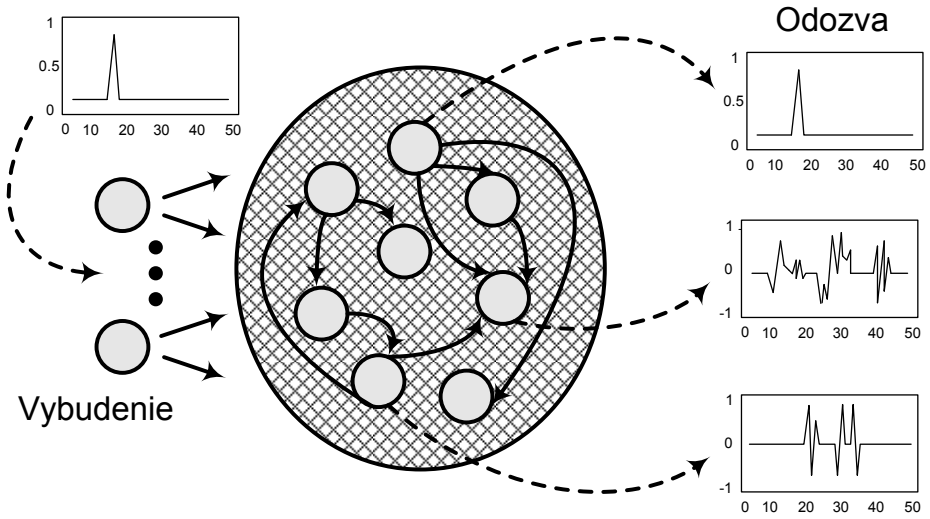
Obr. 1. Echo State neurónová sieť s netrénovanou náhodne inicializovanou rekurentnou vrstvou, takzvaným dynamickým rezervoárom. K adaptácii váh dochádza iba medzi skrytou a výstupnou vrstvou.

Echo State neurónové siete sa teda na rozdiel od klasických sietí nepokúšajú dosiahnuť požadované správanie (dynamiku) adaptáciou váh relatívne malého počtu neurónov, ale využívajú dynamiku mohutnej (náhodne inicializovanej) skrytej vrstvy na extrahovanie zaujímavých vlastností zo vstupnej postupnosti. Úlohou výstupných neurónov je transformovať tieto vlastnosti (aktivity neurónov v dynamickom rezervoári) do požadovaného výstupu.

Proces extrahovania zaujímavých vlastností dynamickým rezervoárom je znázornený na obr. 3. Vstupný signál sa privedie prostredníctvom vstupných neurónov do skrytej vrstvy, kde následne dôjde k interakcii veľkého počtu náhodne poprepájaných neurónov. Takto je výsledná odozva dynamického rezervoára v porovnaní so vstupným signálom podstatne informačne „bohatšia“. V dynamickom rezervoári sa typicky nachádzajú stovky neurónov a ich aktivity reprezentujú obraz vstupného signálu v mnohorozmernom stavovom priestore, ktorý zároveň tvorí priestor zaujímavých vlastností (feature space). Aj v prípade klasických prístupov k trénovaniu dopredných či rekurentných sietí môžeme hovoriť o priestore zaujímavých vlastností, ktorý tvoria aktivity skrytej, resp. rekurentnej vrstvy. Ten má však, ako už bolo spomenuté, zvyčajne menší počet dimenzií (typicky desiatky neurónov). Klasické prístupy sa snažia trénovaním zmeniť rozloženie stavov (aktivity skrytých neurónov) v tomto priestore takým spôsobom, aby boli výstupné neuróny schopné produkovať požadované výstupy [5]. Existuje aj prístup, veľmi podobný Echo State neurónovým sieťam, ktorý sa volá „Liquid State Machine“ a bol nezávisle vynájdený a skúmaný Wolfgangom Maassom (viac v [6, 1]).



Obr. 2. Klasická rekurentná neurónová sieť obsahujúca vstupnú, skrytú a výstupnú vrstvu. Všetky váhy v tejto sieti sú trénované.



Obr. 3. Úloha dynamického rezervoára spočíva vo vytváraní potenciálne zaujímavých vlastností zo vstupného signálu. Riedko poprepájané neuróny skrytej vrstvy produkujú odozvu, ktorá je „informačne bohatším obrazom" vstupnej postupnosti [2].

2 Architektúra a formálny popis

V tejto časti si zjednotíme označovanie jednotlivých vektorov a matíc, ktoré budeme používať v nasledujúcich kapitolách.

Sieť sa bude skladať z K vstupných neurónov, pričom vstupný vektor si označíme ako:

$$\vec{u}(n) = (u_1(n),...,u_K(n))^t, \tag{1}$$

z N skrytých (vnútorných) neurónov, pričom stavový vektor si označíme ako:

$$\vec{x}(n) = (x_1(n),...,x_N(n))^t, \tag{2}$$

a L výstupných neurónov, pričom výstupný vektor označíme ako:

$$\vec{y}(n) = (y_1(n),...,y_L(n))^t, \tag{3}$$

kde index t označuje transponovanie a n predstavuje časovú jednotku. Zavedieme si ešte vektor požadovaných výstupov, ktorý si označíme ako:

$$\vec{d}(n) = (d_1(n),...,d_L(n))^t, \tag{4}$$

Všetky synaptické váhy v ESN budeme ukladať do takzvaných váhových matíc. Matica vstupných váh bude mať rozmery $N \times K$ a označíme si ju

$$\vec{W}^{in} = (w_{ij}^{in}). \tag{5}$$

Matica skrytých váh bude mať rozmery $N \times N$ a označíme si ju

$$\vec{W} = (w_{ij}) \tag{6}$$

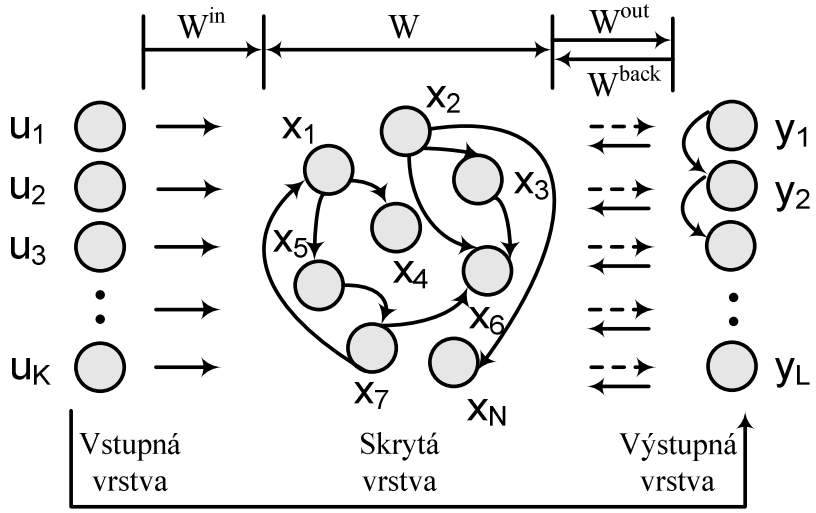
a maticu výstupných váh s rozmermi $L \times (K + N + L)$ označíme

$$\vec{W}^{out} = (w_{ij}^{out}). \tag{7}$$

V niektorých úlohách riešených pomocou ESN môže byť nutné použiť aj spätné synaptické váhy z výstupných neurónov do dynamického rezervoára. Matica týchto váh bude mať rozmery $N \times L$ a označíme ju ako

$$\vec{W}^{back} = (w_{ij}^{back}). \tag{8}$$

Nulovú hodnotu v ktorejkoľvek váhovej matici môžeme interpretovať ako neexistenciu spojenia medzi danými neurónmi. Treba si všimnúť, že matica $\vec{W}^{out}$ nemusí obsahovať iba váhy z vnútorných neurónov. Môžu existovať spojenia aj priamo z vstupných neurónov na výstupné a takisto vzájomné spojenia medzi výstupnými neurónmi.



Obr. 4. Základná architektúra Echo State neurónovej siete. Medzi voliteľné spojenia patria spojenia priamo medzi vstupnou a výstupnou vrstvou a spojenia navzájom medzi jednotlivými výstupnými neurónmi. V niektorých typoch úloh je nutné použiť aj spätné synaptické spojenia z výstupných neurónov do skrytej vrstvy. Šípky v poslednom rade, smerujúce zo skrytej vrstvy (dynamického rezervoára) do výstupnej vrstvy, označujú spojenia, ktoré sa ako jediné trénujú v Echo State neurónových sieťach.

3 Vzťah medzi ESN a neznámym systémom

Z pohľadu systémového inžinierstva, dynamika (neznámeho) systému závisí na nasledujúcom formalizme :

$$\vec{d}(n) = e(\vec{u}(n), \vec{u}(n-1), ..., \vec{d}(n-1), \vec{d}(n-2), ...), \tag{9}$$

kde e je (pravdepodobne vysoko komplexná a nelineárna) funkcia predchádzajúcich vstupov a výstupov nášho systému. Vzťah (9) ukazuje najvšeobecnejší možný spôsob, ako sa dá popísať deterministicky stacionárny systém. V inžinierskych problémoch dochádza často k zjednodušovaniu uvedenej rovnice. Napríklad sa predpokladá, že funkcia e je lineárna a má iba konečnú množinu argumentov (čiže systém má konečnú pamäť). My sa ale budeme držať našej všeobecnej verzie.

Úloha nájdenia vhodného modelu pre neznámy systém (9) nás zavedie k hľadaniu vhodnej aproximácie už spomínanej systémovej funkcie e. Budeme predpokladať ESN s lineárnymi výstupnými neurónmi za účelom zjednodušenia zápisu. Potom výstup trénovanej neurónovej siete je lineárnou kombináciou jej jednotlivých stavov.

$$
\begin{aligned}
& e(\vec{u}(n),\vec{u}(n-1),...,\vec{d}(n-1),\vec{d}(n-2),...) = \\
& = \vec{d}(n) \\
& \approx \vec{y}(n) \\
& = \sum w_i^{out} x_i(n) \\
& = \sum w_i^{out} e_i(\vec{u}(n),\vec{u}(n-1),...,\vec{y}(n-1),\vec{y}(n-2),...).
\end{aligned} \tag{10}
$$

Zo zápisu (10) je jasne viditeľné, že požadovaná aproximácia systémovej funkcie e môže byť interpretovaná ako lineárna kombinácia echo funkcií e_i. Argumenty funkcií e a e_i sú v princípe identické: v obidvoch prípadoch je to kolekcia predchádzajúcich vstupov a systémových výstupov (respektíve výstupov neurónovej siete). Bez existencie echo stavov by sme na jednej strane nemohli matematicky porozumieť vzťahu medzi výstupom neurónovej siete a skutočným výstupom systému a na druhej strane by ani algoritmus trénovania Echo State neurónových sietí nebol funkčný.

4 Matematický náhľad

Hlavným kľúčom k pochopeniu ESN je koncept echo stavov. Existencia (alebo neexistencia) echo stavov v neurónovej sieti vyjadruje schopnosť tejto siete učiť sa (alebo sa neučiť). Pri detailnejšom pohľade zistíme, že táto vlastnosť závisí od váhových matíc $\vec{W}^{in}$, $\vec{W}$ a prípadne, ak existuje, tak aj matice $\vec{W}^{back}$. Táto schopnosť je takisto relatívna vzhľadom k trénovacím dátam. Tá istá netrénovaná sieť môže mať echo stavy pre určité dáta, ale v prípade použitia iných dát už nie. Z tohto dôvodu požadujeme, aby trénovací vstupný vektor $\vec{u}(n)$ pochádzal z kompaktného intervalu U a trénovací výstupný vektor $\vec{d}(n)$ z kompaktného intervalu D. Najskôr si zavedieme presnú matematickú definíciu echo stavov a následne sa pokúsime o intuitívnu interpretáciu tejto definície.

Definícia 1. (Echo Stavy) Predpokladajme netrénovanú neurónovú sieť s váhovými maticami $\vec{W}^{in}$, $\vec{W}$ a $\vec{W}^{back}$.Táto sieť nech je budená učiacim vstupom $\vec{u}(n)$ a učiacim (požadovaným) výstupom $\vec{d}(n)$, ktoré nech sú z kompaktných intervalov U a D. Neurónová sieť ($\vec{W}^{in}$,$\vec{W}$, $\vec{W}^{back}$) má echo stavy vzhľadom na U a D, ak pre každú vstupno/výstupnú sekvenciu $(\vec{u}(n),\vec{d}(n-1))$, zľava idúcu od nekonečna, kde $n = ...,-2,-1,0,$ a pre všetky sekvencie stavov $\vec{x}(n),\vec{x}'(n)$, ktoré dostaneme pomocou vzťahov

$$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}(n) + \vec{W}^{back}\vec{d}(n)) \tag{11}$$

$$\vec{x}'(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}'(n) + \vec{W}^{back}\vec{d}(n)) \tag{12}$$

platí, že $\vec{x}(n) = \vec{x}'(n)$ pre všetky $n \le 0$.

Intuitívne môžeme povedať, že ak neurónová sieť „bežala“ veľmi dlhý čas (od mínus nekonečna), aktuálny stav neurónovej siete je jedinečne určený históriou predkladaných vstupov a požadovaných výstupov. Ekvivalentne môžeme povedať, že pre každý vnútorný signál $x_i(n)$ existuje echo funkcia e_i, ktorá mapuje históriu vstupov a výstupov do aktuálneho stavu:

$$e_i : (U \times D)^{-N} \to R \tag{13}$$

$$(..., (\vec{u}(-1), \vec{d}(-2)), (\vec{u}(0), \vec{d}(-1))) \to x_i(0).$$

Zjednodušene môžeme povedať, že (trénovaná) neurónová sieť $(\vec{W}^{in}, \vec{W}, \vec{W}^{out}, \vec{W}^{back})$ je Echo State neurónovou sieťou ak jej netrénované “jadro“ $(\vec{W}^{in}, \vec{W}, \vec{W}^{back})$ má schopnosť dosiahnuť echo stavy vzhľadom na vstupno-výstupnú sekvenciu z ľubovoľného kompaktného intervalu $U \times D$.

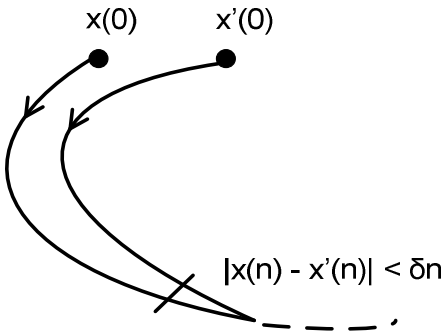
Existuje mnoho iných podmienok, ktoré sú ekvivalentné s podmienkami potrebnými pre dosiahnutie echo stavov. Uvedieme ešte jednu definíciu ekvivalentnú s definíciou 1. Podrobne v [2].

> **Definícia 2**. (Kontraktívnosť). S tými istými predpokladmi ako v definícii 1, neurónová sieť $(\vec{W}^{in}, \vec{W}, \vec{W}^{back})$ je kontraktívna vzhľadom na U a D, ak pre všetky vstupno/výstupné sekvencie $(\vec{u}(n), \vec{d}(n-1)) \in U \times D$ sprava idúce do nekonečna, kde $n = 0,1,2,...$ existuje $(\delta_n)_{n \geq 0}$, také, že pre všetky počiatočné stavy $\vec{x}(0), \vec{x}'(0)$ a pre všetky $n \geq 0$ platí, že $|\vec{x}(n) - \vec{x}'(n)| < \delta_n$, kde $\vec{x}(n)$ (resp. $\vec{x}'(n)$) je stav neurónovej siete v čase *n*.

Intuitívne môžeme povedať, že časom dôjde k takzvanému „vymytiu“ vplyvu počiatočnej inicializácie neurónovej siete (obr. 5).

Schopnosť dosiahnuť echo stavy úzko súvisí s algebraickými vlastnosťami váhovej matice $\vec{W}$. Žiaľ, zatiaľ nie sú známe žiadne nutné a postačujúce podmienky, na základe ktorých by sa dalo jednoznačne rozhodnúť, či pri zadaných $\vec{W}^{in}, \vec{W}, \vec{W}^{back}$ má daná neurónová sieť schopnosť dosiahnuť echo stavy alebo nie.

> **Veta 1.** Predpokladajme netrénovanú sieť $(\vec{W}^{in}, \vec{W}, \vec{W}^{back})$. Stav tejto siete v nasledujúcom kroku vypočítame na základe vzťahu (11) a ako prechodovú funkciu použijeme tanh (hyperbolický tangens). Nech $\vec{W}$ má spektrálny polomer $|\lambda_{\max}| > 1$, kde $|\lambda_{\max}|$ je najväčšia absolútna hodnota vlastného vektora matice $\vec{W}$. Potom táto neurónová sieť nemá schopnosť dosiahnuť echo stavy vzhľadom na žiadnu vstupno/výstupnú sekvenciu z intervalu $U \times D$, obsahujúcu nulovú vstupno/výstupnú sekvenciu (0,0).



Obr. 5. Grafické znázornenie efektu nazývaného „vymytie“ vplyvu počiatočnej inicializácie Echo State neurónovej siete.

Na prvý pohľad sa nám môže zdať, že toto tvrdenie nám nijako nepomôže pri hľadaní Echo State neurónových sietí. Ale praktickými experimentmi bolo zistené, že ak nedodržíme podmienky uvedené vo vete 1, čiže ak spektrálny polomer váhovej matice je menší ako jedna, tak neurónová sieť, ktorú získame týmto spôsobom, má echo stavy.

> **Hypotéza 1.** Nech δ a ε sú dve malé kladné čísla. Potom existuje veľkosť neurónovej siete N taká, že ak dynamický rezervoár o veľkosti N skonštruujeme náhodným vygenerovaním matice $\vec{W}_0$ za použitia uniformného rozdelenia a z rozsahu $[-1,1]$, ďalej túto maticu $\vec{W}_0$ normalizujeme na $\vec{W}_1$, $\vec{W}_1 = \frac{1}{|\lambda_{\max}|}\vec{W}_0$, kde $|\lambda_{\max}|$ je spektrálny polomer matice $\vec{W}_0$ a nakoniec upravíme maticu $\vec{W}_1$ na maticu $\vec{W}_2$, $\vec{W}_2 = (1-\delta)\vec{W}_1$, čím $\vec{W}_2$ získa spektrálny polomer $(1-\delta)$, tak potom neurónová sieť $\vec{W}^{in}, \vec{W}_2, \vec{W}^{back}$ má echo stavy s pravdepodobnosťou $1-\varepsilon$.

Všimnime si, že vo vete 1 a aj v hypotéze 1 sa vôbec nespomína obmedzenie vzťahujúce sa na váhové matice $\vec{W}^{in}$, a $\vec{W}^{back}$. Zdá sa, že hodnoty týchto váh sú nepodstatné pre dosiahnutie echo stavov. V praxi sa ukázalo, že tieto váhy môžu byť ľubovoľne zvolené, bez toho aby to nejakým spôsobom ovplyvnilo schopnosť neurónových sietí dosiahnuť echo stavy.

Starostlivá voľba spektrálneho polomeru váhovej matice dynamického rezervoára $\vec{W}$ má kľúčový dosah na eventuálny úspech trénovania Echo State neurónových sietí. Dôvodom je fakt, že spektrálny polomer je v úzkom kontakte s vlastnou dynamikou skrytej vrstvy Echo State neurónovej siete. Malé hodnoty spektrálneho polomeru vytvoria takzvaný „rýchly“ dynamický rezervoár, veľké hodnoty naopak „pomalý“ dynamický rezervoár. Vlastná dynamika úlohy by mala zodpovedať dynamike Echo State neurónovej siete. Treba poznamenať, že dynamika Echo State neurónovej siete závisí exponenciálne na hodnote $1-$(spektrálny polomer DR). Takže ak budeme voliť

hodnoty spektrálneho polomeru dynamického rezervoára v poradí: 0.99, 0.98, 0.97, bude to viesť k exponenciálnemu a nie k lineárnemu nárastu dynamiky neurónovej siete.

Ako už bolo povedané, podmienkou pre existenciu echo stavov sú hodnoty váh v dynamickom rezervoári – prvky matice $\vec{W}$. Ak je norma tejto matice menšia ako 1, odozva dynamického rezervoára môže byť použitá v ESN sieti. Základná úvaha pre existenciu echo stavov je znázornená v rovnici (14). Stav siete v nasledujúcom časovom kroku $\vec{x}(n+1)$ je určený nasledujúcim vstupom $\vec{u}(n+1)$, stavom siete v aktuálnom kroku $\vec{x}(n)$ a aktuálnym požadovaným výstupom $\vec{d}(n)$. Avšak aj aktuálny stav $\vec{x}(n)$ je možné vyjadriť ako funkciu vstupu $\vec{u}(n)$, stavu $\vec{x}(n-1)$ a požadovaného výstupu $\vec{d}(n-1)$.

$$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}(n) + \vec{W}^{back}\vec{d}(n)) \tag{14}$$

kde

$$\vec{x}(n) = \vec{f}(\vec{W}^{in}\vec{u}(n) + \vec{W}\vec{x}(n-1) + \vec{W}^{back}\vec{d}(n-1))$$

kde

$$\vec{x}(n-1) = \vec{f}(\vec{W}^{in}\vec{u}(n-1) + \vec{W}\vec{x}(n-2) + \vec{W}^{back}\vec{d}(n-2))$$

atď.

Pri tomto rozvoji je názornejšia úloha matice $\vec{W}$ v dynamike siete. Ak je norma matice $\vec{W}$ menšia ako 1, tak sieť má tendenciu postupne „zabúdať“ informáciu z predchádzajúcich časových krokov a „najväčší“ vplyv na stav má aktuálny vstup. Ak je teda sieti prezentovaná najprv rozdielna a následne od určitého času identická postupnosť, sieť postupne zabudne na počiatočné rozdiely a s narastajúcim časom bude odozva dynamického rezervoára pre obe postupnosti rovnaká. Inými slovami, sieť má v sebe zahrnutú pamäť s exponenciálnym poklesom informácie [5].

5 Trénovanie Echo State neurónových sietí

Ako už bolo zmieňované, pri trénovaní Echo State neurónových sietí dochádza iba k adaptácii výstupných váh. Získanie Echo State neurónovej siete s požadovanými vstupno-výstupnými vlastnosťami je ale dvojstupňová úloha. Najprv si musíme zabezpečiť RC ANN so schopnosťou mať echo stavy. V druhej časti je našou úlohou vypočítanie výstupných synoptických váh. Teraz si popíšeme základnú myšlienku tejto stratégie.

V tejto kapitole budeme pracovať so systémom, ktorý je budený iba vstupným signálom. Prevezmeme si štandardný pohľad z teórie systémov a predstavíme si dynamický systém (deterministický a v diskrétnom čase) ako funkciu $\vec{G}$, ktorá produkuje výstup systému na základe histórie jeho vstupov.

$$\vec{y}(n+1) = \vec{G}(...,\vec{u}(n),\vec{u}(n+1)). \tag{15}$$

Pri bližšom pohľade na rovnicu (15) zistíme, že je určitým spôsobom obmedzujúca. Nepripúšťa tú možnosť, že výstup systému $\vec{y}(n+1)$ môže závisieť aj na predchádzajúcich výstupoch $\vec{y}(n),\vec{y}(n-1),....$ Echo State neurónové siete dokážu pracovať aj s autoregresívnymi systémami, ale v tejto časti sa obmedzíme iba na čisto neautoregresívny prípad.

Teraz predpokladajme, že máme Echo State neurónovú sieť. Ďalej si predstavme, že sieti bola predkladaná dlhá vstupná sekvencia. Vzhľadom na to, že naša RC ANN má vlastnosť echo stavov a nachádza sa už v stave po počiatočnom prechodnom procese, môžu byť stavy vnútorných neurónov zapísané nasledujúcou rovnicou (s určitou voľnosťou k notácii):

$$x_i(n) \approx e_i(...,\vec{u}(n),\vec{u}(n+1)), \tag{16}$$

kde e_i je echo funkcia i-teho neurónu. Ak je sieť nehomogénna, rôzne echo funkcie sa budú výrazne navzájom odlišovať, a tak budú vytvárať potrebnú dynamiku neurónovej siete.

Naším cieľom je teda v zmysle rovnice (15) aproximovať systém $\vec{y}_{teach}(n+1) = \vec{G}_{teach}(...,\vec{u}(n),\vec{u}(n+1))$ pomocou výstupu neurónovej siete. Základnou myšlienkou je aproximácia $\vec{G}_{teach}$ pomocou kombinácie jednotlivých echo funkcií $\vec{E} = (e_1,...,e_N)$, v zmysle minimalizácie strednej kvadratickej chyby. Výstup neurónovej siete vypočítame na základe vzťahu

$$\vec{y}(n+1) = \vec{f}^{\,out}(\vec{W}^{out}(\vec{u}(n+1),\vec{x}(n+1))). \tag{17}$$

Ako aktivačná funkcia výstupných neurónov je v tomto prípade použitá funkcia tanh (hyperbolický tangens). Keďže táto funkcia je invertovateľná, môžeme prepísať rovnicu (17) nasledujúcim spôsobom

$$(\vec{f}^{\,out})^{-1}(\vec{y}(n+1)) = \vec{W}^{out}(\vec{u}(n+1),\vec{x}(n+1)). \tag{18}$$

Vložením echo funkcií $\vec{E}$ vytvoríme nový zápis

$$(\vec{f}^{\,out})^{-1}(\vec{y}(n+1)) = \vec{W}^{out}(\vec{u}(n+1),\vec{E}(...,\vec{u}(n-1),\vec{u}(n),\vec{u}(n+1))). \tag{19}$$

Rozdiel medzi skutočným a požadovaným výstupom (invertovaným vzhľadom na aktivačnú funkciu výstupných neurónov) si označíme $\varepsilon_{train}(n)$.

$$\begin{aligned}\varepsilon_{train}(n) &= (\vec{f}^{\,out})^{-1}(\vec{y}_{teach}(n)) - (\vec{f}^{\,out})^{-1}(\vec{y}(n)) \\ &= (\vec{f}^{\,out})^{-1}(\vec{y}_{teach}(n)) - \vec{W}^{out}(\vec{u}(n),\vec{E}(...,\vec{u}(n-1),\vec{u}(n))).\end{aligned} \tag{20}$$

Posledným krokom je vypočítanie váhových koeficientov nachádzajúcich sa v matici $\vec{W}^{out}$ takým spôsobom, aby sme minimalizovali strednú kvadratickú chybu (MSE)

$$MSE_{train} = \frac{1}{(n_{\max} - n_{\min} + 1)} \sum_{n=n_{\min}}^{n_{\max}} \varepsilon_{train}^2(n). \tag{21}$$

MSE_{train} je chyba na trénovacej množine, $n_{\min}$ je index trénovacej vzorky, pri ktorej už došlo k „vymytiu“ vplyvu počiatočnej inicializácie a $n_{\max}$ je index poslednej trénovacej vzorky.

Túto sekciu si uzavrieme krátkym zhrnutím všeobecnej verzie trénovacieho procesu.

- Úloha

  Dané: trénovacia vstupno-výstupná sekvencia $(\vec{u}_{teach}(n), \vec{y}_{teach}(n))_{n=0,...,n_{\max}}$.

  Požadované: RC ANN, ktorej výstup $\vec{y}(n)$ aproximuje $\vec{y}_{teach}(n)$.

- Vytvorenie RC ANN s echo stavmi

  Najprv skonštruujeme neurónovú sieť s echo stavmi a následne zvolíme takú veľkosť dynamického rezervoára, aby odrážala potreby či už komplexnosti úlohy alebo požadovanej presnosti.

- Vzorkovacia fáza

  Začneme z ľubovoľného počiatočného stavu neurónovej siete $\vec{x}(0)$ a vypočítame nový stav siete pre každú vstupnú vzorku $n = 0,..., n_{\max}$ na základe vzorca:

  $$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}_{teach}(n+1) + \vec{W}\vec{x}(n)) \tag{22}$$

  Následne je dôležité zvoliť si vhodný časový okamih $n_{\min}$, po ktorom predpokladáme, že už došlo k vymytiu vplyvu počiatočnej inicializácie siete. Po tomto okamihu je už stav neurónovej siete určovaný iba predchádzajúcimi stavmi a vstupmi neurónovej siete.

- Vypočítanie výstupných váh

  Nech $\vec{y}_{teach}(n) = (y_{teach,1}(n),..., y_{teach,L}(n))$ a $\vec{G}'_{teach}(n) = (\vec{f}^{out})^{-1}(\vec{y}_{teach}(n))$.
  Výstupné váhy $\vec{W}^{out}$ vypočítame takým spôsobom, aby sme minimalizovali MSE:

  $$\frac{\sum_{n=n_{\min}}^{n_{\max}} (\vec{G}'_{teach}(n) - \vec{W}^{out}(\vec{u}_{teach}(n), \vec{E}(..., \vec{u}_{teach}(n-1), \vec{u}_{teach}(n)))^2}{(n_{\max} - n_{\min} + 1)}. \tag{23}$$

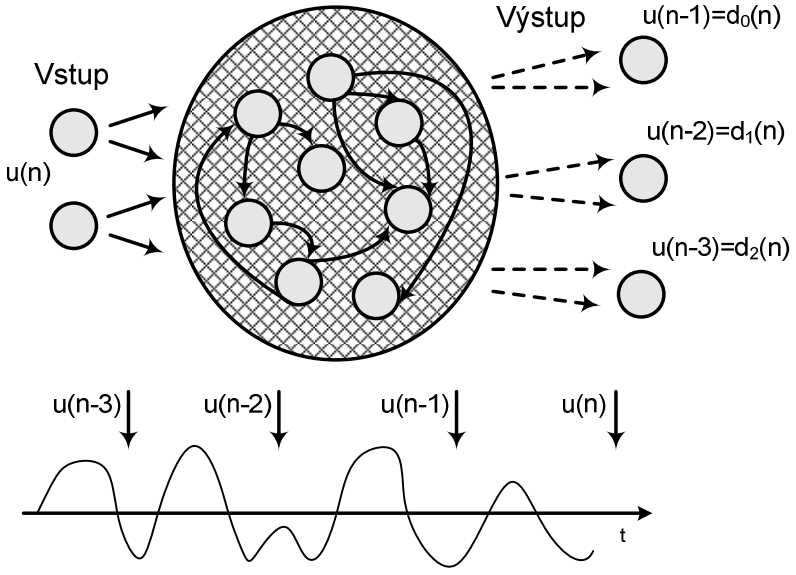
  K vypočítaniu týchto váh môžeme použiť ľubovoľný algoritmus lineárnej regresie. S takto vypočítanými váhami je naša neurónová sieť pripravená na použitie.

6 Krátkodobá pamäť v ESN

Pod pojmom krátkodobá pamäť (STM – Short Term Memory) budeme rozumieť vlastnosť nejakého vstupno-výstupného systému, kde aktuálny výstup $\vec{y}(n)$ závisí na predchádzajúcich hodnotách vstupu $\vec{u}(n-k)$ a/alebo na predchádzajúcich hodnotách výstupu $\vec{y}(n-k)$. Táto vlastnosť úzko súvisí s prechodnými dynamickými procesmi v neurónovej sieti. Ako náprotivok ku krátkodobej pamäti existuje dlhodobá pamäť (LTM – Long Term Memory), za ktorú zodpovedá zmena synaptických váh počas učiaceho procesu neurónovej siete.

Klasickým prístupom pri potrebe krátkodobej pamäte (hlavne v dopredných neurónových sieťach) je vytvorenie takzvaného časového okna na vstupe neurónovej siete. V tomto prípade nebudeme na vstup neurónovej siete privádzať iba sekvenciu v aktuálnom časovom kroku $\vec{u}(n)$, ale aj jej predchádzajúce hodnoty $\vec{u}(n),\vec{u}(n-1),...,\vec{u}(n-k)$. Na rozdiel od tohto prístupu rekurentné neurónové siete ponúkajú alternatívne riešenie potrieb krátkodobej pamäti. Z dôvodu existencie rekurentných prepojení aktuálny stav neurónovej siete $\vec{x}(n)$ obsahuje aj informáciu o predchádzajúcich vstupoch. Z tohto dôvodu, nie je nutné vytvárať časové okno na vstupe neurónovej siete.

Lepšie poznatky o krátkodobej pamäti v Echo State neurónových sieťach získame, ak budeme trénovať Echo State neurónovú sieť na úlohe zameranej priamo na krátkodobú pamäť (obr. 6). Predstavme si Echo State neurónovú sieť s dvoma vstupnými neurónmi a mnohými výstupnými neurónmi. Na vstup $u(n)$ privádzame iba biely šum a na výstupné neuróny privádzame jeho časovo oneskorené hodnoty. Naším cieľom je zistiť, ako dlho a s akou presnosťou je Echo State neurónová sieť schopná zapamätávať si predchádzajúce vstupy.



Obr. 6. Echo State neurónová sieť trénovaná ako prediktor histórie vlastných vstupov.

Najprv si ale musíme uviesť presnú definíciu intuitívneho chápania takzvanej „úspešnosti“ neurónovej siete na úlohách s časovým oneskorením o k krokov. Nech $r(u(n-k), y_k(n))$ je korelačný koeficient medzi skutočným oneskoreným signálom $u(n-k)$ a hodnotou $y_k(n)$ z výstupného neurónu, ktorý sme trénovali na oneskorenie k. Tento koeficient sa pohybuje v rozpätí hodnôt -1 a 1. Jeho umocnením získame veličinu nazývanú v štatistike determinačný koeficient $r^2(u(n-k), y_k(n))$. Hodnoty tohto koeficientu sa už pohybujú v rozpätí 0 a 1. Hodnota 1 indikuje perfektnú koreláciu medzi skutočným signálom a výstupom neurónovej siete a hodnota 0 indikuje totálnu stratu korelácie. Perfektné obnovenie signálu oneskoreného o k krokov teda bude znamenať, že $r^2(u(n-k), y_k(n)) = 1$, kompletná strata zase $r^2(u(n-k), y_k(n)) = 0$.

Teraz si zadefinujme kapacitu pamäte (MC – Memory Capacity) neurónovej siete ako sumu všetkých týchto koeficientov cez všetky oneskorenia na výstupe neurónovej siete:

$$MC = \sum_{k=1}^{\infty} r^2(u(n-k), y_k(n)). \tag{24}$$

Keďže už máme zadefinovanú MC, môžeme uviesť najpodstatnejšie závery týkajúce sa kapacity pamäte Echo State neurónových sietí.

> **Veta 2.** V Echo State neurónovej sieti, ktorej dynamický rezervoár sa skladá z N neurónov, kapacita pamäte MC je menšia, nanajvýš rovná N $(MC \leq N)$.

To znamená, že maximálna možná kapacita pamäte ESN je ohraničená veľkosťou dynamického rezervoára.

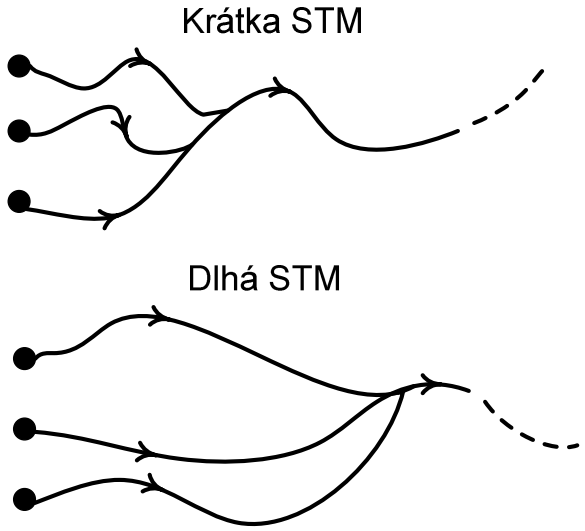
> **Veta 3.** V lineárnej Echo State neurónovej sieti, ktorej dynamický rezervoár sa skladá z N neurónov, kapacita pamäte MC je rovná N $(MC = N)$.

To znamená, že pomocou lineárnej Echo State neurónovej siete vieme dosiahnuť maximálnu možnú kapacitu pamäte. Za lineárnu neurónovú sieť považujeme takú sieť, ktorej vnútorné neuróny majú lineárnu aktivačnú funkciu, napríklad funkciu identity.

> **Veta 4.** V lineárnej Echo State neurónovej sieti nemôžu byť dlhé oneskorenia nikdy naučené lepšie ako krátke oneskorenia (monotónne zabúdanie).

Ak potrebujeme Echo State neurónovú sieť s čo možno najdlhšou krátkodobou pamäťou (obr. 7), môžeme v praktických úlohách vyskúšať niektorý z nasledujúcich krokov (alebo ich kombináciu):

- Použijeme čo možno najväčší dynamický rezervoár. Toto je najefektívnejšia a všeobecne aplikovateľná metóda, ale vyžaduje dostatočne veľkú trénovaciu množinu.
- Použijeme malé hodnoty vstupných váh. Tým zabezpečíme, že neurónová sieť bude pracovať takmer v lineárnom rozsahu. Tu sa ale môžeme dostať do konfliktu s nelineárnou charakteristikou úlohy.
- Použijeme lineárne aktivačné funkcie neurónov v dynamickom rezervoári. Opäť nám hrozí ten istý problém ako v predchádzajúcom bode.
- Použijeme špeciálne pripravený dynamický rezervoár s takmer jednotkovou váhovou maticou (bližšie informácie v [3]).
- Použijeme spektrálny polomer matice blízky hodnote 1. Takto vytvorená Echo State neurónová sieť bude fungovať úspešne iba pri úlohách s pomalou dynamikou.



Obr. 7. Porovnanie takzvanej krátkej a dlhej krátkodobej pamäte.

7 Podrobný popis algoritmu

V tejto kapitole bude podaný podrobný a kompletný popis trénovania ESN na rozdiel od všeobecnej a zjednodušenej verzie uvedenej v kapitole 5. V tomto popise budeme predpokladať, že výstupné neuróny majú sigmoidálnu prechodovú funkciu a že existujú spätné spojenia z výstupných neurónov do dynamického rezervoára. Tým pádom sme zobrali do úvahy tú najvšeobecnejšiu verziu algoritmu. V mnohých prípadoch sa používajú jednoduchšie verzie. Napríklad výstupné neuróny nie sú sigmoidálne ale lineárne, nemusia existovať spätné spojenia z výstupných neurónov do dynamického rezervoára, alebo dokonca môžeme mať aj neurónovú sieť bez vstupných neurónov. V takýchto prípadoch nie je problém upraviť všeobecnú verziu učiaceho algoritmu tak, aby vyhovoval použitej architektúre.

- *Dané*: Trénovacia vstupno/výstupná sekvencia $(\vec{u}(1),\vec{d}(1)),...,(\vec{u}(T),\vec{d}(T))$.
- *Požadované:* Natrénovaná ESN $(\vec{W}^{in},\vec{W},\vec{W}^{out},\vec{W}^{back})$. Od tejto natrénovanej siete vyžadujeme, aby v testovacej fáze, keď je jej predkladaný už len vstup $\vec{u}(n)$, správne aproximovala svojím výstupom $\vec{y}(n)$ požadovaný výstup $\vec{d}(n)$.

  1. V skutočnosti môžeme očakávať od ESN správnu aproximáciu požadovaného výstupu, až keď dôjde k „vymytiu“ počiatočnej prechodnej dynamiky siete. Tá je spôsobená tým, že na začiatku trénovacieho procesu sú stavy ESN náhodne (ľubovoľne) zvolené. Z tohto dôvodu vlastne požadujeme natrénovanú ESN, ktorá správne aproximuje požadovaný výstup v časových krokoch $n = T_0,...,T$, kde $T_0 > 1$. Hodnota parametra T_0 závisí na veľkosti dynamického rezervoára Echo State neurónovej siete. Pohybuje sa od 10 (pre malé siete) až po 500 (pre veľké siete).
  2. Čo vlastne prioritne chceme, nie je dobrá aproximácia učiacich dát, ale oveľa dôležitejšia je dobrá aproximácia testovacích dát. Tieto dáta sú samozrejme vygenerované tým istým systémom ako údaje použité na trénovanie neurónovej siete.

- *Krok 1.* V tomto kroku získame nenatrénovanú ESN $(\vec{W}^{in},\vec{W},\vec{W}^{back})$, ale už so schopnosťou mať echo stavy. Ak v tejto fáze privedieme na vstup nejakú sekvenciu, tak môžeme pozorovať veľmi zaujímavé dynamické správanie sa neurónov v DR.

  Tento krok zahrňuje veľa heuristík. Ukážeme si najčastejšie používaný postup [2].

  1. Náhodne vygenerujeme maticu vnútorných váh $\vec{W}_0$ (matica váh DR).
  2. Vypočítame spektrálny polomer matice $\vec{W}_0$, ktorý označíme $|\lambda_{\max}|$. Následne normalizujeme maticu $\vec{W}_0$ pomocou $|\lambda_{\max}|$, čím získame novú maticu $\vec{W}_1$, $\vec{W}_1 = \frac{1}{|\lambda_{\max}|}\vec{W}_0$.
  3. V tomto kroku vynásobíme maticu $\vec{W}_1$ parametrom α, čím získame novú maticu $\vec{W}$ so spektrálnym polomerom α, pričom $\alpha < 1$.
  4. Náhodne vygenerujeme maticu vstupných váh $\vec{W}^{in}$ a maticu spätných váh $\vec{W}^{back}$. Po tomto kroku môžeme skonštatovať, že naša neurónová sieť $(\vec{W}^{in},\vec{W},\vec{W}^{back})$, nezávisle na tom ako sú zvolené $\vec{W}^{in},\vec{W}^{back}$, je Echo State neurónovou sieťou.

Poznámky:

1. Našou snahou je zabezpečiť bohatú rozmanitosť dynamiky skrytých neurónov v DR. Z tohto dôvodu by mala matica $\vec{W}_0$ obsahovať dostatočné množstvo nulových (alebo nule blízkych) hodnôt – vytvoríme takzvanú „riedku“ maticu. Okrem toho, váhy by mali byť zhruba vyvážené, t.j. priemerná hodnota by sa mala pohybovať okolo nuly. Jednou z možností je vygenerovanie nenulových váh z uniformného rozdelenia a rozsahu [-1, 1].

2. Veľkosť N matice $\vec{W}_0$ by mala odrážať aj dĺžku T trénovacích dát a zároveň aj zložitosť úlohy. V praxi bolo zistené, že N by sa malo držať v rozmedzí od $T/10$ do $T/2$ v závislosti na zložitosti dát (čím pravidelnejšie dáta, tým bližšie k $T/10$ môže byť zvolená hodnota N). Týmto jednoduchým opatrením zabránime preučeniu neurónovej siete. Samozrejme, čím zložitejšie dáta, tým je potrebný väčší dynamický rezervoár.

3. Voľba parametra α je rozhodujúca pre celkovú funkčnosť ESN. Prax je taká, že tento parameter je potrebné doladiť pre danú úlohu vyskúšaním rôznych hodnôt.

4. Dôležitú úlohu zohráva aj veľkosť vstupných váh $\vec{W}^{in}$. Veľké hodnoty váh matice $\vec{W}^{in}$ spôsobujú, že neurónová sieť je silne riadená vstupom. Naopak malé hodnoty spôsobujú, že stavy jednotlivých neurónov sú len jemne vybudené okolo kľudového (nulového) stavu DR. Postupom času začnú neuróny operovať okolo lineárnej – centrálnej časti sigmoidy. Čiže získame sieť s takmer lineárnou dynamikou. Veľké hodnoty váh v matici $\vec{W}^{in}$ vedú neuróny bližšie k saturačnej oblasti sigmoidy, čo vedie k nelineárnejšiemu správaniu výsledného modelu. V extrémnych prípadoch by mohlo dôjsť až k takzvanej binárnej dynamike siete. V tomto prípade by jednotlivé neuróny produkovali len hodnoty $-1/+1$.

5. Obdobné pravidlá platia aj pre hodnoty váh v matici $\vec{W}^{back}$.

- *Krok 2.* V tomto kroku vypočítame maticu $\vec{M}$ (matica vnútorných stavov neurónovej siete) a maticu $\vec{T}$ (matica požadovaných výstupov neurónovej siete) – je to takzvaná vzorkovacia fáza.

  Tento krok je len mechanický a nezahŕňa žiadnu heuristiku:

  1. Náhodne nainicializujeme počiatočné stavy neurónovej siete, napríklad nulové stavy $\vec{x}(0) = 0$.

  2. Následne začneme sieti predkladať jednotlivé kombinácie vstupov $\vec{u}(n)$ a k nim odpovedajúcich požadovaných výstupov $\vec{d}(n-1)$. To opakujeme pre

všetky učiace vzorky, $n = 0,...,T$. Počas toho, čo je sieť budená jednotlivými vstupno/výstupnými sekvenciami, vypočítavame stavy na jednotlivých skrytých neurónoch:

$$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}(n) + \vec{W}^{back}\vec{d}(n)) \tag{25}$$

3. V čase $n = 0$, kde $\vec{d}(n)$ nie je definovaná, použijeme $\vec{d}(n) = 0$.

4. Teraz si musíme zvoliť hodnotu časového kroku T_0, po ktorom predpokladáme, že dôjde už k „vymytiu“ vplyvu počiatočnej inicializácie neurónovej siete. Od tohto časového kroku T_0 ukladáme každý stav neurónovej siete $\vec{x}(n)$ v ďalšom kroku ako nový riadok v stavovej matici $\vec{M}$. Na konci tohto procesu získame maticu vnútorných stavov neurónovej siete s rozmermi $(T - T_0 + 1) \times (K + N + L)$.

5. Podobným spôsobom, pre čas väčší alebo rovný T_0, zozbierame do matice $\vec{Z}$ sigmoidálne-invertované učiace výstupy $\tanh^{-1}\vec{d}(n)$. Tým dostaneme maticu $\vec{Z}$ o rozmeroch $(T - T_0 + 1) \times L$.

Poznámka: Tu si treba dať pozor, aby sme do matíc $\vec{M}$ a $\vec{Z}$ ukladali vektory $\vec{x}(n)$ a $\tanh^{-1}\vec{d}(n)$ a nie $\vec{x}(n)$ a $\tanh^{-1}\vec{d}(n-1)$!

- *Krok 3*. Vypočítanie výstupnej váhovej matice.

1. Najprv vypočítame pseudoinverznú maticu $\vec{M}^{-1}$ z matice $\vec{M}$.

2. Následne vynásobíme maticu $\vec{M}^{-1}$ a maticu $\vec{Z}$. Tým získame maticu $(\vec{W}^{out})^t$ o rozmeroch $(K + N + L) \times L$, ktorej i-ty stĺpec obsahuje výstupné váhy zo všetkých skrytých neurónov do i-teho výstupného neurónu:

$$(\vec{W}^{out})^t = \vec{M}^{-1}\vec{Z}. \tag{26}$$

3. Stačí už len transponovať maticu $(\vec{W}^{out})^t$ na maticu $\vec{W}^{out}$ a získame nami požadovanú maticu výstupných váh.

- *Krok 4.* Fáza použitia neurónovej siete.

Echo State neurónová sieť $(\vec{W}^{in}, \vec{W}, \vec{W}^{out}, \vec{W}^{back})$ je v tejto chvíli natrénovaná a pripravená na použitie. V tejto fáze sú sieti predkladané už len vstupné sekvencie a jej úlohou je čo najlepšie aproximovať požadovaný výstup.

$$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}(n) + \vec{W}^{back}\vec{y}(n)), \quad (27)$$

$$\vec{y}(n+1) = \vec{f}^{out}(\vec{W}^{out}(\vec{u}(n+1), \vec{x}(n+1), \vec{y}(n))), \quad (28)$$

kde $\vec{f}^{out}$ je aktivačná funkcia na výstupných neurónoch.

Možné varianty:

1. Pri niektorých dátach môže dochádzať k problémom so stabilitou vo fáze použitia natrénovanej neurónovej siete. V takýchto prípadoch veľmi často pomôže pridanie šumu vo vzorkovacej fáze a následne upravenie vzťahu (25):

$$\vec{x}(n+1) = \vec{f}(\vec{W}^{in}\vec{u}(n+1) + \vec{W}\vec{x}(n) + \vec{W}^{back}\vec{d}(n) + v(n)), \quad (29)$$

   kde $v(n)$ je biely uniformný šum (typický rozsah od 0,0001 do 0,001). Bližšie informácie sú v [2].

2. V niektorých prípadoch nemusíme použiť synaptické spojenie smerujúce z výstupných neurónov naspäť na výstupné. V takýchto prípadoch nám vznikne matica výstupných synaptických váh $\vec{W}^{out}$ o rozmeroch $(K+N)\times L$ a namiesto rovnice (28) použijeme nasledujúci zápis:

$$\vec{y}(n+1) = \vec{f}^{out}(\vec{W}^{out}(\vec{u}(n+1), \vec{x}(n+1))). \quad (30)$$

3. Niekedy môže byť modelovaný systém vysoko nelineárny. Pre jeho spracovanie sa dobré výsledky dosiahli použitím takzvaných „rozšírených“ stavov neurónov vo vzorkovacej fáze a následne aj vo fáze použitia. To znamená, že ako doplnok k originálnemu stavu $\vec{x}(n)$ je ešte pridaná určitá nelineárna transformácia tohto stavu. Najjednoduchšou transformáciou je druhá mocnina pôvodného stavu. Počas vzorkovacej fázy neukladáme do matice $\vec{M}$ len stavy $\vec{x}(n) = (x_1(n),...,x_{K+N+L}(n))$, ale aj ich rozšírené verzie $(\vec{x}(n), \vec{x}^2(n)) = (x_1(n),...,x_{K+N+L}(n), x_1^2(n),...,x_{K+N+L}^2(n))$. Tým pádom nám vznikne matica výstupných váh $\vec{W}^{out}$ o veľkosti $L\times 2(K+N+L)$ a namiesto vzorca (28) použijeme jeho upravenú verziu:

$$\vec{y}(n+1) = \vec{f}^{out}(\vec{W}^{out}(\vec{u}(n+1), \vec{x}(n+1), \vec{y}(n), \vec{u}^2(n+1), \vec{x}^2(n+1), \vec{y}^2(n))). \quad (31)$$

8 Pokročilá verzia učiaceho algoritmu

Túto modifikáciu učenia navrhli H. Jaeger a H. Haas [4] pre Echo State neurónovú sieť so spätnou väzbou (s váhami smerujúcimi z výstupných neurónov späť do dynamického rezervoára $\vec{W}^{back}$. Jej podstata spočíva v znovu využití výstupného signálu z dynamického rezervoára pre opätovné preučenie siete.

Keď natrénovaná sieť v procese „života" autonómne generuje výstupný signál, prijíma svoj vlastný výstupný signál cez spätnoväzobné váhy. Tento signál sa líši od pôvodne použitého v procese vzorkovania, čo spôsobí, že stavy v dynamickom rezervoári budú odlišné ako v procese učenia. Ak by sa počas učenia siete privádzal na výstupné neuróny signál podobný tomu, ktorý bude vystupovať z dynamického rezervoára v procese života siete, presnosť modelu sa zvýši. Túto myšlienku možno zahrnúť do procesu učenia siete v nasledujúcich etapách:

1. Echo State neurónovú sieť najprv naučíme klasickou verziou učiaceho algoritmu, čím dostaneme predbežné výstupné váhy, ktoré si označíme ako $\vec{W}_{out}^{0}$.
2. Z pôvodných vzoriek použitých na učenie siete a predbežných výstupných váh $\vec{W}_{out}^{0}$ vygenerujeme novú sériu trénovacích vzoriek $\vec{d}'(n)$ pre $n>1$, a to nasledovne:
   - položíme $\vec{d}'(1)=\vec{d}(1)$ a odštartujeme sieť z náhodného stavu $\vec{x}(1)$. Na výstup siete privedieme $\vec{d}(1)$ a vypočítame $\vec{x}(2)$ a $\vec{y}(2)$ pomocou týchto vzťahov:

$$\vec{x}(2)=\tanh(\vec{W}\vec{x}(1)+\vec{W}_{back}\vec{d}(1)), \tag{32}$$

$$\vec{y}(2)=\tanh(\vec{W}_{out}^{0}(\vec{x}(2))), \tag{33}$$

     a položíme $\vec{d}'(2)=\vec{y}(2)$.

   - pre ďalšie vzorky $\vec{d}'(3),...,\vec{d}'(n)$ použijeme nasledovné zovšeobecnené vzorce:

$$\vec{x}(n)=\tanh(\vec{W}\vec{x}(n-1)+\vec{W}_{back}\vec{d}(n-1)), \tag{34}$$

$$\vec{y}(n)=\tanh(\vec{W}_{out}^{0}(\vec{x}(n))), \tag{35}$$

     a položíme $\vec{d}'(n)=\vec{y}(n)$.
3. Tento proces opakujeme n-krát, čím dostaneme novú sériu vzoriek, ktorá predstavuje jednokrokovú predikciu z pôvodne natrénovanej siete.

4. Novo získané vzorky $\vec{d}'(n)$ použijeme ako nové vzorky na opätovné preučenie siete, čím dostaneme konečné výstupné váhy $\vec{W}_{out}$.

Bližšie informácie k tejto metóde, experimenty ako aj dosiahnuté výsledky možno nájsť v [4].

9 Záver

Echo State neurónové siete patria k relatívne novým prístupom v oblasti neurónových sietí. Ich výhodou je väčšia príbuznosť k biologickým modelom kvôli ich rekurentnému charakteru a využitiu rezervoáru dynamického správania sa bez nastavovania váh. V tomto dynamickom rezervoári sa vytvára informačne „bohatšia“ odozva vstupného signálu, ktorá je použitá na produkovanie požadovaného výstupu. Z hľadiska využitia sú vhodné na naučenie sa časovej rady, čo je nevyhnutný predpoklad napríklad pre riadenie motoriky, nech už u človeka, alebo u robota, alebo napr. na spracovanie reči. Oproti ostatným typom rekurentných sietí majú Echo State siete podstatnú výhodu v schopnosti učiť sa "jednokrokovo", keďže adaptácia výstupnej vrstvy je realizovaná metódou lineárnej regresie.

Poďakovanie: Táto kapitola vznikla za podpory grantovej agentúry VEGA SR v rámci grantových úloh 1/0804/08 a 1/0141/10.

Literatúra

[1] Goldenholz, D.: *Liquid Computing: A Real Effect*. Technical Report, Boston University Department of Biomedical Engineering, 2002.

[2] Jaeger, H.: *The „Echo State“ Approach to Analysing and Training Recurrent Neural Networks.* German National Research Center for Information Technology, GMD report 148, 2001.

[3] Jaeger, H.: *Short Term Memory in Echo State Networks.* German National Research Center for Information Technology, GMD report 152, 2002.

[4] Jaeger, H., Haas, H.: *Harnessing Noonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Telecomunication*. Science, 304, 78-80, 2004.

[5] Makula, M., Čerňanský, M.: *Neurónové siete založené na echo stavoch*. Kognice a umělý živo IV, Opava, pp. 363-371, 2004.

[6] Natschlager, T., Maass, W., Markram, H.: *The Liquid Computer: A Novel Strategy for Real-time Computing on Time Series.* Special Issue on Foundations of Information Processing of TELEMATIK, 8(1), 39-43, 2002.

Virtuální agenti

Cyril BROM, Michal BÍDA, Martin KLÍMA, Jakub GEMROT, Radek PÍBIL, Tomáš PLCH, Rudolf KADLEC [1]

Abstrakt. V posledných štyridsiatich rokoch došlo k nebývalému rozvoju číslicových informačných, komunikačných a riadiacich technológií a systémov, ktoré priam revolučným spôsobom menia a ovplyvňujú kvalitu a spôsob života na našej planéte. Celosvetové a celoeurópske inovatívne trendy v oblasti komplexnej informatizácie procesov a procesných činností smerujú k potrebe výskumu, vývoja a implementácie nových systémových riešení a zmien, ktoré si vyžadujú nové inteligentné, vysoko autonómne vnorené systémy riadenia, diagnostiky a komunikácie. Cieľom príspevku je návrh metodológie, metód a algoritmov inteligentnej integrovanej analýzy a syntézy informačných, komunikačných a riadiacich systémov pre nové evolučné štruktúry riadenia využívajúce vnorenosť, autonómnosť a inteligenciu.

1 Byl jednou jeden virtuální agent...

Snad odnepaměti fascinuje lidi představa, že by mohli vytvořit bytosti podobné jim samým, obdařené inteligencí. Kdo by neznal golema, Frankensteina a roboty? Některé z těchto bytostí přestávají existovat jen v našich představách a jejich zjednodušeniny začínají ožívat pod rukama vědců. Zřejmě posledním přírůstkem do rodiny těchto bytostí jsou *virtuální agenti* ("virtual agents"). Jde o počítačové programy, které imitují chování inteligentních bytostí ve fiktivním světě modelovaném na počítači pomocí virtuální reality. Často jsou imitovanými bytostmi lidé, a proto se virtuálním agentům někdy také říká *virtuální lidé* ("virtual humans"). Není to ale označení přesné – opomíjí virtuální ne-lidi; zvířata, fantaskní monstra a – ano – i virtuální roboty.

Kdo nikdy neviděl virtuální agenty v akci, může pochybovat o jejich schopnostech – *ono* to opravdu něco umí? Kdo je někdy viděl krátce – například na popularizační přednášce nějakého nadšence – možná se začal bát – neovládnou nás? Kdo čte hodně sci-fi, věří futurologickým předpovědím kdejakých šarlatánů a už delší dobu hraje počítačové hry, diví se – ono to pořád *ne*funguje?

Existuje ještě čtvrtá kategorie. Byli bychom rádi, aby se do ní zařadil čtenář, který si přečte tuto kapitolu. Aby ohledně schopností virtuálních agentů (dnešních

[1] Kabinet software a výuky informatiky, Matematicko-fyzikální fakulta Univerzity Karlovy v Praze, E-mail: brom@ksvi.mff.cuni.cz

i budoucích) zachovával zdravou skepsi a uvědomoval si, že naprogramovat takového agenta je hodně těžké a že to koneckonců musí udělat člověk. Zdravá skepse je ovšem převlečený opatrný optimismus.

Oslí můstek na závěr úvodu: víte, jak poznáte *dobrého* úředníka? Když za dobrým úředníkem přijdete s problémem, snaží se vám pomoci ho vyřešit. Špatný úředník naproti tomu nalistuje Velké Pravidlo ve své Knize Předpisů a udělá to, co mu Pravidlo řekne. Jde typicky o něco, co ani trochu nesouvisí s tím, proč jste za ním přišli.

Proč o tom mluvíme? Pravda je taková, že dnešní virtuální agenti jsou jako špatní úředníci. Špatní úředníci ještě budou nějakou dobu mezi námi a stejně tak i virtuální agenti budou ještě nějakou dobu listovat ve svých Knihách Předpisů. Pojďme se na to podívat blíže... Začněme ale pěkně po pořádku. I špatný úředník je především bytost z masa a kostí.

2 Virtuální agent je (skoro) z masa a kostí

Každý virtuální agent má *tělo*. Z čeho takové tělo je? Předně si musíme uvědomit, že tato otázka má trochu jiný smysl než tatáž otázka v kontextu živého organismu.

Každý špatný úředník má tělo. Z čeho takové tělo je?

No ovšem, z masa a kostí. Trochu by to mohl kazit biolog, který by mohl namítnout – nene, z buněk. Rýpavý chemik by zase mohl začít vyjmenovávat chemické sloučeniny. Fyzik atomy. Všichni by se ale nakonec shodli, že jejich odpovědi vždy odkazují k jednomu – k matérii, z níž je tělo utkané – k hmotě. Pouze každý z nich popisuje hmotu jinou řečí.

Virtuální agent v tomto smyslu žádné tělo nemá. V jistém – jiném – smyslu je agent z pixelů na monitoru. Pixely ovšem virtuálnímu agentu *nepatří*. Když se virtuální agent pohne, zobrazí ho jiná sada pixelů: pixely jsou mu propůjčovány, zatímco atomy člověka utváří jaksi kontinuálně (pravda, sem tam nějaký atom přibude a jiný ubude, ale to sem teď nepleťme).

V dalším – opět jiném – smyslu je virtuální agent z „nul“ a „jedniček“, které běhají po základní desce počítače, procesoru, paměti, harddisku a tak dále – ale ani tyto „nuly“ a „jedničky“ se nemají k virtuálnímu agentovi tak, jako k člověku atomy.

Přesto – na obrazovce vidíme tělo virtuálního agenta (obr. 1). Dává tedy snad alespoň *nějaký* smysl ptát se, „z čeho to tělo je...“? Ano, dává. Ten smysl je určen těmi, kteří tu otázku přirozeně kladou a odpovídají na ni; grafickými designéry a programátory. Z jejich pohledu je panák na obrazovce z *kostí*, z *trojúhelníků*, z *rigu* a z *textury*. Tyto čtyři pojmy jsou myšlenkové konstrukty, které se ukázaly jako užitečné při rozhovorech o návrhu a tvorbě těla virtuálního agenta. Jsou natolik užitečné, že se jimi pojmenovávají konkrétní obrazce, které se používají jako meziprodukty při návrhu takového virtuálního těla. A také se jimi pojmenovávají konkrétní datové struktury, které při použití jiného měřítka vidíme jako „nuly“ a „jedničky“. Na obrazovce ovšem žádné kosti, trojúhelníky ani rig neuvidíme; když už, tak snad jakousi pokřivenou texturu.

Když stavíme tělo agenta, jako jednu z prvních věcí musíme vytvarovat jeho povrch. Použijete na to *trojúhelníky* (obr. 2), obecně *polygony* (konvexní n-úhelníky; tyto dva termíny budeme dále používat jako synonyma a budeme mít namysli

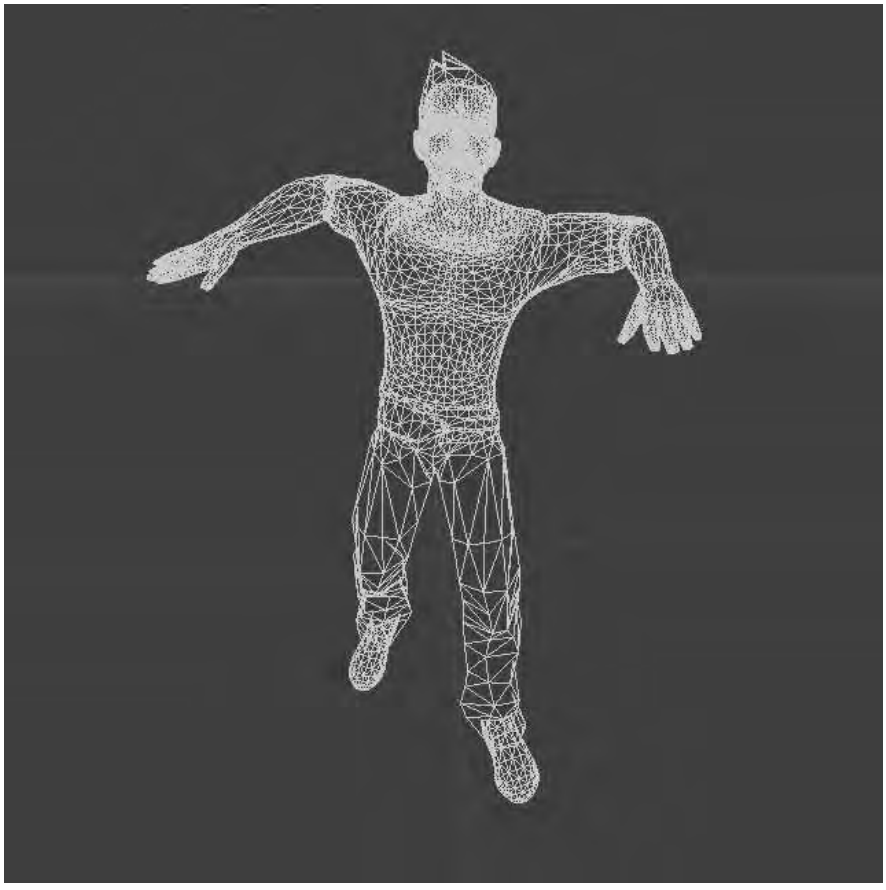
trojúhelníky; pojem *tri*gon se nepoužívá). Čím více jich je, tím je povrh „zaoblenější“. Panáci z dnešních počítačových her mají tělo z tisíců, ba i desetitisíců polygonů. Tělo z polygonů je jako loutka z drátů, proto se mu někdy říká *drátový model* nebo častěji (zvláště v cizině) anglickým slovem *mesh*, které původně znamená „síťovina“.

Dráty musíme obalit „omalovanou gázou“ – *texturou*. Textura určuje barvu kůže a šatů. Tomu, jak je textura na trojúhelníky připevněna, tedy který kousek textury obarví který trojúhelník, se říká *wrap*. Dvě věci jsou důležité. Zaprvé, agent většinou nemá pod šaty kůži: má šaty „přilepené“ rovnou na dráty – jako ta loutka. Zadruhé, „výčnělky“ na těle panáka musíme udělat pomocí polygonů. Textura je „plochá“. Chcete náramek? Přidejte padesát polygonů. Je nicméně pravda, že některé malé „výčnělky“ lze, ba i velmi dobře, „nakamuflovat“ vhodným stínováním textury, popř. aplikací dalších efektů.

Obr. 1. Ukázka virtuálního agenta.

Trojúhelníky a textura tvoří dohromady cosi jako kůži a šaty. Co je „pod“ nimi? A potřebujeme vůbec něco? Potřebujeme, jinak by měl náš agent problém se hýbat. Pokud chcete udělat s panákem krok, znamená to rozpohybovat (v grafickém softwaru) všechny ty tisícovky polygonů. Textura se navíc musí hýbat s nimi – jako ta gáza nalepená na drátovou loutku.

Možností, jak panákem hýbat, je více, ale nejčastěji se používá jeden trik: panákovi se přidělá *kostra* (obr. 3), ta se rozpohybuje a její pohyb se automaticky přenese na vrcholy trojúhelníků. Protože kostra je z *kostí*, jichž jsou desítky, zatímco polygonů je o několik řádů více, vyjde to jednodušeji. Samotná kostra se rozhýbe (většinou) buď ručně nebo pomocí technologie *motion capture* (nepřekládáme do češtiny, české překlady se nepoužívají). Standardní motion capture funguje tak, že na živého herce navěsíme několik referenčních značek, typicky alespoň dvanáct, jejichž pozici v 3D prostoru budeme zaznamenávat pomocí speciálních kamer (často infračervených), a necháme herce provést několik pohybů – například chůzi, rozhlížení se, nadávání apod. Pomocí těchto záznamů potom rozhýbeme kostru.

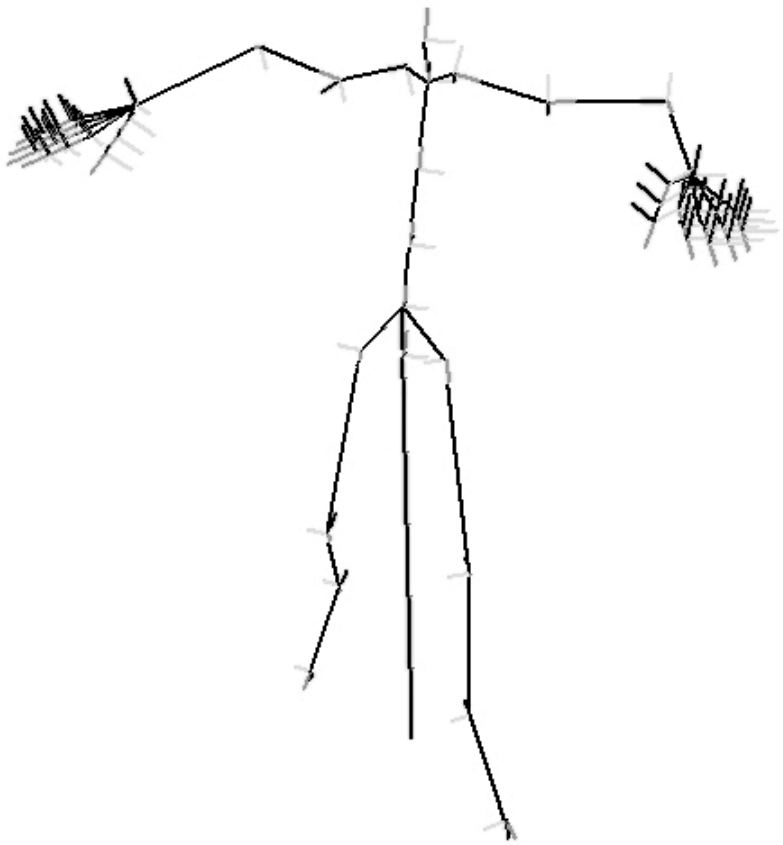


Obr. 2. Drátový model těla virtuálního agenta.

Aby se s kostrou hýbaly i trojúhelníky (a potažmo textura), musíme každému trojúhelníku, přesněji každému jeho vrcholu, přiřadit kost, k níž patří. Vrchol se pak bude hýbat s touto kostí. Zvedneme-li vřetení kost, algoritmus pro nás dopočítá, jak to bude vypadat, když se zvednou vrcholy trojúhelníků na předloktí. A protože s vřetení kostí se zvedne i „dlaňová“ kost, pohnou se i vrcholy dlaní a prstů.

U kloubů je to složitější – trojúhelníkům u kloubů musíme většinou přiřadit obě kosti. To proto, že pokud pohnete například s pažní kostí, měly by se posunout nejen trojúhelníky paže, ale trochu i trojúhelníky, které tvoří rameno a část zad.

Tomuto přiřazení se říká *rig* – zjednodušeně řečeno jde o analogii svalů, pokud budeme svaly chápat pouze jako pojivo mezi kostmi a kůží. Práce je to nad hlavu. Pokud uděláte chybu, při určitých pohybech panáka to může vypadat dost špatně (obr. 4). Ale jak jednou kostru, rig a drátový model máte, můžete vytvořit mnoho variací téže postavy jen výměnou textury.



Obr. 3. Kostra virtuálního agenta. Krátké úsečky u kloubů znázorňují orientaci kloubů**.**

Tělem virtuálního agenta tedy pohybujeme tak, že na kostře „přehráváme“ jednotlivé pohyby, jež jsme získali pomocí technologie motion capture nebo vytvořili ručně, a určitý algoritmus díky rigu přenáší pohyb kostry na drátový model a zase jiný algoritmus na rozpohybované trojúhelníky nanáší texturu. Jednotlivým pohybům se říká *animace*.

Z výše popsaného plynou dvě důležité věci. Zaprvé, aby na sebe animace navazovaly, musí začínat a končit v neutrální pozici, jinak při přechodu mezi dvěma animacemi uvidíme „skok“ (leda bychom přišli s nějakým dalším trikem, jak se „skoku“ vyhnout – viz dále). Zadruhé, pokud si nepřipravíme nějakou animaci předem, máme většinou smůlu. Panák umí sebrat předmět ze stolu, ale zapomněli jsme na animaci sebrání předmětu ze země? Pozdě.

Tedy přesněji řečeno, mnoho lidí z oblasti počítačové grafiky pracuje na tom, aby ty dva body nebyly pravda a někdy se jim to i docela daří. Máme animace chůze a běhu a potřebovali bychom *automaticky* spočítat přechod mezi nimi? Dá se s tím něco udělat. Máme panáka, který nedýchá? (Když panák stojí a *vůbec* se nehýbe, vypadá to velmi nepřirozeně – člověk, který stojí na místě, není latexová figurína a ve skutečnosti se trochu pohybuje.) I s tím se dá něco dělat.

Obr. 4. Špatný „rig“ agenta. Všimněme si, že ramena přiléhají špatně k tělu agenta. V podpaží je tělo jakoby deformované. Toto je důsledek špatného nastavení rigu agenta.

Říká se tomu *procedurální* generování animací; myslí se tím to, že určitý program za nás sám rozpohybuje tělo agenta: dáme mu například na vstup dvě animace, které nekončí v neutrální póze, a on sám dopočítá přechod mezi nimi. Zlí jazykové tvrdí, že lidé vyvíjející počítačové hry ve skutečnosti netouží po lepších algoritmech umělé inteligence, nýbrž po lepších algoritmech pro procedurální generování animací. Chtějí magické tlačítko HUP, které bude jako ubrousek Prostři se: zmáčkneme ho a panáci budou tančit, jak pán poroučí! A desítky grafiků půjdou k vodě.

Nakonec se ještě aspoň stručně vraťme k animacím, které nepoužívají kostru. V některých případech, zejména při animaci obličeje (mimika a řeč) není přístup kostry a rigu vhodný, protože věrohodná animace by vyžadovala více kostí než celý zbytek postavy. Ve hrách se používají různé metody, buď tzv. *morph targety* nebo *deformační klece*. V obou případech je trik v tom, že se model hlavy rozdělí na několik částí (více či méně pečlivě, tím se právě ty metody liší) a tyto části se pak animují pomocí jednoduchých „globálních“ operací jako je roztažení, pootočení a podobně.

2.1 K dalšímu čtení a zamyšlení

V této kapitole blíže popíšeme některé pokročilé technické problémy, které při animování agentů mohou nastat, a způsoby jejich řešení. Podíváme se na to, co všechno se dá spočítat přímo za běhu programu a jaká jsou omezení takovýchto animací. Dále se podíváme na to, jak se dá simulovat např. pád ze schodů a jestli je možné spojit dvě animace do jedné. Čtenář, který spěchá, může tuto kapitolu přeskočit. Čtenář, který hledá souhrnný text o počítačové grafice, se může s důvěrou obrátit na knihu [59]. O animacích obecně se lze dočíst v [45].

Lineární algebra

Počítačová animace (a to nejen ve hrách) je hezkou aplikací základů lineární algebry, jinak jen středně populárního předmětu. Pokud se jí chcete zabývat jinak než jako uživatelé, uděláte dobře, když zbystříte, až se bude mluvit o „homogenních souřadnicích“, „kvaternionech“, o vektorech a maticích ani nemluvě.

Pro animace jsou zvláště důležité *kvaterniony* (jsou to vlastně zobecněná komplexní čísla ne s jednou, ne se dvěma, ale se třemi imaginárními složkami), které umožňují elegantně „sčítat“ rotace, základní stavební blok kosterní animace.

Matematika potřebná pro animace je pěkně popsána například v [3].

Procedurální animace

Uvažme následující příklad: Máme agenta, strážce určitého objektu, a chceme, aby sledoval pohledem kolemjdoucí postavy – tedy aby směrem k nim natáčel hlavu. Takovou animaci si nelze připravit předem, protože nevíme, jak se bude hráč v prostředí pohybovat. Můžeme ale použít tzv. procedurální animace, ty jsou vytvářené přímo za běhu našeho programu. Spočítat totiž, jak má náš agent natočit hlavu, není nijak těžké. Na druhou stranu jsou s tím spojeny určité problémy. Zatím si nijak nedáváme pozor na omezení pohyblivosti krku a očí (agent schopný otočit hlavou

kolem dokola by působil opravdu divně). Tato omezení je opět nutné ručně předem specifikovat. Více se o tomto problému dozvíme níže.

Napadají vás další situace, kde bychom upotřebili procedurální animace? Poznamenejme, že jednoduché procedurální animace, jako je otáčení hlavy, často nestačí. Pro vytvoření opravdu kvalitního dojmu ze scény je třeba je zkombinovat se složitějšími přístupy, z nichž některé jsou vysvětleny dále.

Fyzika

Představme si situaci, ve které chceme, aby náš agent spadl ze schodů. Aby to vypadalo věrohodně, měl by náš agent dynamicky interagovat s prostředím. Měl by se odrážet od zdí, nebo si třeba vykloubit ruku, která se mu zasekla v zábradlí, a podobně. Pokud bychom se drželi pouze standardních animací, tak bychom pravděpodobně měli jenom kácející se postavu, která by ignorovala svoje okolí: mohla by třeba skončit s rukou či nohou ve zdi nebo by jí visely nohy ve vzduchu nad schody.

V takovéto situaci musíme animaci počítat přímo za běhu programu. Co budeme všechno potřebovat? Bude nám stačit obyčejná kostra? O každém kloubu těla víme, že má určitou tuhost, která není pro všechny klouby stejná. To samé se týká třeba i pevnosti (po jejím překročení se kloub přelomí vejpůl) a samozřejmě nesmíme opomenout, že ne všechny klouby mají stejný tvar (strukturu), což ovlivňuje jejich způsob ohýbání (uvažme např. kyčelní a kolenní kloub). Tyto vlastnosti obyčejná kostra nepokrývá, takže o ně budeme muset kostru rozšířit. Tato data je opět nutné přidat ke kostře ručně.

Ve výsledku pak pomocí těchto rozšířených parametrů necháme prostředí působit na našeho agenta – docílíme tím poměrně realistického efektu – agent se skácí ze schodů. Někdy se navíc tento postup kombinuje se standardními animacemi. Výsledek je pak ještě o stupínek lepší, i když náročnější na výpočet. Více se o této problematice lze dozvědět v textech [31], [32].

Inverzní kinematika

Máme vůbec něco takového, co by se podobalo výše zmíněnému ubrousku? Něco, co by nám řeklo, jaké kvaterniony budeme potřebovat, například při onom sbírání věcí ze stolu a ze země?

Jednou možností je to vypočítat. Fyzikové se pohybem zabývají už docela dlouhou dobu. Určitě si mnozí z vás vzpomenou na fyzikální disciplínu zvanou „kinematika“. Pokud máme ocelové rameno s alespoň jedním kloubem, upevníme jeden jeho konec a pak za druhý zatáhneme, fyzik je schopen s velkou přesností říci, jak bude výsledný pohyb *celého* ramene vypadat. Představa, že panák obdobně natáhne k předmětu konec ruky (prsty a dlaň) a zbytek těla se automaticky přizpůsobí, je docela lákavá. Sice ten zbytek těla se v reálném světě na tomto pohybu podílí velmi aktivně svými svaly, ale potřebujeme to takto precizně? Vždyť chování jako ve výše zmíněném příkladu s ocelovým ramenem by nám úplně stačilo! A skutečně – tento přístup se v počítačové grafice používá pod názvem *inverzní kinematika* („inverzní“ proto, že je zbytek těla jenom tahán, a sám netlačí jako v reálném světě) a dává celkem dobré výsledky.

Neradujme se ale předčasně, není to tak jednoduché, jak to může vypadat na první pohled. Za vším jsou celkem složité rovnice a navíc je zapotřebí ke kostem přidat i „klouby“ s danou (rozumnou) tuhostí a rozsahem pohybu, což je opět práce pro člověka, nikoli pro počítač. Navíc je důležité si uvědomit, že tyto rovnice nemusí mít jednoznačné řešení: když zatáhnete za konec ruky nebo ocelového ramene, klouby se mohou do výsledné pozice dostat více způsoby.

Ve hrách se takto řešená inverzní kinematika používá často u jeřábů nebo jiných strojů, kde nevadí (ba je přímo žádoucí) strojený a strojový vzhled těchto animací. O inverzní kinematice se píše ve většině základních učebnic o animacích, například [23], [45]. Doporučit lze také text [20].

Blendování animací

Typickým problémem, který se v rámci animování agentů řeší, je *blendování* (česky někdy spojování, mixování) animací. Obvykle animace fungují tak, že tělo agenta ovládá právě jedna animace, agent například buď jde, nebo běží. Ale co v případě, že agent přechází z chůze do běhu? Mohli bychom si připravit přechodovou animaci a použít ji po skončení probíhajícího cyklu chůze (přechodová animace by začínala v základní pozici chůze a končila v základní pozici běhu). To má tu nevýhodu – kromě toho, že je to práce navíc –, že to může dost dlouho trvat, což je problém, zvláště když se jedná o hráčem ovládanou postavu. Také můžeme problém ignorovat a prostě například z 10. snímku animace chůze přejít na 11. snímek animace běhu. To je řešení, které bylo dřív celkem běžné, ale tehdy byly také postavy hrubší a monitory měly menší rozlišení. Dnes je preferovaný způsob blendování, neboli prolínání, animací. Pro každou kost se vezmou hodnoty z obou animací a vhodným způsobem se zprůměrují (zde se opět s výhodou využijí zmíněné kvaterniony). Jedná se o vážený průměr, kde se váhy mění tak, že jejich součet je stále 1; na začátku je váha animace, do které přecházíme (v našem případě běhu), rovná 0, na konci je rovná nule váha výchozí animace (v našem případě chůze). Výsledkem je plynulý a většinou celkem věrohodný přechod mezi oběma animacemi.

Blendování animací se dá též využít k inverzní kinematice. V takovém případě se postupuje tak, že grafici-animátoři vytvoří několik verzí stejné animace, například krok 30 cm dlouhý a krok 60 cm dlouhý. Pokud pak budeme chtít udělat krok 40 cm dlouhý, zprůměrujeme obě animace s váhami 0,67 a 0,33.

Obdobný problém je skloubení dvou různých animací. Řekněme, že máme předpřipravenou animaci chůze a otočení hlavy. Mohli bychom je automaticky spojit? Animace chůze se stará o nohy a zbytek těla ji už tolik nezajímá. Animace otočení hlavy zase hýbe pouze hlavou. Kosti, které tyto animace rozhýbávají, se tedy nepřekrývají a spojení animací je poměrně jednoduché: pustíme animaci chůze na nohy a animaci otočení hlavy na hlavu a máme agenta, co po nás pokukuje, když prochází kolem. Někdy jde obdobu tohoto postupu aplikovat i tehdy, když se některé kosti překrývají: použijeme na tyto kosti vhodně zvolený vážený průměr.

Více se o blendování animací lze dočíst například v [27], [19].

3 Kde bydlí virtuální agent?

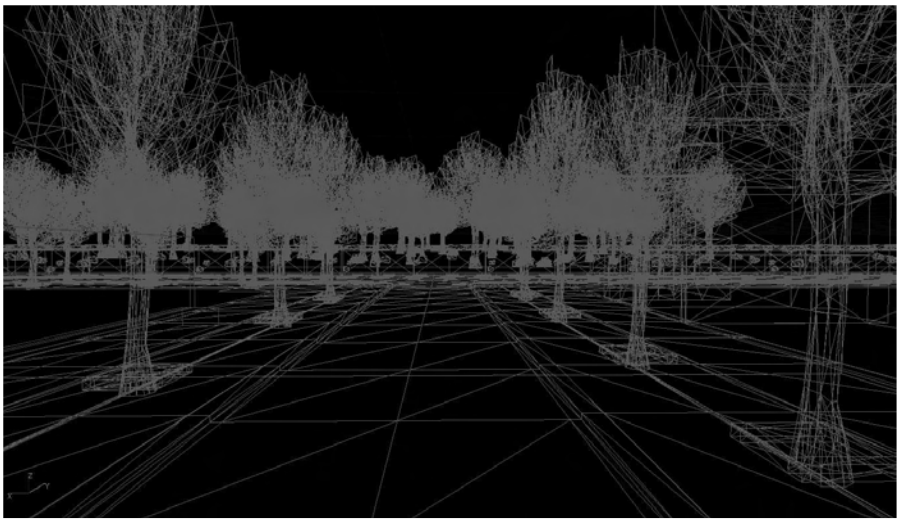
Otázka má mnoho rovin, podobně jako otázka, z čeho virtuální agent sestává. My se zde ptáme na virtuální svět a myšlenkové konstrukty, které designéři používají k jeho návrhu.

Virtuální svět (obr. 5) je tvořen podobně jako tělo panáků: z polygonů (obr. 6) potažených texturami. Drátovému modelu se často říká *geometrie světa*. O konkrétním zobrazení světa rozhoduje ale i mnoho „nehmotných“ prvků, například pozice světel či přítomnost mlhy. V dnešních počítačových hrách mohou virtuální agenti hru světel a stínů také vnímat, například ve hře Thief [35]. Někdy ovšem o těchto prvcích agent vůbec neví: tyto prvky jsou zde jenom z „okrasných“ důvodů: pro lidského diváka. To znamená, že když se schováte do příšeří, agent vás uvidí normálně.

Některé předměty mohou být součástí geometrie, tzn. nelze s nimi pohybovat; například stromy. Jiné předměty může jít přenášet; například batoh. Možná tušíte, že může být docela problém, aby virtuální agent *viditelně* nosil v rukou předměty. Když živý člověk drží předmět, jeho končetiny se kvůli tomu všelijak deformují: to znamená, že podobně bychom museli deformovat i kostru agenta. To je v lepším případě spousta práce, v horším to vůbec nejde: například agent může mít dlaňovou kost, ale už ne kosti pro jednotlivé prsty: a pak mu prsty jednoduše neohnete. V některých hrách, kde se často nosí předměty jenom jednoho typu (ano, například zbraně), se to řeší tak, že se kostra a drátový model vytvoří v pozici, kdy agent něco nosí. Pokud agent nic v rukou nemá, stejně vypadá, jako by něco držel. Jindy se zkrátka přenášený předmět nezobrazí a umístí se do agentova pomyslného batohu – *inventáře*.

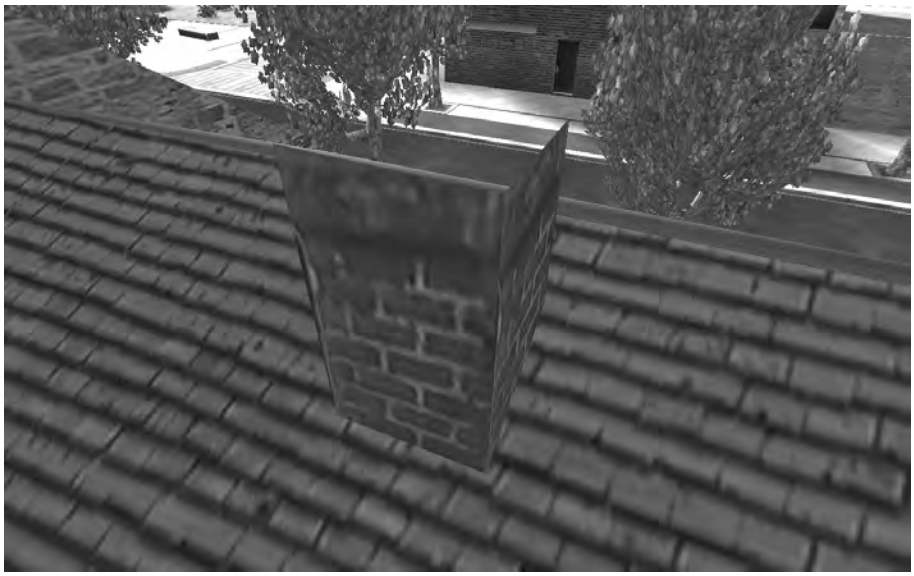
Obr. 5. Ulice města. Na konci ulice vidíme stromy za zdí. Zeď svět ukončuje.

Jiným typickým problémem je, jak svět „ukončit“. V mnoha aplikacích končí svět skálou nebo stěnou, případně se „konec“ utápí v mlze. To, že virtuální svět je jako filmová kašírka, která má pouze vytvářet iluzi reality, se dobře projevuje v místech, na která byste se „normálně“ neměli dostat, například právě za koncem světa (obr. 7, 8).



Obr. 6. Stejná ulice jako na obrázku 5, tentokrát z drátového modelu.

Obr. 7. Stromy za zdí (z obr. 5) z pohledu zvenku. Všimněme si, že svět končí pár metrů za zdí a některé ze stromů se vznáší ve vzduchu – klasický trik.



Obr. 8. Detail komínu shora. Všimněme si, že zadní část stěny komínu chybí. Komín není tvořen vlastním modelem, ale pouze texturami, které jsou z jedné strany průhledné. Při pohledu zdola toto není patrné.

3.1 Širší kontext

Pro úplnost dodejme dvě věci. Zaprvé, mohlo by se zdát, že virtuální agenti obývají jenom světy počítačových her. Je sice pravda, že počítačové hry se stávají plnohodnotným médiem – podle ESA [21] například roční tržby herního průmyslu překonaly v USA tržby z kin, hráčům je v USA v průměru 35 let a 40% z nich jsou ženy – a není tudíž důvod, proč bychom se měli za počítačové hry stydět, nicméně virtuální agenty nalezneme i jinde. Snad nejzajímavějšími oblastmi použití jsou výukové aplikace, terapeutické aplikace a různé druhy simulací [6], [10].

Zadruhé, pojem *agent* se používá i pro entity jiného typu než *virtuální* bytosti. Obecně se požaduje, aby šlo o entitu v nějakém smyslu autonomní či inteligentní. Takováto entita musí být schopna se sama rozhodovat (nepotřebuje k tomu pokyn od člověka) a tato rozhodnutí musí dávat smysl (musí sledovat uživatelův cíl). O jemnosti definice agenta se vedou spory (viz např. [25], [58]) – není totiž příliš jasné, co to znamená „inteligentní“ a „autonomní“. Wooldridge [58] uvádí k zamyšlení příklad s topením a termostatem. „Hloupé“ topení, které můžete pouze zapnout a vypnout, nelze rozhodně nazvat autonomním – jeho stav řídí člověk. Jelikož takové topení nepodniká vlastní akce, nemůže se jevit ani jako inteligentní – je člověku pouze užitečné, dokáže mu vyhřát obydlí. Ale co když takové topení hřeje přespříliš? Člověk jej musí sám vypnout a až začne být zase zima, musí jej opět zapnout. S tím však může být potíž v šestipokojovém rodinném domě, kde udržovat člověku příjemnou teplotu 22 stupňů znamená neustále zapínat a vypínat různá „hloupá“ topení. Proto člověk vymyslel termostat. Z hlediska funkčnosti se můžeme na termostat dívat jako na

způsob, kterým člověk dokáže sdělit topení, jakou teplotu má ve svém okolí udržovat. Topení se tak stává „chytrým“, je schopné autonomně reagovat na změny teploty ve svém okolí a podle potřeby se samo zapnout nebo vypnout. Nasadíme-li si „agentí“ brýle, můžeme takovéto topení s termostatem vnímat jako autonomního agenta, jehož svět je tvořen jedinou veličinou (teplotou okolí). Člověk takovémuto agentu řekne, jakou teplotu má udržovat v domě, a agent již sám podniká potřebné akce k dosažení tohoto cíle. Když však brýle sundáme, vidíme místo agenta kus železa, co hřeje. Lze jej nazvat inteligentním? A o to se, zjednodušeně řečeno, právě vedou spory.

Do definice „být inteligentní“ nakonec často spadnou živé organismy, roboti[2] a mnoho programů – například prohledávačů webu. Pojem lze ovšem nejen upřesňovat, ale také zjemňovat a vytvářet všelijaké podskupiny, např. *softwarové agenty* nebo *inteligentní softwarové agenty* a pod. Konkrétně ti pyšnící se adjektivem „inteligentní“ tvoří samostatnou, poměrně rozsáhlou výzkumnou oblast (blíže viz [58]). Pro náš typ agentů je klíčové adjektivum „virtuální“. Odráží se v něm dvě věci: fakt, že mají virtuální tělo, a fakt, že bydlí ve virtuálním světě. Například světem mnoha inteligentních softwarových agentů je Internet a žádné tělo v našem smyslu nemají.

O vztahu virtuálních agentů k jiným agentům se více píše v [10], [7].

V kontextu virtuálních agentů lze o autonomii říci následující. Při návrhu a tvorbě virtuálního agenta, ať už mluvíme o jeho vizuální podobě nebo o jeho chování, je třeba mít na paměti, k jakému účelu ho konstruujeme. Většinou není naším cílem simulace reality ale napodobení reality, vytvoření virtuálního světa, který zprostředkuje divákovi *iluzi*, že se nachází ve funkčním a fungujícím světě. Toho je zpravidla jednodušší dosáhnout vhodným zjednodušením problému než jeho plnou simulací.

Vezměme si jako příklad hru, kde hráč může chodit nebo jezdit po městě, v zásadě libovolným způsobem (jako je např. GTA nebo Mafie). Bylo by možné simulovat automobilový provoz v celém městě, s tisíci nebo deseti tisíci agenty, kteří by měli svoje autonomní cíle a jejich interakcí při dosahování těchto cílů by emergentně vznikal automobilový provoz. Nebo je možné simulovat provoz jenom v okolí hráče a celý ho bude řídit jeden „loutkář“, pohybující jednotlivými auty jako loutkami (a který bude mít nějaká pravidla, jak má hustota provozu a typ aut záviset na místě a denní době). Druhý způsob je zatím stále mnohem schůdnější a používanější. K tomuto problému se ještě vrátíme na konci kapitoly.

4 Virtuální agent chodí a ohmatává svět

Virtuální agent se může ve virtuálním světě pohybovat – od toho koneckonců celá simulace je. Z pozice programátora to vypadá tak, že agentovi přikáže přehrát animaci chůze, nebo jiného typu pohybu, z pozice <x1, y1>, kde agent právě stojí, na <x2, y2>, kde má stát po ukončení animace. Přitom ovšem programátor narazí na několik problémů:

[2] Další nejasností je, jestli by se měli chápat agenti a roboti jako životní nebo neživotní (tj. jsou to ty roboty a ty agenty nebo ti roboti a ti agenti?) Zde se kloníme k druhé variantě, protože je nám sympatičtější.

1. Jak daleko může být <x2, y2> od <x1, y1>?
2. Co když je mezi <x1, y1> a <x2, y2> překážka?
3. Co když se mezi <x1, y1> a <x2, y2> objeví překážka poté, co animace začne?
4. Jak agent vlastně ví, kudy má jít?

Odpovíme postupně. V navazující kapitole se pak budeme ptát, proč by vlastně měl agent někam chodit: jaký je účel jeho pohybu?

1. Jak daleko může být <x2, y2> od <x1, y1>?

Odpověď je jednoduchá: tak daleko, jak daleko „jde animace“. Pokud máme animaci dvojkroku, kterou jsme získali pomocí motion capture, a herec přitom ušel metr a půl, i virtuální panák, pokud je stejně veliký jako herec, by měl ve virtuálním světě ujít metr a půl. Animaci lze pustit i na delší nebo kratší vzdálenost, například „natáhnout“ ji na tři metry, ale panák potom bude nepřirozeně „klouzat“.

2. Co když je mezi <x1, y1> a <x2, y2> překážka?

Představme si, že náš agent hraje špatného úředníka, který z nějakého důvodu „chce“ opustit svou kancelář. Na pozici <x1, y1> bude jeho stůl, na pozici <x2, y2> budou dveře. Na první pohled triviální: natočíme panáka směrem ke dveřím a přehrajeme za sebou tolik dvojkroků, aby to tak akorát vyšlo až k východu (pokud nebude vzdálenost v metrech dělitelná 1.5, poslední dvojkrok nějak „zašudláme“). Ale co když bude stát v cestě stůl?

Řeší se to takto: představte si, že úředníkovi kouká z pupku několik provázků, za jeden z nich ho někdo, říkejme mu dveřník, tahá směrem ke dveřím. Jakmile si úředník všimne, že mu v cestě stojí stůl, hodí jeden ze zbývajících provázků kolegyni, která stojí bokem – v rohu u okna (obr. 9). Tahají teď oba: dveřník i kolegyně. Výsledek bude, že úředník půjde trochu bokem – mezi roh a dveře. Pokud si stolu všiml včas, podaří se mu ho bez úhony obejít. Jakmile bude na straně stolu, požádá kolegyni, aby přestala tahat a hodila mu provázek zpátky.

Proč ve chvíli, kdy ouřada obchází stůl, nepřestane úplně tahat dveřník? Kolegyně by přeci odtáhla úředníka na stranu stolu rychleji; dveřník jí to jenom „kazí“. Důvod je ten, že by úředník zatočil ve chvíli, kdy by jeho kolegyně začala tahat, příliš prudce; takhle normální lidé nechodí, ti většinou zatáčí plynule. Díky plynulosti to bude ve skutečnosti ještě trochu složitější: úřednice bude napřed tahat málo, potom bude postupně zabírat více a nakonec bude postupně přestávat.

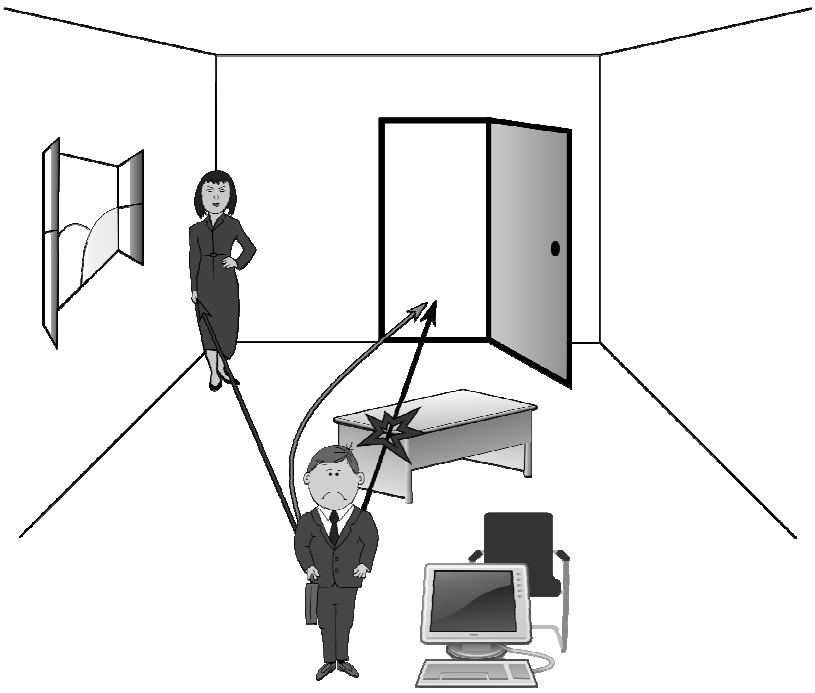
Formálně si ovšem představujeme úředníka (agenta) jako hmotný bod a provázky jako silové vektory. Úředník jde směrem daným složením těchto vektorů. Výpočetně je to velmi efektivní; v jedné počítačové hře takhle dokážete „tahat“ tisíce agentů. Trikem s provázky (silovými vektory) dokážeme udělat i složitější věci: aby se dva agenti navzájem vyhnuli, aby sledoval jeden druhého, aby se houfovali v davu, aby šli podél stěny a podobně. Těmto trikům se říká *steerovací techniky* či *pravidla* („steering rules“). Mimochodem, hodí se i na řízení (virtuálního) auta či houfu zvířat: třeba netopýrů z filmu Batman.

Ne vždy to ale vypadá dobře (obr. 10). Občas potřebujeme složitější mechanismus; vrátíme se k němu za chvilku.

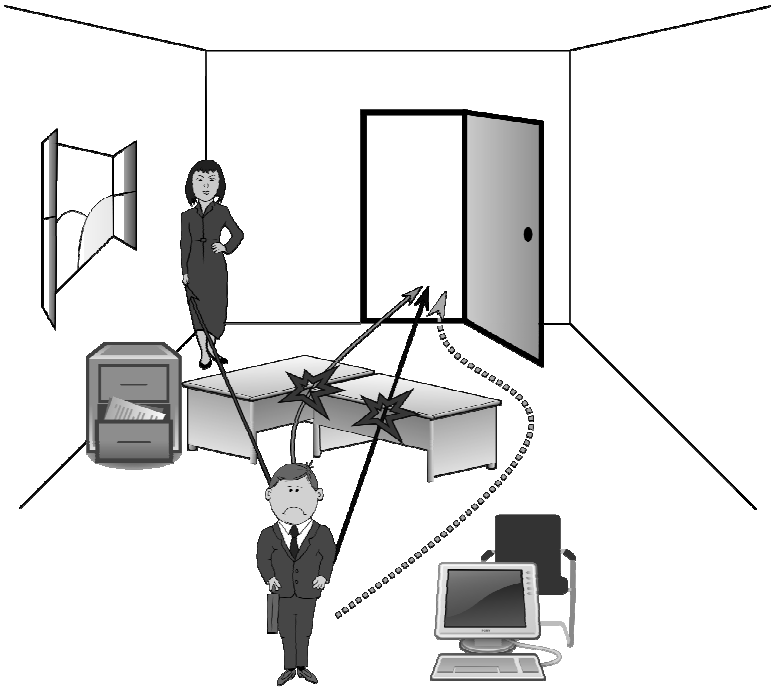
3. Co když se mezi <x1, y1> a <x2, y2> objeví překážka poté, co animace začne?

První část odpovědi je jednoduchá: pokud si agent překážky všimne, použije steerovací techniku pro vyhnutí se překážce (tj. použije „kolegyni“ vpravo či vlevo od překážky). Druhá část odpovědi je těžší: jak si agent překážky všimne? To nás přivádí k myšlence, že agent potřebuje nějakým způsobem *vnímat* svět.

Vzpomeňte si na pojem geometrie prostoru. Geometrii prostoru potáhneme texturou, uděláme její projekci do roviny a tu zobrazíme na monitoru (obr. 5). Vidí náš agent to, co člověk: 2D projekci geometrie potažené texturou? Rozhodně ne. Rozpoznávání toho, co je na 2D obrázcích (nebo videích) se věnuje oblast počítačového vidění. Nikdo na světě ale zatím neumí udělat „umělý zrak“, který by viděl tak dobře, jako vidí člověk: rozpoznával stovky různých objektů, v různém natočení a různě velikých, třebas i částečně zakrytých; dokázal poznat, kdo co dělá, a podobně. Musíme na to jít jinak. Klíčové je uvědomit si, že virtuální agent, který se má vyhýbat překážkám, toho zas tolik vidět nemusí: stačí mu vidět právě ty překážky.



Obr. 9. Úředník jde od počítače ke dveřím a přitom se vyhne stolu. Při obcházení stolu je dočasně „přitahován“ do rohu místnosti.



Obr. 10. Jednoduchý steering dovedl úředníka do pasti mezi uskupení objektů. Chytřejší by bylo vyhnout se celému uskupení najednou.

Jednoduchý trik je následující: vybavíme agenta dlouhými vousy, kterými bude ochmatávat terén před sebou, podobně jako slepec zkoumá prostor holí nebo potkan hmatovými vousy (obr. 11). Vousy informují agenta o tom, že narazil na překážku a v jakém je směru. Přesněji se tomu říká technika *vrhání paprsku* („ray-casting“; paprsek = vous). Kolik má takový agent vousů? Čím více, tím je větší šance, že detekuje překážku. Na druhou stranu každý vous navíc představuje výpočetní zátěž pro procesor. Takový agent z počítačové hry Unreal Tournament 2004 má vousy jen tři. Má je u kolen; do této výšky zasahuje většina překážek (obr. 12). Co když je překážka menší nebo je ve výšce očí? Inu, tak do ní agent vrazí... nebo projde skrz.

Kdyby tedy náš úředník chodil jako panák z Unreal Tournamentu, měl by být úplně slepý a mít tři vousy u kolen (trochu jako Děd Vševěd). Vidíme, že i mozek špatných úředníků toho zvládne víc než mechanismus panáků pro vyhýbání se překážkám.

4. Jak agent vlastně ví, kudy má jít?

I špatný úředník si po několika dnech v práci zapamatuje, že chce-li odejít z kanceláře, musí nejprve obejít stůl. A že obejít stůl znamená jít nejprve k oknu a pak až ke dveřím kanceláře. Tuto cestu si vybere ze dvou důvodů. Jednak je to nejkratší cesta ven z kanceláře a jednak na její trase nejsou žádné překážky.

Obr. 11. Hmatové fousy potkana.

Obr. 12. „Fousy“ virtuálního agenta. „Fous“ vlevo detekuje překážku.

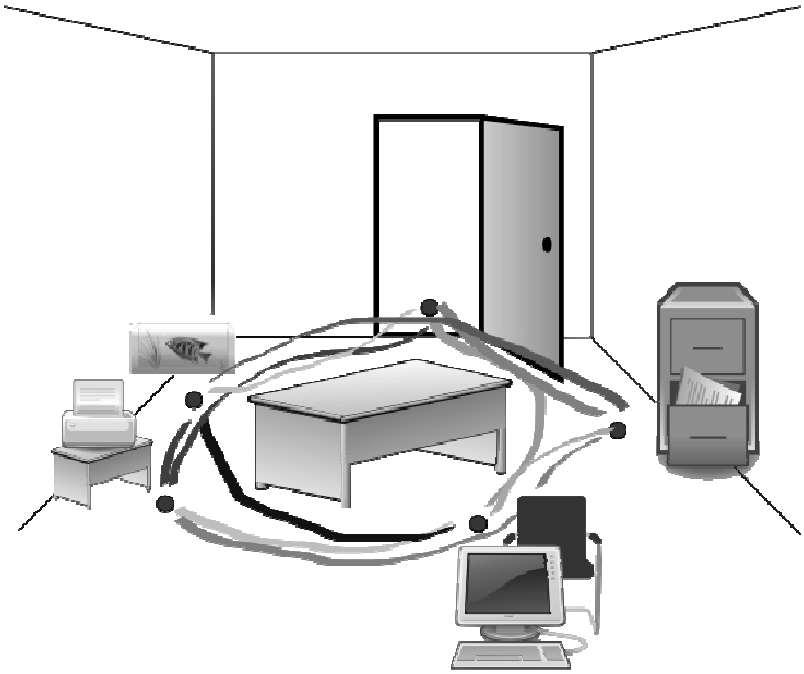
Pokud je náš špatný úředník svědomitý, může se stát, že jej povýší a přestěhuje se do velké kanceláře s velkým stolem, počítačem, tiskárnou, registraturou na šanony a akváriem. Zde již úředník potřebuje znát více cest než jen od své židle ke dveřím. Musí umět zajít si pro vytisknuté dokumenty, odtud je zanést do registratury a čas od času nakrmit rybičky. V hlavě tedy nosí jakousi mapu kanceláře, do které má zaznačeno, jak přecházet mezi jednotlivými místy (obr. 13). Všimněme si, že pro samotný pohyb jsou důležitá pouze místa a cesty – z obrázku by klidně mohl zmizet stůl či podstavec s akváriem. K samotnému pohybu nepotřebujeme vědět, že obcházíme stůl nebo že se vyhýbáme podstavci s akváriem, důležitá je pro nás pouze informace cest, po kterých se můžeme bezpečně pohybovat. Jinými slovy náš úředník nepotřebuje vidět, stačí mu chodit po paměti (představte si, že jdete v noci na záchod a praskla žárovka).

Problém s mapou na obr. 13 je, že počet cest je kvadratický vzhledem k počtu vrcholů – musíme se umět dostat odkudkoli kamkoli (snad až na triviální cesty typu akvárium – akvárium). Pokud bychom chtěli do takovéhoto grafu zahrnout i kanceláře podřízených, neřku-li celou budovu úřadu, dostaneme se rychle do problémů. Určitě vás napadá jednoduchý trik. Bližším pohledem na cesty v kanceláři zjistíme, že cesta od židle úředníka k tiskárně je velice podobná cestě k akváriu, liší se jen dodatečným krokem k akváriu. Můžeme tedy zrušit cestu židle – akvárium a zapamatujeme si, že cesta od židle k akváriu vede přes tiskárnu. Podobným způsobem upravíme všechny cesty, které mají některé části společné (obr. 14). Vznikne nám tak graf, kterému se říká *navigační*, a jeho vrcholy se označují jako *navigační body* („way points“ nebo také „navigation points“ či zkráceně „navpoints“). Odpověď na otázku „kudy má agent jít?“ se tak mění na úkol nalezení cesty v grafu z výchozího místa do požadovaného cíle. Obecně se jedná o úlohu nalezení nejkratší cesty v grafu s kladným ohodnocením hran. Úlohu můžeme řešit například pomocí Dijkstrova algoritmu [18], A* [52] či Floyd-Warshallovým algoritmem [17].

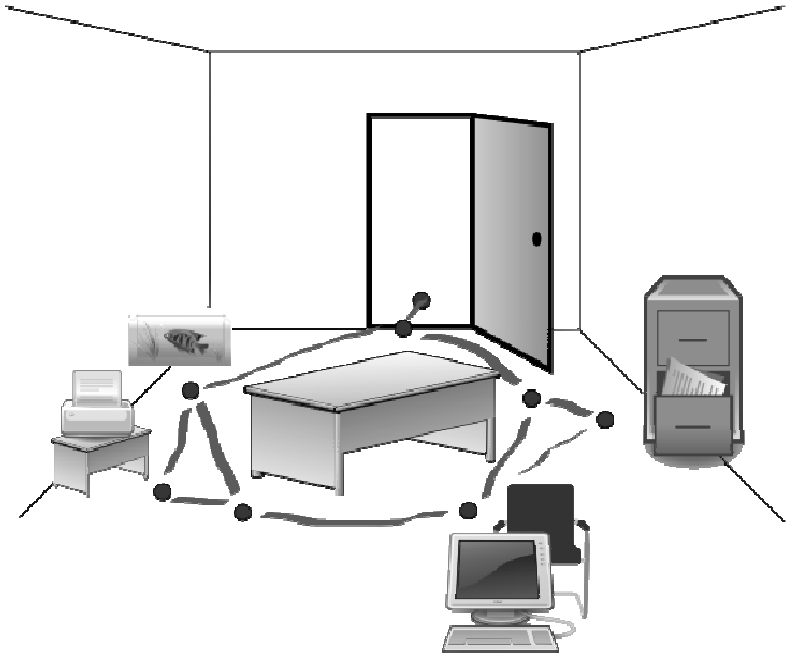
Mohlo by se zdát, že takovýto graf je vše, co náš virtuální agent potřebuje k bezpečnému pohybu po úřadě. Avšak opak je pravdou. Náš graf je totiž statický; je vytvořen s představou, že zdi kanceláře se nezmění přes noc, nábytkem nikdo nebude hýbat a po cestičkách se nebudou pohybovat jiní úředníci. Co když ale stejný graf využívá vícero úředníků a střetnou se na chodbě? Zde je ještě pomoc jednoduchá: zkombinujeme hledání cesty v grafu s navigačními pravidly popsanými výše. Ale co když nějaký zlomyslník schválně šoupne v noci se stolem uprostřed kanceláře a přehradí cestu v grafu? Co když dojde k přestavbě úřadu a do kanceláře povedou nové dveře nebo staré budou zazděny? Ukazuje se, že náš špatný úředník bude opět chytřejší než virtuální agent. Úředník si jednoduše přestaví svou mapu v hlavě a přizpůsobí se novým podmínkám. Toto „přizpůsobení“ agentům již jde hůře, ale i je dokážeme naučit, aby svůj graf uměli revidovat (viz dále).

4.1 K dalšímu čtení a zamyšlení

Techniku steeringu důkladně rozpracoval už v 80. letech Craig Reynolds [49]. Existují samozřejmě i různá rozšíření, některé práce jsou publikované ve známé sérii knih o umělé inteligenci v počítačových hrách AI Game Programming Wisdom [48].

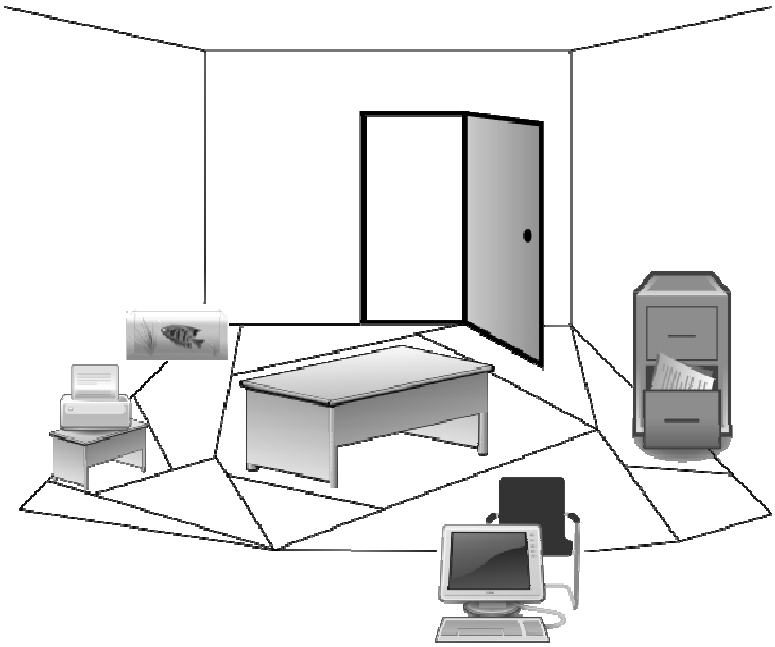


Obr. 13. Všechny možné cesty v úředníkově nové kanceláři.



Obr. 14. Navigační graf.

Způsobů, jak tvořit „mapu“ prostředí po agenty, je více. Pokud jste se zarazili nad přílišnou jednoduchostí navigačních grafů, zarazili jste se správně. Jednoduchý navigační graf popisuje prostředí velice hrubě – zkuste si představit, že nevnímáte nic víc než pár bezrozměrných čar spojující pár důležitých míst ve vašem bytě. Co budete dělat, když po stejné cestě již někdo půjde? Jak daleko se od ní můžete odchýlit, aniž byste do něčeho narazili? Jedním ze způsobů, jak tento problém vyřešit, je použít techniku *navigační mřížky* („navigation mesh“ nebo též „navmesh“) ([54]). Technika vychází z myšlenky využít známou geometrii virtuálního prostředí a automaticky ji transformovat na mapu složenou z trojúhelníků nebo obecně polygonů (obr. 15). Tyto trojúhelníky (polygony) potom použijeme podobně jako vrcholy navigačního grafu. Sousednost dvou trojúhelníků je analogická hraně grafu. Trojúhelníky ale navíc obsahují informaci o tom, kde se může úředník bezpečně pohybovat.



Obr. 15. Navigační mřížka.

S pohybem agentů se pojí také další problémy než jen reprezentace prostředí. Nové problémy přináší například veliké virtuální světy, ve kterých jsou klasické algoritmy na hledání nejkratší cesty v grafu pomalé. Problémy způsobují výtahy, dveře či schody, při kterých se musí klást zvýšená opatrnost při spouštění animací, a v neposlední řadě problémy přináší také koordinace pohybu velkého počtu agentů či pohyb skupinek agentů ve formacích. Pokud prostředí začne být příliš velké, řeší se vyhledávání cesty v grafu pomocí hierarchického plánování cest [4]. Výtahy, dveře a schody lze řešit například speciálními navigačními body, ve kterých se animace synchronizují a pohyb velkého počtu agentů lze elegantně řešit již zmíněným steeringem.

Hledání cesty je problém, který musí vyřešit skoro každá hra a tak není divu, že na tento problém existují programy, zvané *middleware*, které mohou vývojáři integrovat do svého projektu. Příkladem middlewaru, který využívá navmeshe je třeba PathEngine [46] nebo Kynapse [1] (tento middleware toho ovšem umí o hodně víc, než jenom hledání cesty).

4.2 Následuj toho, koho vidíš!

Na závěr si popišme chování agenta – *stopaře* –, který dokáže následovat jiného agenta. Takové chování se často hodí v počítačových hrách.

Vytvoření stopaře je poměrně jednoduché. Jediné informace z prostředí, jež bude potřebovat, jsou pozice ostatních agentů, které momentálně vidí. Máme-li tyto informace k dispozici, můžeme vytvořit stopaře následovně: V momentě, kdy se v jeho zorném poli objeví cizí agent, vyšleme příkaz „jdi na pozici tohoto agenta“ („jdi“ pomocí navigačních pravidel; tzn. pokud se například po cestě objeví překážka, obejdi ji). V případě, že je agentů ve stopařově zorném poli více, bude stopař následovat nejbližšího z nich. Jak se lze zbavit stopaře, pokud nás sleduje? Zaprvé můžeme zmizet z jeho zorného pole. Zadruhé můžeme přiběhnout ke skupince agentů a doufat, že stopař začne sledovat někoho z nich. Nebo ho můžeme zkusit vlákat do pasti jako na obrázku 10.

Zbývá vyřešit jeden problém: co to znamená, že stopař někoho „vidí“? Obvykle to znamená, že se cizí agent (nebo objekt) nachází v jeho *zorném poli* („field of view“) a mezi stopařem a agentem není žádná neprůhledná překážka. Přítomnost překážky zjistíme již známou technikou vrhání paprsků: vrhneme paprsek z „očí“ agenta směrem k těžišti objektu: pokud paprsek narazí do překážky, řekneme, že objekt agent nevidí (dobrá, pokud je objekt zakrytý částečně, tak to nemusí fungovat, ale vrhat víc paprsků zase stojí výpočetní zdroje...). Šířka zorného pole agenta bývá obvykle zhruba 90°.

Existují i složitější mechanismy vnímání, ale používají se jen tam, kde je to opravdu nutné: například ve hře Thief, kde se agenti často pohybují v příšeří a kde jejich vjemy závisí na kdejakém šustnutí [35]. Výše jsme říkali, že agenti často nevidí zdroje světla, nevnímají stíny či mlhu. S vylepšenými mechanismy vnímání to nemusí být pravda. Na akademické půdě se také věnuje několik skupin modelování pozornosti (např. [33]). Jde o to, že když je kolem agenta hodně viditelných předmětů, neměl by si jich stihnout všimnout všech. Pomocí mechanismu pozornosti lze také modelovat oční pohyby: agenti vypadají věrohodněji, pokud se jejich oči natáčí podle toho, na který objekt se dívají.

K zamyšlení: dokázali byste agenta stopaře vylepšit tak, aby se vymotal z pasti (obr. 10), pokud je do ní naveden? Nápověda: zkuste využít navigačního grafu.

5 Agentova Kniha Předpisů

Náš agent umí chodit, ale tím to zatím končí. Jak ho donutit k tomu, aby udělal něco složitějšího, třeba přenesl někam předmět? Pamatujete na Knihu Předpisů špatného úředníka? Je to kniha, do které úředník nahlédne, aby zjistil, jak má zareagovat, když něco chcete. Náš agent potřebuje také takovou Knihu. Této Knize někteří lidé

programující virtuální agenty říkají *reaktivní plán*, jiní *procedurální paměť*, ale to teď není příliš důležité.

V Knize mohou být různé magické formule. Snad nejjednoduššímu typu takových formulí se říká *reaktivní pravidla*. Jsou to pravidla jestliže – pak:

```
if podmínka then příkaz
```

V Knize jich je několik. Takový virtuální prodavač z počítačové hry, který nikdy neopouští svůj krámek, by jich mohl mít třeba pět:

```
1. if hori then utikej pryc
2. if utok then bran se
3. if hrac se pta and neni utocnik then odpovez
4. if hrac je blizko then predstirej praci
5. else nic nedelej
```

Už vidíte paralelu se špatným úředníkem?

Pravidla bereme shora dolů, pokud platí více podmínek („if“ část), vezmeme to vyšší. Důležité je, že pravidla kontrolujeme neustále (přesněji: několikrát za vteřinu). Jde o to, aby agent včas dokázal zareagovat na změnu kontextu (baví se s hráčem, začne hořet).

Co to přesně znamená „utíkej pryč“ nebo „braň se“? V zásadě máme dvě možnosti. Zaprvé, může to znamenat udělat něco, co už umíme: pustit nějakou animaci, sekvenci animací (to bude případ „braň se“) nebo někam poodejít (to bude případ „utíkej pryč“). Může to také znamenat, že během animace zmizí z virtuálního světa nějaký předmět (agent ho „sebral“) nebo se tam objeví (agent ho „položil“) nebo se změní *stav* nějakého objektu (agent během operace „braň se“ rozbil vázu na polici). O předmětech řekneme více za chvíli.

Zadruhé, pokud jde o něco opravdu složitého, může to znamenat, že máme nalistovat v Knize Předpisů další stránku, kde jsou další pravidla. Představme si, že pravidlo 4 bude přikazovat, aby úředník – pardon, virtuální prodavač – opravdu pracoval:

```
4. if hrac je blizko then opravdu pracuj (viz str. 42)
```

To ale znamená spoustu věcí! Prodávat zboží, objednávat zboží, dělat inventuru a podobně. Na straně 42 náš prodavač najde příslušná pravidla. Tomuto „listování“ se říká *hierarchický* přístup. Důležité je, že až si prodavač prohlédne, co je na straně 42, vrátí se zase na předchozí stranu – ne jako v „gameboocích“.

Nyní se dostáváme ke klíčové otázce: kdo Knihu Předpisů připraví? Knihu musí od začátku až do konce napsat programátor nebo (v kontextu počítačových her) herní designér. Od začátku až do konce. A pak ji musí ladit a ladit, aby agent nedělal něco jiného, než co programátor chce. Je to moře práce.

Nedá se Kniha nějak... vygenerovat sama? Jako že by se panák sám něco naučil?

Nedá. Přesněji: nikdo to neumí. Říká se tomu „million dollar question“. Pokud přijdete na to, jak Knihu generovat automaticky, možná nedostanete Nobelovu cenu, ale rozhodně nikdo z vaší rodiny až do třetího kolena nebude muset do smrti pracovat. Lidé, co vyvíjejí počítačové hry, vás zasypou penězi. Vlastně jde o obdobu magického tlačítka HUP. Vzpomínáte? Generovalo samo animace. Zde máme jeho bratříčka HOPLA, který generuje automaticky chování.

Uvažte: člověku to trvá desítky let, než se naučí svou Knihu Předpisů. Navíc není jasné, jestli lidé (vyjma špatných úředníků) opravdu nějaké Knihy Předpisů v hlavách nosí. Virtuálními agenty se lidé zabývají tak 20 let. Opravdu je realistické očekávat, že by za tak krátkou dobu někdo přišel na algoritmus, kterým naučit virtuálního panáka to, co dělá živý člověk?

5.1 Agent nosí předměty a něco si při tom pamatuje

Zůstaňme ještě chvíli u našeho virtuálního prodavače. Představme si, že prodavači právě dorazila zásilka zboží do skladu a on by ji rád přenesl do svého obchodu. Aby to prodavač zvládl, přidáme mu následující pravidla:

```
if mam_vec and jsem_v_obchode then poloz_vec
if mam_vec then jdi_do_obchodu
if jsem_ve_skladu and vidim_vec then vezmi_vec
else jdi_do_skladu
```

Tato sada pravidel zajistí, že prodavač přenese všechno zboží ze skladu do obchodu. Všimněme si jedné věci. Náš prodavač už nereaguje jenom na prostředí, ale také na svůj vnitřní stav – podmínka `mam_vec`. V tomto případě se jedná o jednoduchou formu paměti – prodavač si pamatuje, že něco nese. Jednoduchá forma paměti v našem příkladu se nazývá *krátkodobá paměť*. Zjednodušeně řečeno je to datová struktura, do které ukládáme vše, o čem si myslíme, že se nám v brzké budoucnosti hodí. Časem to z ní musíme také vymazat – v našem případě ve chvíli, kdy prodavač věc položí.

Kdysi někteří namítali, že paměť nepotřebujeme [11]. Skutečnost je ovšem taková, že pokud bychom krátkodobou paměť k dispozici neměli, Kniha Předpisů by se nám psala mnohem hůře. V příkladu s nošením bedny to zas tak dramatické není: agent bez paměti by se mohl jednoduše podívat, jestli v rukách něco drží nebo ne – jako paměť by mu v jistém smyslu posloužilo externí prostředí. Někdy to ale takhle jednoduše nejde: uvažme telefonování. Zvednete sluchátko a nic se neozývá: pokud si nepamatujete, že jste právě sluchátko zvedli a že hovor má právě začít, nemáte jak se jenom na základě stavu vnějšího světa rozhodnout, jestli říct „Haló, je tam někdo?“ (začátek hovoru) nebo sluchátko položit (konec hovoru). Jinými slovy v jednom stavu světa můžete udělat dvě různé akce a dokážete se rozhodnout jenom podle toho, co jste dělali předtím. Tato situace se nazývá „perceptual aliasing problem“.

Nabízí se také otázka, jestli by si agent neměl pamatovat něco dlouhodobě. Psychologové nechápou lidskou paměť jako monolitickou strukturu, ale jako sadu systémů [56]. Zjednodušeně řečeno: jedna složka systému je *dlouhodobá episodická paměť* – to je paměť na minulé zážitky („včera jsem byl na Karlštejně"). Jiná složka je *sémantická paměť* – to je paměť na faktické informace („Karlštejn je v Čechách."). Další složka je *procedurální paměť* – paměť na motorické schopnosti. Obdobou dlouhodobé procedurální paměti je u virtuálních agentů Kniha Předpisů. Zatím se příliš nevytvářeli virtuální agenti, kteří by upotřebili sémantickou a episodickou paměť, proto výzkum v této oblasti teprve začíná. Není to tak jednoduché, jak to může na první pohled vypadat. Například episodická paměť nemůže fungovat jako prosté video. Tak jako si skuteční prodavači nepamatují všechno zboží, co jim kdy prošlo rukama, ani virtuální prodavač by neměl mít kompletní seznam všech beden zboží, co přenesl. Navíc by to stálo příliš mnoho výpočetních zdrojů. Jako úvodní čtení na toto téma lze doporučit například [9], [8].

5.2 K dalšímu čtení

Reaktivní pravidla jsou jenom jedním ze způsobů, jak psát do Knihy Předpisů. Mezi další způsoby patří například *konečné automaty* („finite state machines") nebo *stromy chování* („behavioral trees"). Úvod k těmto a dalším technikám můžete nalézt například v [10], [13], [14]) nebo opět v knihách ze série AI Game Programming Wisdom [48]. Komplexní text, který uvádí čtenáře do mnoha technik používaných pro tvorbu agentů v počítačových hrách je [30]. Pojednává dokonce i o několika nesmělých pokusech týkajících se učení. Jiný článek pojednávající o učení agentů v počítačových hrách je [24]. Nejde ovšem o učení celé Knihy Předpisů, ale pouze o drobné modifikace (například: umím už jíst a naučím se, že kráva chutná lépe než kamení). Problému učení se věnuje také open source hra NERO [41]. Agenti z této hry jsou namísto Knihy Předpisů řízeni umělou neuronovou sítí, pro jejíž učení se používají genetické algoritmy [55]. Pokud bylo agentovo chování uživatelem kladně ohodnoceno, budou jeho geny, kódující výsledné chování, s velkou pravděpodobností přeneseny i na další generaci agentů. Naopak geny špatně hodnocených agentů z populace vymizí. Může být zajímavé sledovat, jak se agenti postupně učí jednotlivá chování jako například chůzi podél stěn, výsledné chování ale zdaleka nedosahuje komplexnosti chování napsaného člověkem. Podobné tvrzení platí i o počítačové hře Creatures, která uvádí virtuální agenty, jejichž chování se rovněž částečně učí neuronovými sítěmi [28] (tentokrát bez použití genetických algoritmů).

Kniha Předpisů je ve skutečnosti často naprogramovaná přímo v programovacím jazyce, jímž je psaná počítačová hra (např. C++), nebo v „pomocném", tzv. *skriptovacím* jazyce, který základní jazyk doplňuje (např. Lua). Specielně pro rozhodovací pravidla ale také existuje několik programů, které umožňují takové Knihy psát a pak v nich vyhledávat – například reaktivní plánovač POSH [12], nástroj Jess [26] nebo nástroj Soar [57] (který toho ve skutečnosti umí mnohem víc).

Jak ví každý, kdo někdy potkal špatného úředníka, pomocí Knihy Předpisů se mnoho problémů vyřešit nedá. O omezení tohoto přístupu se píše v [5]. O alternativě se dočtete dále.

5.3 Umí agent také něco sám vyřešit?

Už víme, že Kniha Předpisů je návodem, jak má úředník reagovat na předem popsané situace. To ovšem nemusí stačit na to, aby se agent z našeho pohledu choval inteligentně. Představme si, že bychom chtěli, aby náš úředník vyřešil určitý problém. Pokud by měl pouze Knihu Předpisů a v té by řešení problému nenašel, nevěděl by, co má dělat a jen by stál a díval se před sebe, případně reagoval na jiné podněty z okolí. My bychom rádi, aby se úředník zamyslel a pokusil se *sám* najít řešení – potřebovali bychom chytrého úředníka.

Na takový úkol už Kniha Předpisů nestačí a je třeba naučit našeho úředníka přemýšlet. Nikdo neví, jak skutečný člověk myslí, ale existují techniky umělé inteligence, které některé aspekty myšlení alespoň částečně postihují – například plánování. Plánování je činnost, kterou se lidé zabývají každý den, ať už v malém (jak se ráno obléct) nebo ve velkém (jak postavit most). Samotné plánování se dá charakterizovat jako proces, jehož výsledkem je *plán* – posloupnost akcí (např. najdi tričko, obleč si tričko, najdi kalhoty, obleč si kalhoty), které je třeba provést, abychom se z určitého *počátečního stavu* (stojím v pokoji nahý) dostali do *cílového stavu* (stojím v pokoji oblečený).

Aby mohl úředník plánovat, musí vědět, jaké akce může provést. Akci lze popsat seznamem jejích *předpokladů* a *efektů*. K provedení určité akce potřebujeme, aby její předpoklady byly splněné. Například, pokud si chceme obléct tričko, musíme ho nejprve držet v rukou. Držení trička představuje předpoklad pro akci oblékání. Kromě předpokladů musíme vědět, jaké budou mít akce dopad na svět kolem nás, potažmo na svět kolem našeho úředníka. Jestliže si obleču tričko, dopadem (efektem) bude, že tričko už nedržím v ruce (toto se nazývá *negativní efekt*, protože už není pravda, že držím tričko v ruce), ale mám ho oblečené na těle (tzv. *pozitivní efekt*, protože se stalo pravdou, že tričko je na těle).

Ovšem ani popis akcí našemu úředníkovi na vyřešení problému – naplánování postupu – nestačí. Nejprve potřebuje znát výchozí stav, ze kterého plánuje – čili poznat stav světa, ve kterém se úředník nachází před tím, než začne provádět plán. Zde s výhodou využijeme systém vnímání, který jsme popsali výše. Problémem ale může být, že agent nezná stav světa kompletně: nevidí (nebo by neměl vidět) za zdi. Plánovat tak můžeme jen s částečnou informací.

Jak by ale náš úředník mohl vytvořit plán, kdyby nevěděl, k čemu se má dopracovat? Potřebuje tedy znát i cíl svého úsilí – cílový stav světa, ve kterém se má nacházet poté, co svůj plán uskuteční.

Samotné plánování – neboli schopnost úředníka vytvořit posloupnost takových akcí, kterými se dostane z výchozího stavu do cílového stavu – představuje velmi složitý a vědecky zajímavý námět. Nebudeme zde rozebírat jednotlivé algoritmy, ale existují dva základní postupy, jak k plánování přistupovat – dopředný a zpětný. *Dopředný* („forward“) způsob se pokouší budovat plán od počátečního stavu ke koncovému. *Zpětný* („backward“) způsob se snaží vycházet z konečného stavu a postupně se dopracovat až k počátečnímu.

Proč se vlastně používají Knihy Předpisů a ne plánování? Protože s Knihami Předpisů je to jednodušší a hlavně rychlejší. Knihy představují jakýsi spolehlivý

automat, který dobře funguje pro často se opakující situace. Uvažte: když si ráno čistíte zuby, také „neplánujete“ každý pohyb kartáčkem. Praktické omezení klasického plánování je rovněž v tom, že jeho výsledkem je pevně daná sekvence akcí: pokud se svět změní předtím, než se dostanete do cíle (což by se například v akčních hrách mohlo stávat dost často), původně naplánované akce přestanou dávat smysl. Agent by musel plán změnit, a protože samotné plánování nějakou dobu trvá, mohlo by se stát, že by agent nedělal nic jiného, než plánoval a plánoval. Je třeba najít způsob, jak Knihy Předpisů s plánováním kombinovat – to je ovšem otázka pro budoucí výzkum.

O plánování se více můžete dočíst například v [34], [38] nebo souhrnné knize [29]. V kontextu počítačových her opět odkazujeme na sérii AI Game Programming Wisdom [48] nebo článek o hře F.E.A.R. [44].

6 Spolupráce agentů

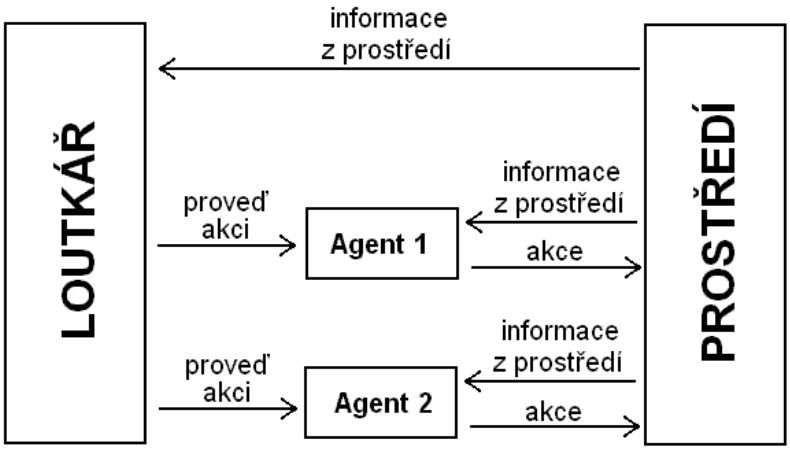
Představte si, že náš špatný úředník se pohádá se svou kolegyní o to, kdo má lepší sešívačku. V reálném světě vznikne taková žárlivá scéna sama od sebe (člověku se až dere na jazyk odborný termín *emergentně*). Jak je to ve virtuálním světě? Zde si musíme uvědomit jednu důležitou věc: ve většině aplikací nám nepůjde o to, aby se virtuální úředník s virtuální kolegyní opravdu hádali. Půjde o to, aby si to divák *myslel*. Ano, jako v loutkovém divadle chceme vytvořit *iluzi* hádky, chceme diváka *podvést*. V oblasti virtuálních agentů se v této souvislosti hovoří o *uvěřitelnosti* agentů („believability“) [36].

Za tím účelem použijeme spíše než poznatků z umělé inteligence triků z oblasti kreslených filmů. Takovým trikem například je, že by divák měl pochopit motivaci postav. Vědec studující umělou inteligenci nebo inteligentní softwarové agenty by mohl trvat na tom, že oba agenti by měli být autonomní. To ovšem trochu připomíná situaci, kdy loutkové divadlo hrají dva loutkáři naráz a nemají předem daný scénář. Jak jsme již naznačili výše, z technického hlediska bývá často jednoduší, když agenti pomyslně spojí síly, „domluví se“ na tom, jak má celá scénka vypadat, a pak ji přehrají. Jak vypadá ono „domluvení se“? Jsou dvě možnosti. Za prvé, buď si tajně vymění informace, aniž by o tom divák věděl („Hele, já se s tebou teď jako pohádám o to, kdo má lepší sešívačku.“ – „O kterou? Já mám dvě.“ – „O tu červenou.“ – „A kdo vyhraje?“ – „Já.“ – „Ne, já.“ – „Ok, tak ty. A v půlce hádky po tobě hodím jablko.“ – „Ok, a já uhnu a jablko rozbije okno.“). Často ovšem může být jednodušší druhý způsob: oba agenti se odevzdají do rukou centrálního *loutkáře* („drama manager“, „story director“), který je chvíli bude řídit oba (obr. 16).

Důležité je ovšem rozlišovat, jestli je scénka interaktivní nebo ne. Může vstoupit divák do děje a začít uklidňovat? Bez interakce je to jednoduché: v podstatě generujeme scénku z animovaného filmu. S interakcí ovšem musíme vymyslet spoustu reaktivních pravidel, které panákům říkají, jak mají zareagovat na tu kterou situaci, a běda, jak na nějakou situaci zapomeneme!

Centrální způsob řízení se ovšem nehodí ve chvílích, kdy nám opravdu záleží na tom, aby agenti jednali samostatně. Uvažte simulaci chování davu při krizových situacích: máme například (virtuální) stadion s tisícovkou lidí, náhle začne hořet, lidé začnou v panice utíkat, a my se ptáme, jestli jsou východy ze stadionu postaveny dost

dobře na to, aby všichni lidé ve zdraví utekli (podobný problém se řeší např. v [53]). Zde je samozřejmě důležité, aby každý agent jednal sám za sebe; nejde nám totiž o vytvoření iluze prchajícího davu, ale o model reálné situace. A v reálném světě se lidé většinou nedomlouvají přes centrálního arbitra o tom, jak utéct z hořící budovy.



Obr. 16. Architektura programu s centrálním „loutkářem“.

6.1 K dalšímu čtení

V akademické komunitě se problém spolupráce virtuálních agentů na přehrávání scének řeší zejména v oblasti disciplíny, která se jmenuje *virtuální vypravěčství* („virtual storytelling“). Zatím však, pokud můžeme soudit, k žádnému významnému pokroku nedošlo – je zkrátka potřeba správně napsat Knihu Předpisů. Někteří se snaží průběh scénky naplánovat. K dalšímu čtení lze doporučit články [15], [37], [51].

Z jiného úhlu se řeší problém spolupráce virtuálních agentů v týmových akčních hrách: zde je o to, jak řídit několik agentů, aby dosáhli společného cíle (např. získat vlajku). Agenti se také mohou domlouvat, ale většinou v rámci zachování věrohodnosti bez centrálního arbitra. Narozdíl od řízení jednoho agenta zde musíme dbát na to, aby si agenti vzájemně nepřekáželi, spolupracovali (např. krytí) a podobně. V kontextu počítačových her se tomuto problému věnují články [40], [42], [43], [50].

7 Závěr

Dospěli jsme na konec našeho příběhu. Co si z něj odnést? Především dvě věci. Zaprvé, pokud se chceme věnovat studiu virtuálních agentů, musíme si uvědomit, že jde o oblast z pomezí umělé inteligence a počítačové grafiky. Znalosti z těchto disciplín musíme vhodně skloubit. Zadruhé, chování virtuálního agenta musí někdo naprogramovat: nespadne samo z nebe. Hlavní technikou, která se dnes používá a ještě

dlouho bude používat je Kniha Předpisů (nebo její analogie), doplněná o navigační pravidla a mechanismy pro hledání cesty v grafu. Pokročilejší techniky umělé inteligence, jako například neuronové sítě nebo plánování, se zatím příliš nepoužívají, a to zejména díky komplexnosti a dynamice virtuálních světů. Obecně platí, že tyto techniky jsou vyzkoušeny jen v malých a statických světech (výjimky samozřejmě existují).

Koho zajímá, jak bude příběh virtuálních agentů pokračovat, nechť sleduje vhodné konference a fóra. Zejména lze doporučit konference Intelligent Virtual Agents, Artificial Intelligence for Interactive Digital Entertainment, Computational Intelligence and Games, Computer Animation and Social Agents, Autonomous Agents and Multiagent Systems a Interactive Digital Storytelling. Zajímavé fórum je aigamedev.com a relevantní žurnál IEEE Transactions on Computational Intelligence and AI in Games. Kdo by si chtěl vyzkoušet sám programovat virtuální agenty, může začít s projektem Pogamut [16].

Pro úplnost dodejme, že některým problémům jsme se zde nevěnovali, například problematice sociálního chování nebo modelování emocí. Některé z těchto témat jsou rozvinuty v [2], [10]. Další komplexní úvodní texty k problematice virtuálních agentů jsou [7], [47].

Poděkování: Tato kapitola vznikla za podpory projektu CZ.2.17/3.1.00/31162, který je financován Evropským Sociálním Fondem, rozpočtem České Republiky a rozpočtem hl. m. Prahy. Náš vlastní výzkum týkající se virtuálních agentů je podpořen projektem P103/10/1287 (GA ČR), výzkumným záměrem MSM0021620838 MŠMT ČR, grantem 201/09/H057 a GAUK 21809. Autorem obrázků 1-8 je Zbyněk Krulich, jemuž patří velký dík.

Literatura

[1] Autodesk: Kynapse project. URL: http://www.autodesk.com/kynapse [29.1.2010]

[2] Bída, M., Kadlec, R., Brom, C: Význam emocí pro umělé bytosti. In: *Myseľ, inteligencia a život*. Vydavateľstvo STU, Bratislava (2007) 158-172.

[3] Bourgh, D.M.: *Physics for Game Developers*. O'Reilly Media, 2001.

[4] Botea, A., Müller, M. and Schaeffer, J.: Near optimal hierarchical path-finding. *Journal of Game Development* 1 (2004) 7-28.

[5] Brom, C.: Hierarchical Reactive Planning: Where is its limit? In (Bryson J. J., Prescott T. J., Seth A. K., eds): *Modelling Natural Action Selection: Proceedings of an International Workshop Edinburgh*, Scotland, 2005, pp. 235 – 242.

[6] Brom, C.: Virtuální učení. In: *Vesmír* 87/10 (2008) 706-711.

[7] Brom, C.: *Řízení virtuálních lidí ve velkých virtuálních světech*. Disertační práce. MFF UK, 2007.

[8] Brom, C., Lukavský, J.: Towards Virtual Characters with a Full Episodic Memory II: The Episodic Memory Strikes Back. In: *Proceedings of Empathic Agents*, AAMAS workshop, 2009, pp. 1 – 9.

[9] Brom, C., Pešková, K., Lukavský, J.: What does your actor remember. Towards characters with a full episodic memory. In: *Proceedings of 4th ICVS*, LNCS 4871. Berlin, Springer-Verlag, 2007, pp. 89 – 101.

[10] Brom, C., Šisler, V., Hoffmann, M.: Virtuální lidé. In (Mařík V., Stěpánková O., Lažanský J., eds): *Umělá inteligence 5*. Academia, Praha, 2007.

[11] Brooks, R.A.: Intelligence without representation. In: *Artificial Intelligence* 47, (1991) 139-159.

[12] Bryson, J.J.: Inteligence by design: Principles of Modularity and Coordination for Engineering Complex Adaptive Agent. PhD Thesis, MIT, Department of EECS, Cambridge, MA, 2001.

[13] Bryson, J.J.: Action Selection and Individuation in Agent Based Modelling. In (Sallach D. L., Macal Ch., eds): *The Proceedings of Agent 2003: Challenges of Social Simulation*, 2003.

[14] Bryson, J.J.: How to Make a Monkey Do Something Smart. URL: http://www.cs.bath.ac.uk/~jjb/web/how-to-monkey.pdf [29.1.2010]

[15] Cavazza, M., Charles, F., Mead, S.J.: Character-based Interactive Storytelling. In: *IEEE Intelligent Systems* 17(4) (2002) 17-24.

[16] Gemrot, J., Kadlec, R., Bida, M., Burkert, O., Pibil, R., Havlicek, J., Zemcak, L., Simlovic, J., Vansa, R., Stolba, M., Plch, T., Brom C.: Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: Agents for Games and Simulations, LNCS 5920, Springer, 2009, pp. 1 – 15. URL: http://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php [29.1.2010]

[17] Cormen, T.H., Leiserson, C.E., Rivest, R.L.: *Introduction to Algorithms* (1st ed.). MIT Press and McGraw-Hill, 1990, pp. 558 – 565.

[18] Cormen, T.H., Leiserson, C.E., Rivest, R.L, Stein, C. Section 24.3: Dijkstra's algorithm. In: *Introduction to Algorithms* (Second ed.). MIT Press and McGraw-Hill, 2001, pp. 595 – 601.

[19] Edsall, J.: Animation Blending: Achieving Inverse Kinematics and More. Gamasutra Online, July 4, 2003. URL: http://www.gamasutra.com/features/20030704/edsall_01.shtml [29.1.2010]

[20] Elias, H.: Inverse Kinematics – Improved Methods. Online. 2004. URL: http://freespace.virgin.net/hugo.elias/models/m_ik2.htm [29.1.2010]

[21] The Enternainment Software Association (ESA). URL: http://www.theesa.com/ [29.1.2010]

[22] Epic Games: UnrealTournament 2004. URL: http://www. unrealtournament.com [29.1.2010]

[23] Erleben, K., Sporring, J., Henriksen, K., Dohlmann, H.: *Physics Based Animation*. Charles River Media, 2005.

[24] Evans, R.: Varieties of Learning. In: *AI Game Programming Wisdom I* (Rabin S., ed.), Charles River Media, Inc., Hingham, Massachusetts, 2002, pp. 567 – 579.

[25] Franklin, S., Graesser, A.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In: *Intelligent Agents III Agent Theories, Architectures, and Languages*, LNAI Volume 1193, Springer, Berlin, 1997, pp. 21 – 35.

[26] Friedman-Hill, E.: *Jess in action: java rule-based systems*. Manning Publications, 2003.

[27] GAME ANiM Articles: Basics: Animation Blending. In: *GAME ANiM Online*. June 19, 2005, URL: http://www.gameanim.com/2005/06/19/blending-the-future-of-non-linear-animation/ [29.1.2010]

[28] Grand, S., Cliff, D., Malhotra, A.: Creatures: Artificial life autonomous software-agents for home entertainment. In (Johnson W. L., ed.): *Proceedings of the First International Conference on Autonomous Agents*, ACM press, 1997, pp. 22 – 29.

[29] Ghallab, M., Nau, D., Traverso P.: *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers, San Francisco, Ca, USA, 2004.

[30] Champandard, A.J.: *AI Game Development: Synthetic Creatures with learning and Reactive Behaviors*. USA, New Riders, 2003.

[31] Jakobsen, T.: Advanced Character Physics. In: *Proceedings of Game Developer Conference (GDC'01)*, San Diego, USA, 2001. URL: http://www.gpgstudy.com/gpgiki/GDC%202001%3A%20Advanced%20Character%20Physics [29.1.2010]

[32] Kennedy, R.: Contact Physics. Gamasutra Online. January 21, 2003. URL: http://www.gamasutra.com/view/feature/2903/contact_physics.php [29.1.2010]

[33] Kim, Y., Hill, R.W. Jr., Traum, D.R.: A Computational Model of Dynamic Perceptual Attention for Virtual Humans. In: *Conference on Behavior Representation in Modeling and Simulation (BRIMS)*. Universal City, CA, 2005.

[34] Lažanský, J.: Plánování. In: *Umělá inteligence I.* Academia, 2000, 184-216.

[35] Leonard, T.: Building an AI Sensory System: Examining The Design of Thief: The Dark Project. In: *Gamasutra Online*, March 7, 2003.

[36] Loyall, B.A.: Believable Agents: Building Interactive Personalities. Ph.D. diss. Carnegie Mellon University, 1997.

[37] Magerko, B.: Intelligent Story Direction in the Interactive Drama Architecture. In: *AI Game Programming Wisdom 3*, Charles River Media, 2006.

[38] Mařík, V.: Řešení úloh a využívání znalostí. In: *Umělá inteligence I.* Academia, 1993, pp. 33 – 66.

[39] Mulley, G., Bittarelli, M.: Ragdoll Physics. Essay. Rensselaer Polytechnic Institute, Computer Science Department, 2007. URL: http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S07/final_projects/mulley_bittarelli.pdf [29.1.2010]

[40] Munoz-Avila, H., Hoang, H.: Coordinating Teams of Bots withing Hierarchical Task Network Planning. In (Steve Rabin, ed.): *AI Game Programming Wisdom 3*, Charles River Media, 2006, pp. 417 – 427.

[41] NERO homepage. URL: http://code.google.com/p/opennero/ [26.1.2010]

[42] Orkin, J.: Simple Techniques for Coordinated Behavior. In (Steve Rabin, ed.): *AI Game Programming Wisdom 2*, Charles River Media, 2004, pp. 199 – 206.

[43] Orkin, J.: Applying Goal-Oriented Action Planning to Games. In (Steve Rabin, ed.): *AI Game Programming Wisdom 2*, Charles River Media, 2004, pp. 217 – 227.

[44] Orkin J.: 3 States & a Plan: The AI of F.E.A.R. In: *Proceedings of Game Developer's Conference*, 2006.

[45] Parent, R.: *Computer Animation, Second Edition: Algorithms and Techniques.* Morgan Kaufmann, 2007.

[46] PathEngine homepage. URL: http://www.pathengine.com/ [29.1.2010]

[47] Prendinger, H., Ishizuka, M.: *Life-like characters*. Springer, Berlin, 2004.

[48] Rabin, S. (ed.): *AI Game Programming Wisdom 1-4*. Charles River Media, 2002-2008.

[49] Reynolds, C.: A web page on agents and steering behaviour. URL: http://www.red3d.com/cwr/ [29.1.2010]

[50] Reynolds, J.: Team Member AI in an FPS. In (Steve Rabin, ed.): *AI Game Programming Wisdom 2*, Charles River Media, 2004, pp. 207 – 215.

[51] Riedl, M.O., Young, R.M.: An Intent-Driven Planner for Multi-Agent Story Generation. In: *Proc. Third ACM Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2004, pp. 186 – 193.

[52] Russell, S.J., Norvig, P.: A*Search. In: *Artificial Intelligence: A Modern Approach, 2nd Ed*. Upper Saddle River, N.J.: Prentice Hall, 2003, pp. 97 – 104.

[53] Shia, J., Renb, A., Chenb, Ch.: Agent-based evacuation model of large public buildings under fire conditions. In: *Automation in Construction* 18(3) (2009) 338-347.

[54] Snook, G.: Simplified 3D Movement and Pathfinding Using Navigation Meshees. In (Mark DeLoura, ed.): *Game Programming Gems*, Charles River Media, 2000, pp. 288 – 304.

[55] Stanley, K.O., Bryant, B.D., Miikkulainen R.: Real-time Neuroevolution in the NERO Video Game. In: *IEEE Transactions on Evolutionary Computation*, 9(6) (2005) 653-668.

[56] Sternberg, R.J.: *Kognivní psychologie*. Portál, 2002.

[57] University of Michigan: SOAR. URL: http://sitemaker.umich.edu/soar/home [24. 4. 2007]

[58] Wooldridge, M.: *An Introduction to MultiAgent Systems.* John Wiley & Sons, 2002.

[59] Žára, J., Beneš, B., Sochor, J., Felkel, P.: *Moderní počítačová grafika*. 2. vydání. Computer Press, 2005.