**Univerzita Hradec Králové**

**Fakulta informatiky a managementu**

**Katedra informačních technologií**

Zpětná vazba v učícím se systému

# Model v Netlogu

Diplomová práce

Autor: Bc. Michal Michna

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. RNDr. Kamila Štekerová, Ph.D.

Hradec Králové srpen 2016

**Prohlášení**

Prohlašuji, že jsem předloženou diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 12. 8. 2016 Michal Michna

**Poděkování**

Tímto bych chtěl vyjádřit velké poděkování vedoucí diplomové práce doc. RNDr. Kamile Štekerové, Ph.D. za její ochotu, vstřícnost, cenné připomínky k obsahu a čas, který mé práci věnovala.

## Anotace

Cílem diplomové práce bylo navrhnout a implementovat simulační model učícího se systému v prostředí NetLogo. Jako učící se systém byl zvolen lidský agent a jeho deklarativní paměť uchovávající informace a souvislosti mezi nimi. Teoretickým podkladem pro tvorbu modelu je teorie ACT-R a její modul deklarativní paměti. Po kalibraci výsledného modelu jsou výsledky srovnatelné s výsledky regresního modelu pro zvolená kalibrační data. Provedené experimenty s modelem se zaměřují na základní efekty pozorované u deklarativní paměti, vliv nastavení parametrů modelu na měřené výstupy a porovnání náhodné strategie učení se strategií, která se přizpůsobuje stavu paměti modelovaného agenta.

**Klíčová slova** NetLogo, model, simulace, stav, časování, systém, zpětná vazba, ACT-R, deklarativní paměť, aktivace, kontext, validace, kalibrace, efektivita učení, strategie učení, simulační experimenty

## Annotation

**Title:** Feedback in learning system - NetLogo model

The aim of this Master Thesis was to devise and implement a simulation model of a learning system in NetLogo. As a learning system was chosen a human agent and his declarative memory containing information chunks and connections between them. The model is built on ACT-R theory. After calibration the model outputs are equal with results of the regression model constructed for calibration data. The experiments with model focuses on the basic effects studying the principles of declarative memory, the effect of setting parameters of the model to measured outputs and comparison of the random learning strategy with strategy that adapts to the state of memory of the modeled agent.

**Keywords** NetLogo, model, simulation, state, timing, system, feedback, ACT-R, declarative memory, activation, context, validation, calibration, learning efficiency, learning strategy, simulation experiments

# Obsah

# 1 Úvod

Proces učení se zaměřuje na získávání, posilování a modifikaci znalostí. V kontextu inteligentních agentů a systémů jsou tyto znalosti následně využity pro rozhodování, kterému může být agent nebo systém vystaven. Toto rozhodování může být nedílnou součástí činnosti, jež agent vykonává, a souvisí s řešením jistého typu úloh, pro které byl agent navržen.

Pokud má mít agent schopnost učit se, nejsou jeho rozhodnutí prováděna pouze na základě informací získaných ze vstupu, ale i s využitím znalostí. Tyto znalosti, které má agent k dispozici, představují informace použitelné pro rozhodování. Jejich aktuálnost a platnost je dána podmínkami rozhodovací situace, ve které mají být použity, a tedy záleží na kontextu řešené úlohy.

Znalosti může agent získat jednak explicitně, formou deklarativních pravidel (například naprogramováním). V tomto případě je kontextem činnost, kterou agent vykonává. Druhým způsobem je získání znalostí vlastní zkušeností díky opakovanému vystavení identické nebo podobné rozhodovací situaci v minulosti. Kontext je pak získán opakovaným použitím dané informace při rozhodování a také díky datům a informacím, které má agent k dispozici před rozhodovací situací.

Opakovaný a správný výběr a uplatnění znalostí v rozhodování následně vede k rozvoji chování agenta. Toto se projevuje v první řadě zejména schopností efektivněji a rychleji plnit funkci, která je od agenta nebo systému vyžadována. Dále může vést k rozvoji nových schopností nebo strategií pro řešení komplexnějších úloh.

Při tomto procesu mohou být stávající znalosti získané v minulosti modifikovány novými informacemi získanými z výsledků nedávných rozhodovacích situací. Výsledky těchto rozhodovacích situací potom ovlivňují aktuální rozhodování a rozhodování v budoucnu je ovlivněno výsledkem aktuální rozhodovací situace. Proces učení je tak iterativní a rozhodnutí jsou prováděna zpravidla nedeterministicky, avšak podle daných pravidel.

Učení systému nebo agenta ve známém prostředí může spočívat ve správném nastavení parametrů zadaného modelu, který následně umožní požadovanou odezvu na zadaný vstup. Modelem může být například zadaná rovnice, neuronová síť nebo testování statistických hypotéz. Znalosti jsou v tomto případě prezentovány modelem, který je parametrizován procesem učení a následně umožní pro zadanou rozhodovací situaci získat potřebnou informaci pro rozhodování.

Pokud se agent pohybuje v neznámém prostředí, je vhodné, aby si sám byl schopen vytvořit jeden nebo více modelů pro rozhodovací situace, kterým může být vystaven. Podrobit tyto modely simulovaným vstupům a na základě výsledků z těchto modelů vybrat jistým způsobem ten, který má největší pravděpodobnost úspěchu, a tedy potenciálně největší prediktivní sílu. Následně po rozhodovací situaci vyhodnotit úspěšnost vybraného modelu a opakovaným rozhodováním opouštět nefunkční modely, parametrizovat stávající a vytvářet nové modely se schopností přesnější předpovědi.

Schopnost učit se mají jak lidé, tak v současnosti již díky rozvoji technik strojového učení (*machine learning*) do velké míry také stroje. Ačkoliv dnes počítače umožňují zpracovávat do nedávné doby nemyslitelné množství informací obrovskými rychlostmi, jejich schopnost učit se je stále relativně omezena. K většímu rozvoji technik strojového učení nedošlo okamžitě s rozšířením počítačů, ale až díky potřebě analyzovat a zpracovávat velké množství digitalizovaných informací a následně získat z těchto informací znalosti použitelné pro rozhodování.

Dostupnost informací je dnes již vyřešena pomocí počítačových sítí, ale stále chybí univerzální algoritmus, který by z libovolných informací vytvořil znalosti a tedy informace použitelné pro rozhodování. Tento algoritmus by potom provedl rozhodnutí a podle získaných výsledků vytvořil vlastní modely a parametrizoval je. Tyto modely by pak pomocí simulace chování umožnily odhalit, zda jsou přesnější pro budoucí předpověď a následně mohou být použity v další rozhodovací situaci. Největší omezení počítačů ve schopnosti učit se je tak nyní na straně programového vybavení, tedy softwaru.[1]

V tomto ohledu pochopení toho, jak funguje lidské myšlení a schopnost paměti vybavovat si informace v kontextu, provádět kombinování, kategorizaci, filtrování a uvádět informace do souvislostí, a tím tvořit znalosti může být velkým přínosem ve snaze přiblížit schopnost strojů učit se blíže k úrovni lidí či ji případně i překonat.

Pro získání reálných výsledků, které budou uplatnitelné pro modelování a simulování lidského myšlení na počítačích a následně potom umožní přenést tuto schopnost na stroje, je nutné se přesunout od kvalitativního popisu myšlenkových procesů směrem ke kvantitativnímu přístupu založenému na matematických modelech.

Tento přístup umožní popsat a modelovat lidské myšlení a jeho výstupy pomocí formálních modelů a následně s využitím počítačů provádět simulace s těmito modely. Získané výsledky přispějí k hlubšímu pochopení problematiky a mohou vést k tvorbě modelů s větší předpovědní silou v budoucnu. Touto oblastí se zabývají myšlenkové modely a architektury (*cognitive models and architectures*).

V současné době existuje několik teorií, které popisují pomocí matematických modelů myšlenkové procesy lidí a jim podobných agentů. Jednou z teorií je ACT-R, která vychází z dat získaných laboratorními i jinými experimenty. Hlavním představitelem této teorie je profesor psychologie a informatiky John R. Anderson. Teorie ACT-R je vyvíjena od roku 1990 iterativně tak, aby byla ve shodě s daty získanými z experimentů. V současné době (2016) je ve verzi ACT-R 6.0.[2]

Teorie kvantitativně popisuje princip vzniku naměřených výsledků a umožní vytvořit předpovědní statistické modely, které lze kalibrovat a jejichž výstupy lze následně validovat s daty získanými z pokusů. Dalším přístupem je pak například kognitivní architektura PSI (*Principles of Synthetic Intelligence*), která umožňuje modelovat umělou inteligenci autonomních agentů a jako jednu ze svých komponent využívá právě ACT-R.[3]

Teorie ACT-R je mimo jiné využívána pro tvorbu kognitivních tutorů, kteří umožňují díky myšlenkovému modelu lidského agenta odhadovat náročnost úloh, které tento agent musí řešit. Případně potom umožní poskytnout řešiteli patřičnou pomoc nebo zpětnou vazbu při řešení těchto úloh.

## 1.1 Cíl

V teoretické části diplomové práce je popsána terminologie, která se týká pojmů souvisejících s lidskými agenty jako učícími se systémy. Tato část se převážně zaměřuje na popis teorie ACT-R jako prostředku pro modelování myšlenkových procesů lidských agentů. Jsou zde uvedené použité vztahy, rovnice a parametry a je vysvětlen jejich význam. Následně je popsáno, jak tyto vztahy umožňují vysvětlit základní procesy paměti, jako je udržení aktuálního obsahu, vybavení si pojmu, zapomínání a efekt opakování.

S uvedenou problematikou také souvisí stručné představení nástrojů pro podporu myšlení a vytváření relací mezi pojmy. Jedná se převážně o myšlenkové mapy (*mind maps*), jejichž hlavním propagátorem je Tony Buzan.[4]

Následuje představení aplikací výsledků teorie ACT-R nebo výsledků podobných kognitivních teorií v praxi. Konkrétně se jedná o výuku cizích jazyků prostřednictvím aplikace Duolingo a o výuku matematiky pomocí projektu KhanAcademy. Zmíněné aplikace jsou stručně představeny a je uveden jejich přínos v oblasti individualizovaného vzdělávání. V závěru teoretické části je popsána možnost simulace, její význam jako metody pro zkoumání učících se systémů a nástroj NetLogo [5], který umožňuje relativně snadnou implementaci komplexních modelů založených na agentech.

Praktická část je zaměřena na návrh simulačního modelu deklarativní paměti definovaného teorií ACT-R. Je proveden odhad časového rozsahu simulace a časování částí modelu. Návrh podléhá požadavkům prostředí NetLogo, což znamená, že je navržen s využitím dostupných programových prostředků. Následuje část týkající se implementace modelu a jeho částí. Nejdříve je popsána celková struktura modelu a následně všechny jeho části a bloky, kód modelu a implementační detaily. Validace modelu je rozdělena do jednotlivých kroků, kdy nejdříve jsou validovány základní funkce modelu, popsány očekávané výsledky a poté je pokračováno funkcemi, které na ně navazují. Následná kalibrace modelu probíhá s využitím dat z ACT-R experimentů, jako jsou experimenty zapomínání, získávání praxe opakováním a „fan“ experiment.[2]

Po validaci a kalibraci modelu je navrženo několik experimentů s modelem a je tak otestováno, zda dochází k efektům, které předpovídá ACT-R pro deklarativní paměť. Tyto experimenty probíhají s různým nastavením modelu a je otestován vliv nastavení jednotlivých parametrů. Dále pak jsou popsány pozorované jevy při simulaci v různě dlouhém simulovaném časovém období. Na konci diplomové práce jsou shrnuty výsledky z experimentů a poznatky, které byly získány simulacemi.

## 1.2 Metoda

Pro realizaci praktické části diplomové práce je navržen simulační model deklarativní paměti podle teorie ACT-R [2] založený na agentech. Modul deklarativní paměti ACT-R reprezentuje jak informace, tak relace mezi nimi. Relace mezi informacemi následně umožňují také zachytit kontext, do kterého daná informace patří.

Navržený model je implementován v prostředí NetLoga [5] a je schopen popsat informace uložené v paměti a jejich aktuální stav aktivace v jakýkoliv časový okamžik. Dále tento model umožňuje v časových krocích simulovat vývoj schopnosti paměti si konkrétní informaci na požadavek vybavit, případně si s opakovaným vybavováním informace déle udržet tuto schopnost.

Model předpovídá čas vybavení dané informace a pravděpodobnost, že vybavení bude úspěšné. Aby byl model také vysvětlující, je aktuální stav informací a jejich vazeb v modelu vhodně zobrazen, což umožňuje uživateli mít přehled o stavu modelovaného systému.

Navržený model je validován a kalibrován tak, aby získané výsledky byly ve shodě s experimenty provedenými podle teorie ACT-R. Toto umožňují různé scénáře, jež budou reprezentovat typické úlohy, ve kterých různé informace spolu souvisí a jsou tedy ve vzájemných relacích. Následně je možné provádět simulační experimenty s vybraným scénářem.

Na závěr jsou prezentovány dosažené výsledky a znalosti získané jak tvorbou modelu, tak experimenty s ním. Vytvořený model je vysvětlující (*explanatory*) i předpovědní (*predictive*). Získáme jak znalosti o modelovaném systému, tak informace pro předpověď budoucího vývoje stavu systému.

# 2 Teoretická část

V rámci této části diplomové práce jsou vymezeny základní pojmy. Začátek je zaměřen na pojem učící se systém, následuje úvod do teorie ACT-R a jejího modulu deklarativní paměti. Dále je zmíněn princip zpětné vazby, její typy a význam. Je vysvětlen pojem efektivita učení v souvislosti s teorií ACT-R. Následně jsou představeny praktické aplikace výsledků teorie ACT-R, případně podobných teorií a použitá metoda simulace. Závěr teoretické části je věnován zvolenému prostředí NetLogo, použitého pro implementaci modelu v praktické části práce.

## 2.1 Učící se systém

Systémem je zpravidla množina komponent, které od sebe lze odlišit a jež jsou do jisté míry autonomní. Tyto komponenty vzhledem k ostatním komponentám v systému tedy fungují převážně samostatně. V rámci systému však dochází mezi těmito komponentami ke vzájemným interakcím, jejichž následkem může být celkově komplexní chování daného systému.[18]

Každý systém je definován svými prostorovými a časovými hranicemi. Prostorovou hranici lze stanovit jako vzdálenost, na kterou dochází k interakci mezi jednotlivými komponentami systému, a tedy k jejich vzájemnému ovlivňování. Časovou hranici potom určuje časový interval, který je nutný k tomu, aby na podnět vzniklý v jedné části systému byla schopna reagovat prostorově nejvzdálenější část systému. Tato hranice je tedy dána časovým intervalem vzájemné interakce mezi nejvzdálenějšími komponentami systému, které jsou do systému prostorově zařazeny.[18]

Systém vymezený prostorovou a časovou hranicí je obklopen a ovlivňován svým okolím. Z tohoto pohledu je možné rozlišit systémy pasivní, které pouze reagují na změny stavu okolí, a systémy aktivní, jež umožňují měnit stav svého vnímaného okolí. Dochází tedy k případné výměně informací mezi okolním prostředím a systémem, který v tomto prostředí funguje.[3]

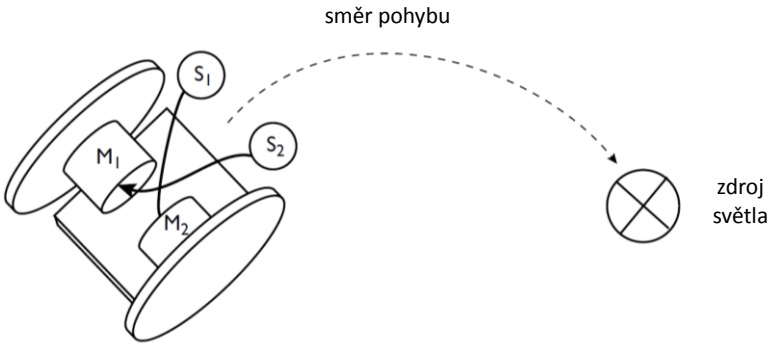
Autonomní systémy jsou takové, které mají schopnost samostatně fungovat ve svém prostředí, a u kterých dochází k interakcím s jejich okolním prostředím. Na takové systémy lze pohlížet jako na jako agenty. Multi-agentní systémy jsou pak takové, jejichž komponenty tvoří velké množství agentů, kteří svojí vzájemnou

činností a díky zpětné vazbě při interakci s prostředím formují prostředí, ve kterém fungují, a které je utvářeno ostatními agenty.[19]

Pokud uvažujeme agenta jako systém, předpokládá se u něj jistá vnitřní struktura, kterou tvoří komponenty, ze kterých se agent skládá. Ty jako celek určují možnosti a schopnosti agenta. Pro získávání informací z okolního prostředí je agent vybaven senzory (snímači), které tvoří vstupní komponenty systému agenta. Informace ze senzorů je následně zpracována řídícími komponentami, které rozhodují o další akci, kterou má agent provést. Zvolenou akci potom provádějí akční členy (*actuators*), které ovlivňují stav prostředí vnímaného pomocí senzorů agenta. Ty tak tvoří výstupní komponentu pro informace ze systému agenta.

Jako jeden z typických příkladů lze uvést jednoduché autonomní vozidlo (*Braitenberg vehicle*), jehož úkolem jako agenta je dostat se ke zdroji světla (viz Obrázek 1). K tomuto účelu je agent vybaven párem na světlo citlivých senzorů, které umožní získat z prostředí informaci o rozdílu intenzity osvětlení. Tato informace je potom přímo použita pro řízení motorů, které plní roli akčních členů a umožní pohyb vozidla. Tímto je agentovi umožněno změnit vnímaný stav okolního prostředí, protože při pohybu a změně polohy agenta dochází také ke změně intenzity osvětlení dopadajícího na senzory.[3]

Díky zpětné vazbě, kdy je stav vnímaného prostředí ovlivněn polohou agenta, který má následně možnost změnit svoji polohu a opět změnit stav vnímaného prostředí, dojde postupně v interakci agenta s tímto prostředím k tomu, že se autonomní vozidlo začne pohybovat ke zdroji světla.




**Obrázek 1: Braitenbergovo autonomní vozidlo pohybující se ke zdroji světla.[3]**

Každý systém, případně agent, má tak jistý účel nebo úkol, který se snaží plnit (*utility function*). Toho je dosahováno požadovaným celkovým chováním systému,

které je výsledkem správné funkce jednotlivých komponent systému a dále pak jejich vzájemnou souhrou a tedy časováním funkcí jednotlivých komponent.

Pokud má mít agent nebo systém schopnost učit se, znamená to, že jeho výstup nebude záviset pouze na aktuální hodnotě vstupu, ale i na hodnotě vnitřního stavu agenta. Tento vnitřní stav je reprezentován daty, která agent získal jistým způsobem z předchozích vstupů a z minulé interakce s prostředím. Vnitřní stav je závislý na posloupnosti akcí a jejich časování, které ke vzniku tohoto vnitřního stavu vedly a tedy jej vytvořily.

Data popisující vnitřní stav agenta spolu s daty ze vstupu jsou potom u agenta použity v komponentách řídících jeho chování při rozhodování. Tato data potom určují, jakou další akci má agent vykonat, a tím i pozorované výstupy jeho akčních členů.[3] Vnitřní stav agenta je uchováván v paměti agenta, která je fyzickým nosičem těchto dat. Podle typu agenta se může jednat o různé druhy paměti. U počítačů se jedná například o digitální polovodičovou paměť, u strojů pak o mechanickou paměť materiálu, u biologických systémů o sílu vazeb mezi neurony v neuronové síti.

Všechny uvedené typy pamětí, ačkoliv značně odlišné, vždy uchovávají informaci o předchozím vystavení systému nebo agenta podmínkám vnějšího prostředí. Stav prostředí v podobě hodnot veličin představuje vstupní informace do systému nebo agenta. Informace uložené v paměti v daný okamžik potom reprezentují stav systému nebo agenta, který je ve shodě s jeho vystavením konkrétním hodnotám prostředí ze vstupů tohoto agenta či systému v minulosti.

## 2.2 Teorie ACT-R

Jedná se o teorii popisující matematickými modely myšlenkové procesy lidských agentů. Zkratka ACT-R označuje racionální adaptivní řízení myšlenek (*Adaptive Control of Thought - Rational*) a byla vyvinuta na základě série předchozích, stále se zpřesňujících modelů psychologa a informatika Johna R. Andersona.

Její historii lze vysledovat až k původnímu modelu HAM (*Human Associative Memory*), který popsali v roce 1973 John R. Anderson a Gordon Bower. Později byl tento model rozšířen, čímž vznikla první verze ACT teorie. V roce 1998 pak byla představena první komplexní verze ACT-R, kterou navrhl John R. Anderson společně s Christianem Lebierem a dalšími autory.[16]

V současné době je aktuální verze ACT-R 6.0, jejímž cílem je vytvořit rámec, který lze použít pro tvorbu komplexních a relativně přesných modelů popisujících myšlenkové procesy lidí při řešení aritmetických, logických a jiných podobných typů úloh. Následně lze předpovědět budoucí vývoj těchto procesů. Tato teorie byla postupně rozpracovávána a zdokonalována tak, aby pomohla vytvořit přesnější modely lépe popisující údaje získané měřením v reálných experimentech.[2]

Teorie ACT-R umožňuje popsat lidské kognitivní procesy v různých typech řešených úloh pomocí modelů, které jsou postaveny na znalostech o lidském myšlení vycházejících z empirických údajů. To umožňuje přijímat předpoklady pro daný typ úlohy, kterou lidský agent řeší. Na základě tohoto předpokladu lze vytvořit jeho matematický model s předpovědním charakterem.

Výsledky modelů lze porovnat s výsledky získanými v experimentech, které se zaměřují na měření veličin jako čas nutný pro vykonání úlohy či přesnost v úloze. V novějších verzích také porovnání dat získaných prostřednictvím lékařských přístrojů magnetické rezonance (MRI – *Magnetic Resonance Imaging*) s výsledky předpovědního modelu.[2]

ACT-R jako systémová teorie lidského myšlení popisuje několik typů modulů, přičemž každý se specializuje na jistý typ funkce. Při zpracovávání úkolu potom pracují tyto moduly s různými typy informací a jsou propojeny tak, aby vytvořily celkový pohled na myšlenkové procesy a jejich výstupy. Z tohoto pohledu ACT-R definuje moduly vnímání (*sensors*) a pohybu (*actuators*). Do modulů zajišťujících a ovlivňujících proces řízení pak patří modul cílů (*goal module*) a moduly deklarativní a procedurální paměti. Následně jsou funkce těchto modulů mapovány na různých částech mozku, které souvisejí s vykonáváním funkcí těchto modulů a jejichž aktivita může být sledována pomocí přístrojů MRI.[15]

Hlavní a základní oblastí, kterou ACT-R pokrývá, je matematický model toho, jak si paměť u lidí vybavuje dříve zapamatované fakty (*chunks*). Jak vybavení těchto faktických informací souvisí s jinými, se kterými jsou tyto informace v relaci. Dále také popisuje, jak je ovlivněna úroveň aktivace těchto informací v závislosti na předchozích zkušenostech a vazbách k řešené úloze. Tento mechanismus je zvláště důležitý v oblasti matematických výpočtů prováděných lidmi, kde je vyžadována přesnost, rychlost vykonávání a pochopení logických vazeb mezi koncepty.[16]

Jednotlivé moduly si v ACT-R předávají bloky informací (*chunks*) pomocí dočasných pamětí (*buffers*). V jakýkoliv okamžik může být vybráno jedno z pravidel (*production*). Tento výběr může být proveden jako reakce na vstup, případně na základě požadavků aktuálně probíhajících procesů v jiných modulech.

Sub-symbolické aktivační procesy slouží k podvědomému výběru pravidel prováděných v různých modulech ACT-R. Tato pravidla potom umožňují například získávání dříve naučených faktů z deklarativní paměti nebo výběr pravidel pro zpracování.

Většina výsledků učení poté zahrnuje správné nastavení těchto sub-symbolických aktivačních procesů. To znamená, že člověk je schopen vybavit si požadované informace v kontextu řešené úlohy, případně způsob, jakým tyto informace zpřístupnit pomocí naučených pravidel (*productions*).[15]

### 2.2.1 Modul deklarativní paměti

Jedná se o jeden ze základních modulů teorie ACT-R a popisuje způsob, jakým jsou v paměti uchovávány informace faktické povahy a jak jsou informace z tohoto modulu v případě potřeby získávány. Fakty jsou v deklarativní paměti podle ACT-R reprezentovány jako bloky (*chunks*).[15]

Jedná se například o informaci `5 + 3 = 8`, se kterou se agent dříve setkal, byla pro něj užitečná a může být opět využita v budoucnu. Jednodušeji se může jednat například o bloky `1 další 2`, `5 další 6`, `9 předchozí 8` a tak dále. Z jednodušších příkladů si lze povšimnout, že tyto bloky používají koncepty jako `1`, `2`, `5`, `6` a další. Tyto koncepty jsou ze své povahy pevně dané a neměnitelné (*immutable*).[16]

Agent musí být tedy schopen jednoznačně identifikovat a rozlišovat pevně dané koncepty, které tak mohou být v paměti reprezentovány jako bloky. K těmto účelům pak paměť používá kombinaci mechanismů abstrakce, která slouží k rozpoznání těchto konceptů pomocí vzorů (*pattern matching*), a klasifikace, které získané informace asociují s předem známými koncepty nebo bloky. U lidí jsou tyto koncepty formovány na sub-symbolické úrovni opakovaným vystavením informacím, které používají tyto koncepty a jsou s nimi v relaci.

Tvorba základních abstraktních neměnných konceptů, jako jsou například celá čísla, a následně pravidel pro manipulaci s nimi je u lidí zdlouhavý proces. Tento

proces je založen jak na aktivaci v závislosti na vstupu ze senzorů, tak aktivaci aktuálního stavu paměti a díky schopnosti lidské paměti zapomínat musí být formován opakovaně a dlouhodobě.

Zvládnutí tohoto procesu zabere dětem mnoho let formálního učení. Schopnost mozku přizpůsobit se téměř jakémukoliv typu informace a pracovat s ní nedeterministicky pak činí člověku při řešení úloh aritmetického typu, kde je vyžadovaná přesnost, jednoznačnost a logická návaznost, velké obtíže, ačkoliv stroje s tímto typem úloh nemají potíže a zvládají ho velice jednoduše.[16]

Formování těchto konceptů je iterativní proces a lze při něm pozorovat jistou úroveň emergence, což znamená, že systém je nastaven tak, že i při náhodných počátečních podmínkách dojde časem za působení informací z vnějšího prostředí k jeho přechodu do požadovaného stavu nebo k chování.

Blok informace (*chunk*) podle ACT-R pak propojuje několik sub-symbolických konceptů (jiných bloků) a tvoří tak relaci mezi dvěma nebo více bloky. Blok si lze představit jako záznam informace, kterou daný blok nese a zároveň se odkazuje na další bloky, jež mohou sloužit jako kontextová nápověda při vybavování bloku z paměti.[15]

Například blok `1 další 2` představuje relaci typu `další` mezi koncepty `1` a `2`. Pokud se podíváme na blok `5 + 3 = 8`, lze jej uvažovat jako blok, který je ve vztahu ke konceptům `5`, `3` a `8`. To platí v případě, pokud neuvažujeme u bloku jeho vnitřní strukturu.[15]

Případně by bylo možné do konceptů zahrnout také `+` a `=` za předpokladu, že tyto koncepty jedinec zná a dokáže je identifikovat. Výskyt konceptů pak bude úměrný množství bloků zakódovaných v deklarativní paměti. Například pokud je agent vystaven pouze faktům obsahujícím `=`, neexistuje potřeba rozlišovat koncept `rovnosti`. Potřeba kategorizace a struktury vyvstává až v okamžiku, kdy je co strukturovat, to znamená v případě, pokud je množství faktů dostatečně velké.

Pokud se však zaměříme na strukturu bloku `5 + 3 = 8`, lze jej uvažovat jako strom, který dává do relace `=` koncepty `8` a `5 + 3`. Blok `5 + 3` zde vyžaduje vyhodnocení a následně pro získání odpovědi musí být vyhodnocena i relace `=`. Modul deklarativní paměti podle ACT-R však uchovává pouze bloky faktů (*chunks*) jako jednotlivé záznamy a jejich vztah ke konceptům, se kterými daný blok souvisí.[2]

K vyhodnocení významu bloku (zde vyhodnocení výrazu) je nutné provést jeho analýzu a vyhodnocení v procedurálním modulu správně zvoleným pravidlem (*production*).[15] Možnost strukturovat obsah bloku také ukazuje na schopnost, jak agent interpretuje dané fakty, což mu umožní vytvořit vzájemné vazby mezi nimi.

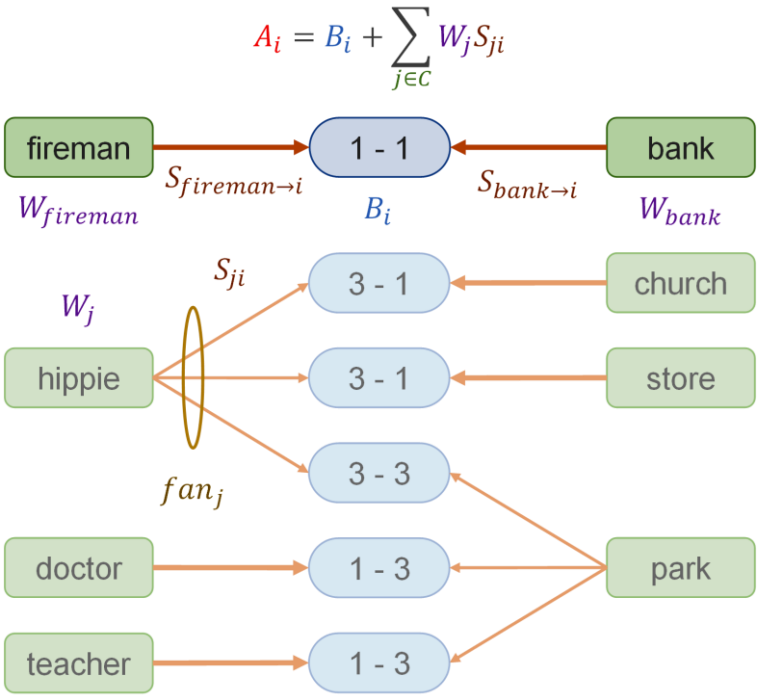
To, že má agent znalost faktu `5 + 3 = 8`, ještě neznamená, že chápe jeho význam. K pochopení významu může dojít až poté, co agent získá dostatečné množství faktů, a tyto fakty jsou uspořádány do jisté struktury. Z pohledu paměti potom dochází k restrukturalizaci obsahu, která umožní stejné množství faktů zakódovat efektivněji a následně v nich rychleji vyhledávat. Tvorba trvalých nebo dočasných kategorií nebo konceptů, se kterými je pak daný blok v relaci, je potom logickým důsledkem tohoto procesu. Restrukturalizace může následně učinit některé fakty dostupné snadněji (učení opakováním) a jiné hůře (zapomínání).

### 2.2.2 Fan experiment

John R. Anderson provedl v roce 1974 desetidenní experiment, ve kterém měli účastníci za úkol rozpoznávat věty. V těchto větách byly do relací dávány osoby a místa a tyto relace potom z pohledu teorie ACT-R tvoří bloky informací (*chunks*), které si měli účastníci za úkol vybavit a rozpoznat. Například se jednalo o věty typu: *„A hippie was in the park“*.[15] Účastník experimentu měl za úkol určit, zda předložená věta byla přítomna v sadě, kterou se předem učil či nikoliv.

Osoby a místa potom sloužily v předložených větách při vybavování jako kontextová nápověda, která může usnadnit a urychlit vybavení konkrétního bloku, a tím ovlivnit celkový čas rozhodování. Fan číslo potom vyjadřuje, v kolika blocích je použita daná kontextová nápověda, která může být sama představována blokem informace o osobě nebo místu.

V experimentu mohla být jedna osoba použita v jedné nebo ve třech větách a jedno místo pak také v jedné nebo ve třech větách. Tím vznikly následující typy bloků z pohledu kontextové nápovědy, a to 1-1 (jedna osoba na jednom místě), 1-3 (jedna osoba na třech místech), 3-1 (tři osoby na jednom místě) a 3-3 (tři osoby na třech místech). Každá věta jako blok tak měla svoji základní úroveň aktivace $B_i$ a aktivaci, kterou získala z předložené kontextové nápovědy v podobě věty obsahující osobu a místo obsažené v dříve naučených blocích. Součtem těchto úrovní aktivací pak každý blok získal svoji celkovou úroveň aktivace $A_i$. Vzniklou situaci pomůže ilustrovat následující obrázek (viz Obrázek 2) [15].



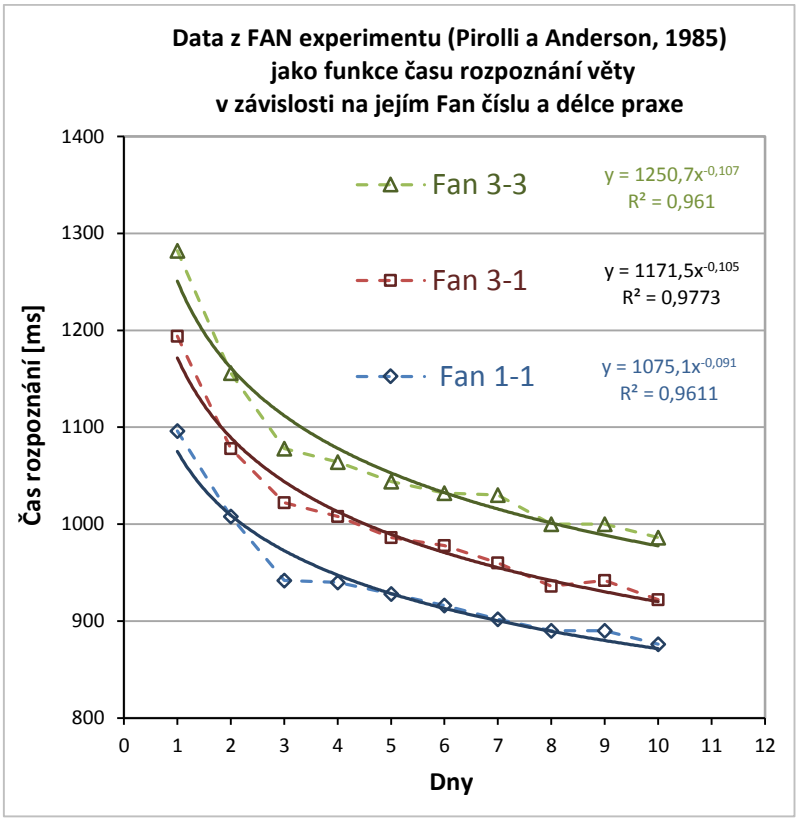

**Obrázek 2: Schéma „Fan experimentu“, vybavované bloky jsou modré, bloky kontextové nápovědy zelené a šipky označují vazby mezi bloky a jejich intenzitu. Podle teorie ACT-R blok 1-1 získá největší kontextovou aktivaci a blok 3-3 nejmenší kontextovou aktivaci.**

V experimentu byl potom měřen čas, jak dlouho trvalo rozpoznání, zda byl představený blok v sadě, kterou se účastník učil či nikoliv. Z výsledků potom bylo patrné, že bloky používající jako kontextovou nápovědu bloky s malým Fan číslem měly neustále výhodu v podobě menších časů nutných pro vybavení oproti blokům, které používaly bloky s vyšším Fan číslem ve funkci kontextové nápovědy. Tento rozdíl se zmenšoval, avšak zůstával znatelný i po deseti dnech.[15]

Naměřené časy nutné pro rozpoznání bloku jako funkce Fan čísla a zároveň v závislosti na počtu dní praxe jsou vidět v níže uvedeném grafu (viz Graf 1). Pro úplnost jsou uvedeny také použité hodnoty (viz Tabulka 1). Aproximace naměřených hodnot byla provedena pomocí mocninné funkce, ačkoliv by bylo možné použít i funkci přirozeného logaritmu s dokonce vyšší mírou shody. Tato funkce však není vhodná z důvodu, že pro vysoké počty dní praxe by mohly vyjít i záporné časy vybavení, což by nedávalo z pohledu pozorované skutečnosti smysl. U uvedených mocninných funkcí se s vysokým počtem dní hodnota času asymptoticky přibližuje k nule, ale nikdy nebude záporná pro kladný čas vybavení, což je ve shodě jak s naměřenými hodnotami v dlouhodobých experimentech, tak s obecným předpokladem.[20]

| Den | Fan 1-1 [ ms ] | Fan 3-1 [ ms ] | Fan 3-3 [ ms ] |
|---|---|---|---|
| 1 | 1096 | 1194 | 1282 |
| 2 | 1008 | 1073 | 1156 |
| 3 | 942 | 1022 | 1078 |
| 4 | 902 | 1008 | 1064 |
| 5 | 928 | 986 | 1044 |
| 6 | 916 | 978 | 1024 |
| 7 | 902 | 960 | 1030 |
| 8 | 890 | 936 | 1000 |
| 9 | 890 | 942 | 1000 |
| 10 | 876 | 922 | 986 |

**Tabulka 1: Hodnoty získané z Fan experimentu.[15]**




**Graf 1: Vizualizace dat získaných Fan experimentem (vlastní zpracování dle [15]).**

Zajímavostí v uvedených hodnotách (viz Graf 1) je výrazný pokles v prvních třech dnech, který může souviset s tím, že měřený čas je kombinací dvou časů a tedy i dvou efektů. Jedním z nich je vybavení bloku a druhým čas potřebný pro kódování požadavku a zadání odpovědi. Uvedené hodnoty jsou zprůměrovány od všech účastníků experimentu pro každý den.

### 2.2.3 Čas vybavení bloku

Čas, po který trvá vybavení bloku, je základní fyzikální veličinou, kterou je možné objektivně měřit v experimentech prováděných s lidmi. Podle Johna Andersona lze tento čas rozpoznání bloku ve Fan experimentu určit následovně: [2]

$$T_r(i) = I_T + T_i$$

**Rovnice 1: Čas $T_r(i)$ nutný pro rozpoznání bloku $i$ podle Johna Andersona.[2]**

Čas $T_r(i)$ vyjadřuje čas potřebný pro vybavení bloku $i$ (*recognition time*). Hodnota parametru $I_T$ potom vyjadřuje čas potřebný pro kódování požadavku na vybavení a čas potřebný pro zadání odpovědi (*intercept time*). Parametr $T_r(i)$ následně vyjadřuje čas vlastního vybavování bloku $i$.

Pro označení časů se používá velké písmeno T, což naznačuje, že se jedná o celkovou délku periody daného procesu. Hodnota $I_T$ se liší v závislosti na typu úlohy, pro kódování jednoduchého požadavku na akci a následného stisknutí klávesy se může jednat i o stovky milisekund (Anderson ve svém *fan* experimentu uvádí hodnotu $I_T = 597$ ms).[15] Pro čas vlastního vybavení bloku se používá následující vztah:

$$T_i = F \cdot e^{-A_i}$$

**Rovnice 2: Čas $T_i$ nutný pro vybavení bloku $i$ podle Johna Andersona.[2]**

V uvedeném vztahu $T_i$ vyjadřuje, jak dlouho bude trvat vlastní vybavení bloku $i$. Tento čas je závislý na aktuální celkové úrovni aktivace bloku $A_i$, jejíž úroveň se praxí a opakováním zvyšuje a při nevyužívání bloku časem degraduje. Parametr $F$ nastavuje měřítko odezvy paměti (*latency scale factor*).[15]

Vztah, který navrhuje John Anderson, má však jeden zásadní nedostatek pro dlouhodobý odhad vývoje aktivační úrovně bloků. Pokud budeme chtít určit úroveň aktivace bloku $A_i$ z času získaného v experimentu $T_r(i)$, je nutné navržený vztah $T_r(i) = I_T + F \cdot e^{-A_i}$ upravit na základě tohoto požadavku následovně:

$$A_i = \ln\left(\frac{F}{T_r(i) - I_T}\right)$$

**Rovnice 3: Výpočet aktivační úrovně $A_i$ z času naměřeného v experimentu $T_r(i)$ podle Johna Andersona.[2]**

Problém vzniká v okamžiku, kdy hodnota $T_r(i)$ asymptoticky klesá téměř k nulovému času (v reálných situacích k času přibližně 100 ms pro úlohy s počtem opakování v řádu 100000 [20]). V okamžiku, kdy $T_r(i)$ poklesne pod stanovenou hodnotu $I_T$, dojde k selhání modelu, protože v argumentu logaritmu bude použita záporná hodnota.

Uvedený model tak lze použít pouze pro malý rozsah dní (cca do 50), přičemž experiment, který provedl Anderson, trval deset dní a pozorovaná odchylka tedy nebyla příliš velká. Vztah nelze použít pro dlouhodobé předpovědi. Důvodem je, že u všech třech procesů, jimiž jsou procesy kódování bloku, vybavování bloku a zadání odpovědi, probíhá zrychlování odezvy v závislosti na počtu řešených úloh.[20] Pozorovaný a měřený čas $T_r(i)$ je potom součtem časů těchto třech oddělených a na sebe navázaných procesů.

Aby bylo možné vytvořit model s dlouhodobou schopností předpovědi, musí dostatečně přesně aproximovat jak rozsah 10 dní, tak 100 dní, 1000 dní atd. Čím bude

celkový čas vybavení $T_r(i)$ nižší, o to větší musí být odpovídající úroveň aktivace $A_i$ pro celý proces rozpoznávání daného typu bloku (fan 1-1, fan 3-1, fan 3-3). Z vlastních experimentů, jak mapovat hodnotu $T_r(i)$ na hodnotu $A_i$, a podle návrhu Christiana Lebiereho je možné požít následující vztah:

$$T_r(i) = F \cdot e^{-f \cdot A_i}$$

**Rovnice 4: Čas $T_r(i)$ nutný pro rozpoznání bloku $i$ v závislosti na úrovni aktivace bloku $A_i$ podle Christiana Lebiereho.[16]**

Vztah upravíme pro účely výpočtu úrovně aktivace $A_i$ z času měření $T_r(i)$.

$$A_i = \ln\left(F \cdot \left(T_r(i)\right)^{-1}\right) \cdot f^{-1}$$

**Rovnice 5: Výpočet úrovně aktivace bloku $A_i$ podle času nutného pro jeho rozpoznání $T_r(i)$.[16]**

Výhodou tohoto vztahu je fakt, že čas vybavení bloku $T_r(i)$ bude vždy kladné číslo a pro kladné koeficienty $F$ (*time scaling constant*) a $f$ (*activation scaling constant*) bude aktivace $A_i$ s klesajícím časem $T_r(i)$ vždy růst, což je v souladu s předpokladem, že vyšší aktivace bloku má za následek kratší čas vybavení.

### 2.2.4 Pravděpodobnost vybavení bloku

Pravděpodobnost, že se podaří daný blok z paměti získat, závisí podobně jako čas vybavení na celkové úrovni aktivace bloku $A_i$. Existují dva základní způsoby, jak lze tuto pravděpodobnost odhadnout. V aktivačních úrovních je neustále přítomný jistý šum, a tak si můžeme vybavit i blok informace, který vůbec nesouvisí s řešenou úlohou, nicméně tato pravděpodobnost je nízká s přihlédnutím k celkové úrovni aktivace tohoto bloku v relativním poměru k aktivacím ostatních bloků a velikosti aktivačního šumu. Průměrnou pravděpodobnost vybavení bloku lze pak určit podle ACT-R následujícím vztahem.[15]

$$P_r(i) = \frac{1}{1 + e^{-\left(\frac{A_i - \tau}{s}\right)}}$$

**Rovnice 6: Průměrná hodnota pravděpodobnosti vybavení $P_r(i)$ bloku $i$.**

Bloky si lze vybavit pouze v případě, že je jejich celková úroveň aktivace nad hranicí, kterou udává parametr $\tau$ (*threshold*). Protože aktivační hodnoty podléhají šumu, v reálném případě existuje pouze jistá pravděpodobnost, že daný blok bude vybaven. Parametr $s$ slouží k řízení úrovni šumu v aktivačních hodnotách a je typicky nastaven na hodnotu v rozsahu od 0,4 [15] do 0,8 [2]. Hodnota hranice pro vybavení $\tau$ je typicky kolem hodnoty 1 až 1,5 [2].

Pokud budeme v simulačním modelu vyžadovat přímo práci s aktivačním šumem, je možné k vypočtené aktivační úrovni $A_i$ podle aktivační rovnice

(viz Rovnice 11) přidat náhodnou hodnotu šumu. Tyto náhodné hodnoty mají Gaussovo rozdělení pravděpodobnosti s nulovou střední hodnotou a standardní odchylkou $\sigma$. Pro výpočet standardní odchylky $\sigma$ pro aktivační šum navrhuje Christian Lebiere následující vztah.[16]

$$\sigma = \frac{s \cdot \tau}{\sqrt{3}}$$

**Rovnice 7: Výpočet standardní odchylky $\sigma$ šumu pro aktivační úrovně $A_i$ aktivací bloků.[16]**

Zde je vhodné poznamenat, že násobek $s$ a $\tau$ musí být vždy kladné číslo (nelze pracovat se zápornou standardní odchylkou šumu). Jak parametr $s$, který řídí úroveň šumu, tak parametr $\tau$ udávající limit pro vybavení musí být kladné. Vzhledem k hodnotám získaných z reálných experimentů není problém tuto podmínku dodržet (viz typické hodnoty $s$ a $\tau$). Úroveň aktivace zahrnující šum lze získat níže uvedeným vztahem.

$$A_{i(+noise)} = A_i + N(0, \sigma)$$

**Rovnice 8: Úroveň aktivace bloku $i$ zahrnující aktivační šum.**

Pro výpočet pravděpodobnosti vybavení bloku s uvažováním šumu $P_{i(+noise)}$ v aktivačních hodnotách pak použijeme verzi předchozího vztahu (viz Rovnice 6).

$$P_{r(+noise)}(i) = \frac{1}{1 + e^{-\frac{(A_{i(+noise)} - \tau)}{s}}}$$

**Rovnice 9: Hodnota pravděpodobnosti vybavení bloku $i$ při zahrnutí aktivačního šumu.**

Pokud žádný z bloků nemá dostatečnou úroveň aktivace pro vybavení, což znamená, že úroveň aktivace je menší než $\tau$, dojde k selhání procesu vybavování při použití daného pravidla (*production*) a je vybrán jiný typ pravidla pro vybavování.[16]

Příkladem může být situace, kdy si ani po dostatečně dlouhé době nemůže modelovaný agent vzpomenout na jistou informaci. V tomto případě je tedy vybráno jiné pravidlo, pomocí kterého lze danou informaci získat. Vlastní informaci si nedokážeme vybavit, ale známe způsob, jak se k dané informaci dostat (například hledání v knize, výpočet výrazu krok za krokem, představování si prožité události). Tato vlastnost je důležitá pro schopnost paměti filtrovat informace, které nejsou důležité pro aktuálně probíhající úlohu a kontext.

### 2.2.5 Výběr z vybavených bloků

Pokud je pamětí vybaveno více bloků, je nutné z nich vybrat jeden, který splňuje požadavky dané úlohy. Aby bylo možné si bloky vybavit, musí být podle

ACT-R jejich aktivační úroveň nad hranicí pro vybavení $\tau$. Proces vybavování potřebných bloků zabírá jistý celkový čas na základě úrovní aktivací $A_i$ daných bloků. Delší čas může díky šumu v aktivačních hodnotách vést i k vybavení bloků s menší úrovní aktivace, než je $\tau$, protože se tyto bloky dočasně ocitnou nad tímto limitem.[2]

Teorie ACT-R popisuje dva režimy pro výběr bloků (*matching mode*). Jedná se o mód přesné shody (*exact*), kdy jsou srovnávány pouze úrovně aktivací bloků. Druhým módem je částečná shoda (*partial*). Nejdříve bude popsán implementovaný mód přesné shody.

Mód přesné shody (*exact*) popisuje situaci, kdy vybavovaný blok musí přesně splňovat požadované podmínky. V takovém případě lze provést výběr podle vztahu, který navrhuje Christian Lebiere.[16]

$$P(i) = \frac{e^{A_i/t}}{\sum_j e^{A_j/t}}$$

**Rovnice 10: Pravděpodobnost výběru bloku *i* z *j* bloků, které jsou nad limitem $\tau$ nutným pro vybavení v módu přesné shody.**

Uvedený vztah vyjadřuje pravděpodobnost volby bloku $i$ z $j$ vybavených bloků a je popsáno Boltzmannovým rozdělením pravděpodobnosti. Parametr $t$ má potom podle Christiana Lebiereho hodnotu $t = \sqrt{2} \cdot s$.

V módu částečné shody (*partial*) je zvažován každý blok správného typu (to znamená všechny bloky související s danými kontextovými nápovědami). Pokud daný blok neplní požadované podmínky, je od jeho úrovně aktivace odečtena jistá penalizační úroveň. Místo úrovně aktivace se pak v ostatních výpočtech používá tato snížená úroveň aktivace $M_p(i)$, která vyjadřuje úroveň shody (*match*) daného bloku s požadovanými podmínkami. Tento mód je výpočetně složitý, a proto zde nebude uváděn. Jeho základní popis lze však nalézt v uvedeném zdroji.[16]

### 2.2.6 Úroveň aktivace bloku

Většina výše uvedených vztahů používá pro získání kvantitativně měřených výsledků v experimentech pomocí navrženého modelu ACT-R aktivační úroveň bloku $A_i$, která hraje zásadní roli pro výpočet pravděpodobnosti vybavení $P_r(i)$, výběru $P(i)$ a času vybavení bloku $T_r(i)$ a následnou předpověď budoucího vývoje těchto veličin. Nyní bude popsáno, jak je tato aktivační úroveň bloku získána.

Jako první bude představena aktivační rovnice bloku běžná v aktivačních teoriích a používaná i v různých oblastech (například u neuronových sítí). Byl přijat předpoklad, že tato rovnice umožní modelovat výsledky procesů, které probíhají na

úrovni neuronů a z nich složených sítí, jejichž chování lze popsat obdobnou rovnicí.[2]

$$A_i = B_i + \sum_{j \in C} W_j S_{ji}$$

**Rovnice 11: Celková úroveň aktivace $A_i$ bloku $i$.**

Bloky v deklarativní paměti mají celkovou aktivační úroveň $A_i$. Tato úroveň následně určuje jak pravděpodobnost, že dojde k vybavení $P_r(i)$ daného bloku $i$, tak pravděpodobnost $P(i)$, že bude tento blok vybrán, a následně i čas $T_r(i)$, který bude potřebný k vybavení bloku z paměti.

Kontext, ve kterém dochází k vybavení bloku $i$, je značen $C$ a je definován jako množina bloků $j$, které jsou aktuálně v dočasné paměti (*buffer*) a jsou v relaci s blokem $i$. Základní úroveň aktivace bloku $B_i$ (*base level activation*) potom vyjadřuje úroveň aktivace bloku $i$ získanou díky používání bloku $i$ v minulosti. Koeficient $W_j$ vyjadřuje pozornost, která je věnována kontextové nápovědě získané díky přítomnosti bloku $j$ při vybavování bloku $i$. Aktivační úroveň asociace $S_{ji}$ vyjadřuje sílu asociace od bloku kontextové nápovědy $j$ k bloku $i$, který se snažíme vybavit.[2]

Uvedená rovnice (viz Rovnice 11) je standardní neurální aktivační rovnice, kde úroveň aktivace neuronu $i$ je určena základní úrovní aktivace a kontextovou úrovní aktivace, kterou získá neuron na vstupech od neuronů $j$ v závislosti na síle předchozí vazby a úrovni na vstupu. Uvedený vztah umožní vybavit si bloky, které budou nejpravděpodobněji potřeba v daném kontextu $C$.[2]

Pro zjednodušení lze rovnici (viz Rovnice 11) pro výpočet základní úrovně aktivace $A_i$ bloku $i$ rozložit na dvě části. Na základní úroveň aktivace $B_i$ bloku $i$ získanou opakovaným vybavením bloku $i$ v minulosti (předchozí učení a potřeba vybavování bloku) a na kontextovou úroveň aktivace $C_i$ (*context level activation*) bloku $i$, která poskytuje vstup ve formě kontextové nápovědy (*cue*) pro blok $i$ při jeho vybavování (viz Rovnice 12).

$$A_i = B_i + C_i$$

**Rovnice 12: Zjednodušená rovnice celkové úrovně aktivace $A_i$ bloku $i$.**

Základní úroveň aktivace $B_i$ popisuje aktuální stav naučení bloku $i$ vyjadřující jeho užitečnost v minulosti, tedy nepřímo aktuální stav paměti z pohledu vybavení tohoto bloku bez kontextové nápovědy.

Kontextová úroveň aktivace $C_i$ vyjadřuje stav naučené asociace mezi blokem $i$ a souvisejícími bloky $j$ a $W_j$ váhu, která je této asociaci přisuzována, a tím tedy kontext (vstup), v rámci kterého je blok vybavován.[2]

### 2.2.7 Sub-symbolické učení

Základní úroveň aktivace bloků $B_i$ je zvýšena těm blokům, které jsou používány opakovaně, což naznačuje, že byly tyto bloky opakovaně užitečné v minulosti. Aktuální úroveň základní hodnoty aktivace je také vyšší, pokud byly dané bloky použity nedávno, a tím určuje, že mohou být požadovány i v blízké budoucnosti. Uvedená rovnice základní úrovně aktivace je založena na mnoha empirických výsledcích získaných v experimentech.[17]

$$B_i = \ln \sum_{u=1}^{n} t_u{}^{-d}$$

**Rovnice 13: Diskrétní výpočet základní úrovně aktivace $B_i$ bloku $i$.**

Rovnice výpočtu základní úrovně aktivace bloku $B_i$ vyjadřuje situaci, že blok $i$ byl vyžadován $n$-krát v minulosti v časech $t_1, t_2, \dots, t_n$. Tyto časy jsou pak vztaženy k aktuálnímu času. Například pokud došlo k prvnímu užití bloku v čase $t_u = 10$ a nyní je čas $t_p = 15$, pak bude velikost hodnoty $t_1 = t_p - t_u = 5$. Stejný vztah platí i pro ostatní časy užití bloku $t_u$ pro $u$ od 1 do $n$.[17]

Časové jednotky se nikde neuvádí, nicméně experimenty probíhají nejčastěji po jednotlivých dnech, takže lze předpokládat, že všechny hodnoty souvisejících parametrů jsou vztaženy k základní časové jednotce jednoho dne. Pokud bude hodnota zisku aktivace při užití proměnná, lze toto vyjádřit pomocí následujícího vztahu.

$$b_u = b_u(t_u) \cdot t^{-d}$$

**Rovnice 14: Výpočet hodnoty základní aktivace užití $b_u$ podle původní hodnoty $b_u(t_u)$ aktivace v čase užití $t_u$ a času $t$ od vzniku tohoto užití.**

Každá hodnota $b_u$ představuje absolutní hodnotu aktivace vzniklou daným užitím bloku $u$. Její velikost je tak vždy větší než 0 (doba $t$ od času užití $t_u$ může být pouze kladná) a tato hodnota aktivace degraduje rychlostí $d$, která udává míru degradace paměti. Velikost počáteční hodnoty aktivace v čase užití je v ACT-R nastavena na hodnotu 1, obecně lze vztah doplnit o hodnotu koeficientu $b_u(t_u)$, která vyjadřuje hodnotu aktivace užití na konci časového intervalu $t_u$, kdy dané užití vzniklo.

$$B_i = \ln b_i$$

**Rovnice 15: Převod hodnoty aktivace $b_i$ na úroveň aktivace $B_i$.**

Součtem všech hodnot aktivací jednotlivých užití $b_j$ pak získáme celkovou hodnotu užití bloku $b_i = \sum_{u=1}^{n} b_u$. Následně platí vztah, že pro získání úrovně aktivace užití bloku je nutné hodnotu $b_i$ mapovat pomocí funkce přirozeného logaritmu na úroveň základní aktivace (viz Rovnice 15). Z tohoto důvodu je nutné zásadně rozlišovat, zda pracujeme s hodnotou aktivace $b_i$, která je vždy kladná a nenulová, nebo s úrovní aktivace $B_i$, která může být kladná i záporná. Díky uvedenému vztahu je pak možné jakoukoliv úroveň aktivace převést na hodnotu aktivace a naopak.

Pokud jsou užití bloku rovnoměrně rozloženy v čase, navrhuje Christian Lebiere vztah, který tuto skutečnost analyticky popisuje a umožní sice snadnější a efektivnější, ale méně přesné výpočty (viz Rovnice 16). Pro účely diskrétní simulace je však vhodnější předchozí vztah (viz Rovnice 13) a v případě, že dojde k rovnoměrnému využití bloku, by potom měl výsledek odpovídat vztahu, který navrhuje Lebiere.[16]

$$B_i = \ln\left(\frac{n \cdot L^{-d}}{1-d}\right)$$

**Rovnice 16: Analytický výpočet základní úrovně aktivace $B_i$ bloku $i$ za předpokladu jeho rovnoměrného využívání v minulosti.[16]**

V uvedeném vztahu vyjadřuje parametr $L$ celkovou dobu existence bloku (kolik času uběhlo od jeho vytvoření) a $n$ vyjadřuje, kolikrát byl blok využit.

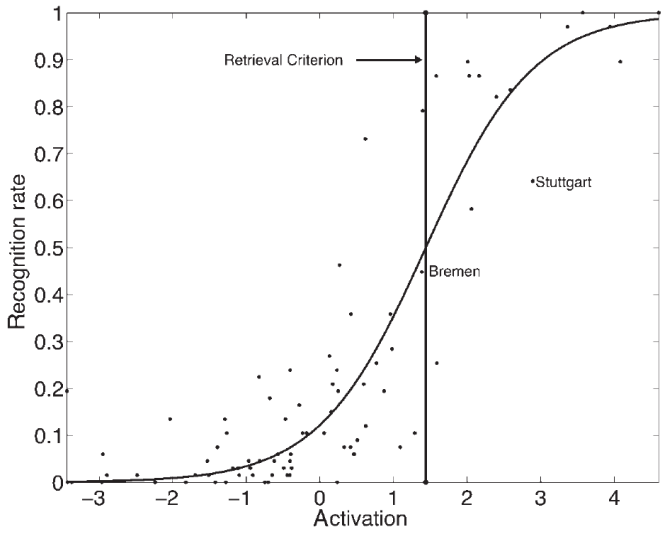
Podobný vztah navrhuje Christian Lebiere také pro výpočet úrovní aktivace relací, aby tyto úrovně odrážely naučenou asociaci mezi bloky $j$ a $i$. Jedná se však o velice komplexní analytický výpočet, který popisuje neméně komplexní systém, kterými relace mezi bloky zajisté jsou. Přesto se však jedná pouze o průměrný odhad dané síly relace.

### 2.2.8 Míra degradace paměti

Tento parametr udává faktor degradace paměti. Vyšší hodnota vyjadřuje rychlejší degradaci paměti a naopak. V ACT-R komunitě se pro velký rozsah modelů často používá hodnota $d = 0{,}5$. Při pokusech o modelování velmi dlouhodobé paměti se pak používá hodnota menší, což naznačuje, že míra degradace paměti $d$ nemusí být konstantní v čase.[2]

John R. Anderson ve své publikaci popisuje experiment z roku 2002, jehož cílem bylo testovat heuristické rozhodování lidí. Úkolem bylo určit, zda uvedené město leží nebo neleží v zadaném státě. Experiment proběhl v Chicagu ve Spojených státech amerických. Aby nedošlo k ovlivnění naměřených údajů předchozími znalostmi, měli účastníci z řad studentů místní univerzity za úkol identifikovat německá města. Informace o četnosti výskytu názvů těchto měst v souvislosti s tím, že jsou německá, byly získány analýzou obsahu místního deníku *Chicago Tribune* mezi roky 1985 až 1997. Velikost města potom koreluje s četností jeho zmiňování v daném tisku s celkem vysokou hodnotou 0,82 [2].

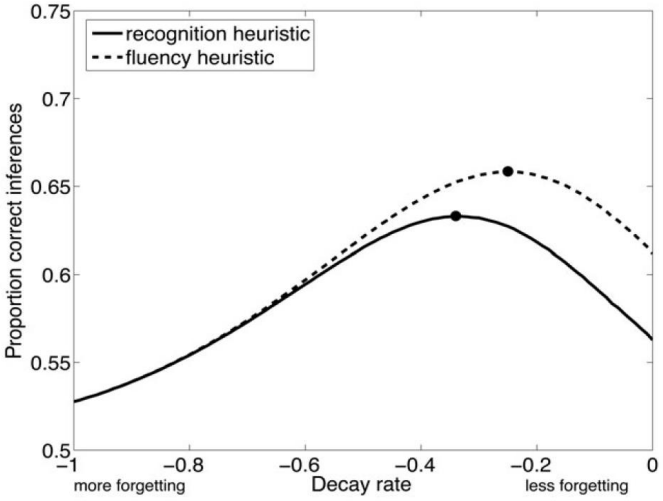
Byl měřen čas, jak dlouho trvalo rozhodování, a úspěšnost tohoto rozhodování. Čas potřebný pro rozhodnutí byl potom převeden na úroveň aktivace $A_i$ daného bloku pomocí vztahu $T_i = F \cdot e^{-A_i \cdot f}$, který popisuje ACT-R. Zjištěné aktivace byly aproximovány regresním modelem s využitím logistické funkce (viz Graf 2). Hodnota limitu pro vybavení $\tau$ (*threshold*) byla podle této aproximace stanovena na $\tau = 1{,}44$ a šum v aktivačních hodnotách $s$ měl velikost $s = 0{,}73$.[2]




**Graf 2: Schopnost rozpoznání německých měst jako funkce aktivace bloků obsahující názvy těchto měst v experimentu, v němž měli studenti americké univerzity rozhodovat, zda je či není uvedené město německé.[2]**

Při experimentech s proměnnou rychlostí degradace paměti v čase při úlohách vyžadujících heuristické rozhodování bylo zjištěno, že při hodnotě $d = 0{,}5$ (což je i základní hodnota podle ACT-R) byla zjištěna úspěšnost rozhodování 61 %. Nejlepších výsledků dosáhly modely při nastavení $d = 0{,}34$, a to v 63 % úloh (*recognition heuristic*). Při menších a vyšších hodnotách parametru $d$, například pro $d = 0$ (žádné zapomínání) nebo $d = 1$ (rychlé zapomínání), klesala úspěšnost k 50 %

a tedy k náhodnému rozhodování bez dopadu heuristiky na výsledky tohoto rozhodování. Při proměnném parametru $d$ v procesu zapomínání bylo dosaženo úspěšnosti až 66 % (*fluency heuristics*). Pro údaje získané z modelů viz Graf 3.[2]




**Graf 3: Funkce poměru správných odpovědí v závislosti na rychlosti degradace paměti při experimentech s kognitivními modely.[2]**

### 2.2.9 Kontextová úroveň aktivace bloku

Každý blok má jak svoji základní úroveň aktivace $B_i$, tak i, pokud je v relaci s jinými bloky, kontextovou úroveň aktivace $C_i$. Obě tyto úrovně aktivace dají pak v součtu celkovou úroveň aktivace $A_i$ (viz Rovnice 12). Pokud je blok $i$ v relaci s jinými bloky, mohou být tyto bloky brány jako zdroje aktivace při snaze vybavit si daný blok $i$. Toto se projeví zvýšenou úrovní aktivace $C_i$ podle toho, jak důležitá byla relace s těmito bloky v minulosti (vyjadřuje úroveň aktivace relace $S_{ji}$) a jaká je jí přiřazována váha pro aktuální kontext (vyjadřuje váha pozornosti $W_j$). Celkové zvýšení úrovně aktivace $A_i$ má potom za následek pravděpodobnější a rychlejší vybavení bloku z paměti.[2]

$$C_i = \sum_{j \in C} W_j S_{ji}$$

**Rovnice 17: Kontextová úroveň aktivace $C_i$ bloku $i$.**

Kontextová úroveň aktivace se získá součtem úrovní aktivací $S_{ji}$ jednotlivých relací bloku $i$ s blokem $j$, kde blok $j$ je brán jako zdroj aktivace pro blok $i$ ve snaze si blok $i$ vybavit.

Rovnice pro výpočet vah pozornosti nastavuje $W_j$ standardně na hodnotu $W/n$, kde $n$ je počet zdrojů aktivace a hodnota $W$ vyjadřuje, jakou důležitost je schopen jedinec přiřadit kontextové aktivaci, a tedy učinit obsah paměti více či méně

kontextově závislý. Tato aktivace tak může mít celkově vyšší hodnotu než základní aktivace bloku. V základním nastavení lze hodnotu celkové váhy kontextu nastavit na $W = 1$ (viz [2]).

### 2.2.10 Úroveň aktivace relace

Naučenou sílu relace mezi bloky $j$ a $i$ lze vyjádřit jako úroveň aktivace této relace pomocí parametru $S_{ji}$. Tato hodnota vyjadřuje, o kolik přítomnost bloku $j$ jako kontextové nápovědy (*cue*) zvyšuje pravděpodobnost vybavení bloku $i$, pro který je tedy blok $j$ brán jako zdroj kontextové aktivace. Této úrovni pak může být přisouzena různá váha $W_j$, která určuje kontext (zda použít danou relaci či nikoliv a jaká je váha přiřazená této relaci). Pro výpočet hodnoty aktivace relace $S_{ji}$ bylo navrhováno mnoho různých vztahů (například viz [16]) [2]. Aktuálně se v ACT-R používá následující vztah:

$$S_{ji}(t_0) = S - \ln(fan_j)$$

**Rovnice 18: Výchozí úroveň aktivace $S_{ji}$ relace mezi bloky $j$ a $i$, kde $j$ je zdrojem aktivace pro blok $i$.**

V uvedené rovnici parametr $fan_j$ vyjadřuje, u kolika bloků je blok $j$ jako zdroj aktivace, tedy počet odchozích relací (výstupních hran) pro blok $j$. Hodnota $S$ se nastavuje v rozsahu 1,5 až 2 tak, aby korespondovala s hodnotami získanými v experimentech. Počet odchozích relací je pro existující blok vždy minimálně jedna, pokud je tento blok zdrojem aktivace pro jiný blok.[15]

Je nutné poznamenat, že počáteční úroveň aktivace relace $S_{ji}(t_u)$ v okamžiku použití vazby degraduje podobně jako hodnoty základních aktivací bloku (viz Rovnice 13). Důvodem pro tento předpoklad je, že degradace kontextové aktivace musí podléhat stejným pravidlům jako degradace základní aktivace bloku[1]. Protože se však jedná o úroveň aktivace, může být i záporná. Je nutné ji převést na hodnotu aktivace $s_{ji}$ pomocí vztahu $s_{ji} = e^{S_{ji}}$, aby bylo možné uvedený vztah použít. Získanou hodnotu aktivace relace lze potom časem degradovat podobně jako hodnotu užití základní aktivace bloku.

[1] S tímto předpokladem se zatím v ACT-R neuvažuje. To znamená, že předpovědní model má omezené možnosti. Důvodem jsou příliš složité vztahy, které je obtížné popsat analyticky.

$$s_{ji} = \ln \sum_{u=1}^{n} \frac{s_u(t_{0u})}{t^d}$$

**Rovnice 19: Degradace celkové hodnoty aktivace $s_{ji}$ relace v čase mezi bloky $j$ a $i$, kde blok $j$ je zdrojem aktivace pro blok $i$.**

Parametr $s_u(t_u)$ vyjadřuje počáteční hodnotu aktivace $u$-tého užití relace na konci prvního časového intervalu od jejího vytvoření. Parametr $t$ je čas, ke kterému se počítá hodnota aktivace relace, a $d$ je faktor degradace paměti (viz s. 21).

## 2.3 Zpětná vazba

Zpětná vazba vyjadřuje situaci, kdy stav výstupu systému může ovlivňovat stav jeho vstupu. Data vystupující ze systému jsou ovlivněna daty, která do systému vstoupila v předchozích krocích a byla generována nebo ovlivněna předchozím výstupem systému. Výstup systému je tedy přesměrován z části na jeho vstup, čímž tvoří zpětnovazební smyčku. Systém se tak případně může sám řídit prostřednictvím této zpětné vazby.[18]

Aby bylo možné uvažovat o zpětné vazbě, musí být systém, který zpětnou vazbu využívá, stavový. Lze u něj tedy předpokládat, případně sledovat, změnu jeho vnitřního stavu v čase. Díky této zpětné vazbě je následně aktuální stav systému ovlivněn předchozími stavy jeho výstupů. Ty byly v předchozích krocích přivedeny na vstup systému a ovlivnily tak vývoj jeho stavu.

Systémy, které užívají zpětnou vazbu, se mohou dynamicky přizpůsobovat a pružně reagovat na měnící se podmínky vnějšího prostředí. Takové systémy mohou být soběstačné a plnit úkoly bez nutnosti operátora, který by tyto systémy řídil. Tímto mohou dynamicky reagovat na aktuální stav vstupu s přihlédnutím k aktuálnímu vnitřnímu stavu systému. Základem těchto systémů je vždy zpětnovazební smyčka. Komponenty systému mohou být řízeny přímo (*managment*), kdy jedna smyčka přímo řídí jinou, nebo nepřímo (*stigmergy*), což znamená, že smyčky sdílejí jeden pod-systém jako společné prostředí pro vzájemnou komunikaci a řízení. Nepřímý typ řízení je častý v multiagentních systémech.[9]

### 2.3.1 Typy zpětných vazeb

Podle vlivu na stav systému rozlišujeme dva základní druhy zpětných vazeb. Pozitivní zpětná vazba (*positive feedback*) nejčastěji zesiluje oscilace stavu systému představovaných hodnotami jeho parametrů. Negativní zpětná vazba (*negative feedback*) naopak tyto oscilace tlumí. V okamžiku, kdy dojde k výkyvu v hodnotě na

vstupu systému, může být tento výkyv buď utlumen negativní zpětnou vazbou, nebo zesílen pozitivní zpětnou vazbou. Případně může být zároveň použita vzájemná kombinace pozitivní a negativní zpětné vazby.[18]

V různých vědeckých oborech je na termín zpětná vazba nahlíženo z odlišného úhlu pohledu. Jednotlivé definice pro popis zpětné vazby a jejího vlivu na systém si často dokonce zdánlivě odporují. Technická literatura zdůrazňuje vliv zpětné vazby na odchylku systému (kvantitativní pohled), případně výkyv aktuální hodnoty od referenční, přičemž pozitivní vazba odchylku zvětšuje a negativní zmenšuje.[18] V psychologii a sociálních vědách je zkoumáno, jak se výsledný systém díky zpětné vazbě chová a co způsobuje zpětná vazba se stavem tohoto systému (kvalitativní pohled). V tomto případě má pozitivní zpětná vazba žádoucí vliv na stav systému, zatímco negativní zpětná vazba nežádoucí. Příklady různých používaných pojmů jsou uvedeny v následující tabulce.

| Terminologie typy zpětných vazeb podle dopadu na stav systému | | | |
|---|---|---|---|
| Pozitivní | (*positive*) | Negativní | (*negative*) |
| Regenerativní | (*regenerative*) | Degenerativní | (*degenerative*) |
| Posilující | (*reinforcing*) | Vyvažující | (*balancing*) |
| Samo-posilující | (*self-reinforcing*) | Samo-opravná | (*self-balancing*) |
| Odchylku-posilující | (*discrepancy-enhancing*) | Odchylku-redukující | (*discrepancy-reducing*) |

**Tabulka 2: Různé pojmy požívané pro dva typy zpětné vazby podle vlivu na stav systému.**

### 2.3.2 Vliv zpětné vazby

V komplexních systémech, které sestávají z mnoha komponent, jež mohou také využívat vlastní zpětnou vazbu, vznikají složité závislosti mezi vstupem, stavem a jeho následným vlivem na výstup. Díky přítomnosti zpětných vazeb, a tím i interakce mezi výstupem a vstupem může docházet k exponenciálnímu zesilování nebo tlumení v hodnotách vnitřních stavů těchto komponent.

Takové systémy mohou vykazovat chaotické chování, jehož konkrétní výsledek je nepředvídatelný. Při znalosti pravidel systému a jeho počátečního stavu lze však výsledek simulačně spočítat. V každý časový okamžik simulace je podle pravidel aktualizován stav systému do dalšího kroku. Díky povaze zpětné vazby na vliv vývoje stavu systému však i nepatrná změna stavu na vstupu může vést k rozsáhlé změně na výstupu s postupnými kroky simulace. Nelze tedy přímo určit

budoucí stav v libovolný časový okamžik, aniž by byly spočítány jednotlivé změny stavů, a tím simulováno chování systému.[31]

### 2.3.3 Využití zpětné vazby

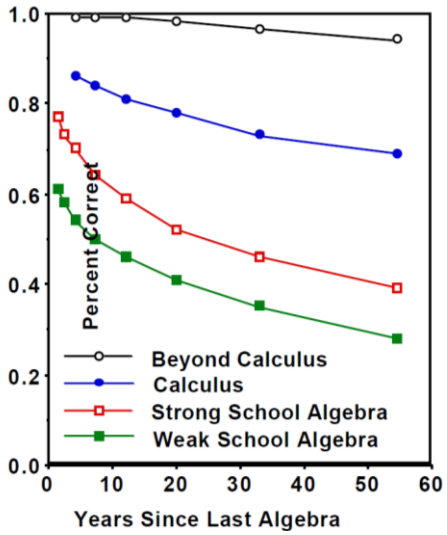
Podle typu systému může mít využití zpětné vazby předvídatelné následky. Například při návrhu číslicových obvodů je zpětná vazba zásadní pro tvorbu pamětí a paměťových členů. Ty uchovávají nastavený stav a dávají jej k dispozici na výstupu, dokud není tento stav daty z vnějšího prostředí pomocí vstupu změněn. Komplexní systémy mohou vykazovat chaotické chování a uplatňuje se v nich exponenciální růst a pokles hodnot v kombinaci s nelineární závislostí výstupů na nastavených parametrech. Příkladem z jiné oblasti pamětí jsou potom Hopfieldovy neuronové sítě (*Hopfiels networks*) využívající zpětnou vazbu pro uchování naučené informace k pozdějšímu vybavení.[22] Zjednodušenou implementaci tohoto typu sítě využívají některé komponenty teorie ACT-R.[3]

V teorii řízení se potom zpětná vazba využívá v širokém spektru metod. Tyto metody popisují formálně model fyzického systému a reprezentaci jeho stavového prostoru (*state space*). Často jsou vstup, výstup a stavové proměnné systému mezi sebou propojeny pomocí zpětných vazeb a vztahy jsou popsány prostřednictvím diferenciálních rovnic, které vyjadřují, jak daná rychlost změny jednoho parametru ovlivní rychlost změny jiného parametru systému. Stavovým prostorem je potom jakákoliv dosažitelná kombinace stavů všech stavových proměnných daného systému. Zpětnovazební smyčka je následně použita k řízení vývoje tohoto stavu požadovaným způsobem.[18]

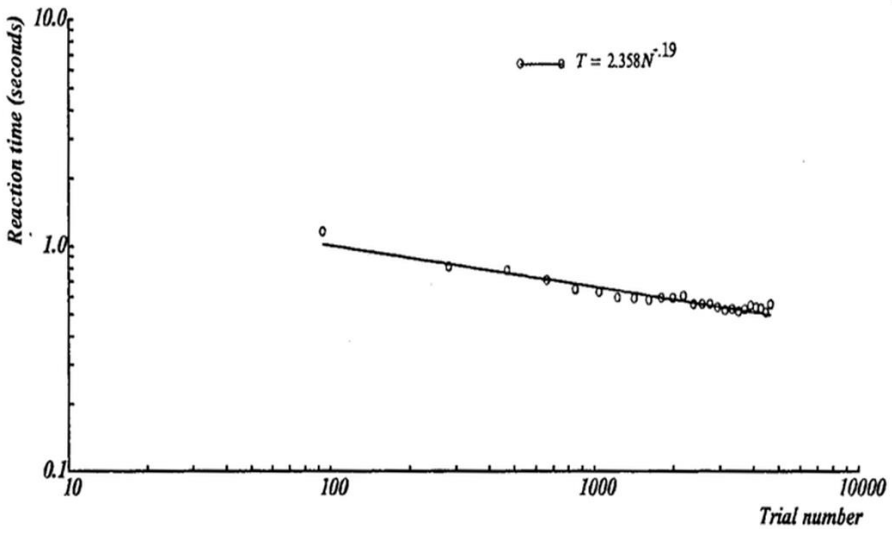
## 2.4 Efektivita učení

Efektivitu učení by bylo možné určit jako poměr celkového zisku znalostí, zkušeností a dovedností k vynaloženému úsilí nebo případně k časovému intervalu, po který je tento proces učení realizován. Větší efektivita učení potom znamená, že jsme stejné množství zisku se shodnými výstupy při jeho měření schopni realizovat za kratší časové období, případně za stejné časové období realizovat potenciálně více zisku v tomto procesu. Se zvýšenou efektivitou učení také souvisí snížení množství energie, které je nutné vynakládat pro dosažení stejných výsledků. Toto se projeví ve zvýšeném výkonu v dané úloze, kdy lze v experimentech měřit rychlost a přesnost zvládnutí dané úlohy a trvalost získaných znalostí, zkušeností a dovedností v čase.

V teorii ACT-R použitý vztah $t^{-d}$ (viz Rovnice 13, s. 20) popisuje tři důležité efekty, které se vyskytují v experimentech souvisejících s pamětí. Prvním je mocninný zákon učení (*Power Law of Learning*), který udává, jak se při opakovaném řešení problémů výkon zvyšuje, čímž se snižuje doba pro jejich řešení. Druhý je mocninný zákon zapomínání (*Power Law of Forgetting*), jenž vyjadřuje pokles schopnosti paměti vybavit si zapamatované položky v průběhu času. Třetí je efekt násobení praxe a schopnosti udržet informace v paměti (*Multiplicative Effect of Practice and Retention*) vyjadřující, že opakovaná praxe vede k násobení schopnosti paměti udržet si zapamatované informace.[17] Byla navržena také řada analytických modelů popisující tyto a další pozorované jevy.[30]




**Graf 4: Pokles schopnosti úspěšně řešit matematické úlohy algebry podle úrovně vzdělání a tím i velikosti případné praxe.[17]**




**Graf 5: Data z Fan experimentu (log-log) provedeného J. Andersonem pro bloky typu Fan 1-1.[20]**

Existuje mnoho různých statistických modelů, které mohou být použity pro aproximaci empirických dat naměřených v experimentech. Nejčastěji jsou používané exponenciální, mocninné a logaritmické regresní modely.[20] Z naměřených údajů (viz Graf 5) lze pak určit čas rozpoznání nebo vybavení jako funkci počtu pokusů.

$$T_r(i)_{fan\,1-1} = 2{,}358 \cdot N^{-0.19}$$

**Rovnice 20: Čas rozpoznání bloku *i* jako funkce počtu vystavení pro blok typu Fan 1-1.**

## 2.5 Myšlenková mapa

Jedná se o grafický a vizuální nástroj pro reprezentaci znalostí a jejich vzájemných vazeb, pomocí kterého lze snadněji získat informace pro rozhodování o dalším postupu. Tím lze zvýšit efektivitu procesu, ve kterém rozhodování probíhá. Myšlenkové mapy byly navrženy s cílem zvýšení schopnosti paměti udržet informace, a tím i zvýšit produktivitu.[4] Tento nástroj umožní sledovat a formovat strukturu znalostí jednotlivce nebo pracovní skupiny o zvoleném předmětu zájmu nebo tématu a jsou jen jedním z nástrojů pro grafickou reprezentaci znalostí.[24]

Důležitým prvkem v myšlenkových mapách jsou vazby, které vybrané koncepty propojují. Hlavní koncept je umístěn do středu mapy. Okolo je umístěno omezené množství souvisejících témat, které daný koncept propojuje pomocí vazeb. Vazby jsou identifikovány klíčovými slovy, které udávají význam propojení mezi tématem a hlavním konceptem. Každé z témat může obsahovat sadu souvisejících pojmů nebo konceptů, která se s tímto tématem pojí a tvoří druhou úroveň hierarchie myšlenkové mapy. Principem je posun od abstraktních konceptů na nejvyšší úrovni ke konkrétnějším konceptům na nižších úrovních. Grafická reprezentace mapy je často provedena s využitím barev a doplněna obrázky, nákresy nebo symboly.[23]

Z pohledu teorie ACT-R jsou koncepty v myšlenkové mapě reprezentovány jako bloky informací. Pomocí relací jsou potom tyto koncepty propojovány. Abstraktní, a tím málo známé nebo pochopené koncepty, jsou umístěné ve středu mapy. Potenciálně mají mnoho výchozích vazeb a relací s jinými koncepty, a tedy vysoké fan číslo (viz Obrázek 2, s. 13). Díky tomu jsou tyto koncepty hůře vybavovány a nemohou být tak snadno použity pro pochopení toho, jak hlavní koncept propojuje související oblasti a jak zapadá do celkového kontextu řešené úlohy.

V tomto ohledu myšlenková mapa umožní v prvním kroku graficky reprezentovat abstraktní koncept jako větší a umístěný ve středu mapy, a tím zvýšit

jeho základní aktivaci. Druhým krokem je omezit množství použitých relací, což se projeví nárůstem kontextové aktivace bloku z důvodu snížení fan čísel souvisejících konceptů. Abstraktní koncept tak může být použit pro propojení jiných konkrétnějších konceptů a umožní tvůrci myšlenkové mapy uvědomit si souvislosti mezi nimi a jejich přínos pro hlavní koncept, který je formován. Výsledkem tvorby mapy je potom minimalizace počtu vazeb a síťový efekt, kdy je abstraktní koncept používán častěji, čímž je šance na jeho vybavení zvýšena, ačkoliv má vyšší fan číslo. Toto následně vede k vybavení si souvislostí mezi různými koncepty, které hlavní abstraktní koncept propojuje, v důsledku čehož jsou tvořeny a formovány znalosti použitelné pro rozhodovací situace v kontextu řešené úlohy.

## 2.6 Praktické aplikace výsledků ACT-R

Význam teorie ACT-R a jiných podobných teorií spočívá v tom, že lze vytvořit matematický model uživatele. Tento model lze využít například pro předpověď chování uživatelů.[29] Další oblastí je potom tvorba kognitivních tutorů. Jedná se o speciální typ programového vybavení, které umožní zvýšit efektivitu výukového procesu díky matematickému modelu uživatele. Tento model je následně použit pro předpověď budoucího stavu schopností uživatele.[15]

Model může být vytvářen na základě předchozích dat získaných z interakce uživatele s danou programovou aplikací a jejím obsahem. Pro získání dat, podle kterých bude model vytvářen, lze použít databázi naměřených hodnot a technik *data miningu*. Nástrojem pro tvorbu modelu mohou být statistické metody. Tímto přístupem je vytvářen statistický model uživatele na základě jeho předchozího chování a získaných výstupů. Model potom umožní za předpokladu neměnných podmínek případnou predikci budoucího stavu.

Druhým přístupem pro tvorbu modelu je přístup simulační. V tomto případě je na základě znalostí a pravidel chování uživatele vytvořen simulační model. Tento model je parametrizován podle zjištěných výstupů modelovaného uživatele a následná simulace chování tohoto modelu umožní odhad budoucího stavu. Výhodou tohoto přístupu je možnost model průběžně parametrizovat tak, aby byl ve shodě s aktuálním stavem modelovaného uživatele. Tím je dosaženo větší míry shody modelu s modelovaným uživatelem využívajícím služby kognitivního tutora. Uvedený přístup se v současné době pro svou výpočetní náročnost příliš nevyužívá. Dalším

z důvodů je převládající centralizovaná orientace služeb na servery a potenciální množství uživatelů v řádech milionů. S tím také souvisejí vysoké náklady na udržování a využívání simulačního modelu každého uživatele.

Jedním z představitelů kognitivních tutorů pro výuku jazyků je projekt Duolingo[25], který vytváří analytický model uživatele na základě dat získaných z předchozí interakce uživatele s obsahem kurzu. Tento model je pak použit pro předpověď stavu uživatele v dané oblasti kurzu a pro řízení předkládaného obsahu. Díky přizpůsobení se stavu a potřebám uživatele je tak zvýšena celková efektivita procesu učení.[26] V současné době (rok 2016) využívá služeb Duolinga více než 120 milionů uživatelů, kteří každý měsíc vyplní více než 6 miliard cvičení.[25] Efektivita Duolinga byla zkoumána a potvrzena několika nezávislými studiemi, jednou z těchto studií je například [27]. Duolingo je dostupné jak prostřednictvím webové aplikace, tak pomocí mobilních aplikací, a rozvíjí používání vybraného cizího jazyka na úrovni překladu psaného textu, poslechu i verbální komunikace ve zvoleném jazykovém páru.

Druhou uvedenou aplikací kognitivního tutorování je projekt KhanAcademy[28], který je nástrojem pro podporu výuky v širokém spektru oborů. Jednou z hlavních oblastí je výuka matematiky, kterou komplexně pokrývá od naprostých základů až po integrální počet. V matematice se KhanAcademy zaměřuje jak na podpůrný obsah tvořený výukovými materiály v podobě textů, videí a multimédií, tak na individuální přístup k potřebám vzdělávaného uživatele prostřednictvím generovaných interaktivních úloh a pořadí, ve které jsou jednotlivé úlohy doporučovány k řešení nebo studiu. Aktuálně (rok 2016) KhanAcademy využívá více než 40 milionů uživatelů.[28]

Duolingo i KhanAcademy jsou v současné době volně přístupnými projekty s plnou funkcionalitou poskytovanou zdarma komukoliv a odkudkoliv prostřednictvím Internetu.

## 2.7 Simulace jako metoda

Pro následující podkapitolu byly použity informace z [6], dále informace z kurzu Znalostní Technologie 3 na UHK FIM od doc. Kamily Štekerové a ze skript Stochastické modely a modelování prof. Hany Skalské [7].

Model dostatečně věrně a s nějakým záměrem popisuje a reprezentuje objekt, kterým může být pozorovaný předmět, systém, proces nebo děj. Modely jsou vytvořeny a používány pro řešení problémů nebo jako prostředek pro získání odpovědí týkajících se modelovaného systému nebo celé třídy systémů. Některé typy systémů se vyvíjejí na časových měřítkách, které neumožňují s nimi provádět přímé experimenty v reálném čase (například růst měst a využití zemědělské půdy). Dalším typem jsou systémy obsahující mnoho interagujících částí a reálné experimenty, pokud je možné je provádět, jsou obtížně opakovatelné. Nicméně znalosti o fungování těchto systémů použitelné pro další rozhodování lze získat s využitím výpočetní síly počítačů. Nástrojem se pak stává vhodně zvolené paradigma pro popis těchto modelů a následná simulace chování těchto modelů při různém nastavení.[6]

V případě, že je model napodobeninou reálného předmětu, hovoříme o modelu fyzikálním, který popisuje fyzické vlastnosti modelovaného objektu (například váha, výška, šířka, materiál). Výsledný model je pak v jistém měřítku k předmětu, který je modelován. Pokud model nepopisuje přímo fyzický předmět, ale pouze informace o modelovaném objektu, jejich strukturu a vzájemné vztahy, potom hovoříme o modelu formálním. Může se jednat například o datový model, model informačního systému nebo procesu. Vztahy a informace v modelu jsou pak zachyceny formálně jako sada pravidel a popsány slovně nebo symbolicky s využitím rovnic, grafů nebo schémat. Pokud má být formální model realizován v počítači, musí být převeden do podoby počítačového programu, přičemž takový model označujeme jako model digitální.

Modelováním označujeme proces, který vede k vytváření modelu. Vzniklé modely lze rozlišovat podle jejich chování na modely statické a dynamické. Statický model je takový, jehož výstup v daný okamžik přesně odpovídá zadaným vstupům. Takový model může být reprezentován pomocí matematické funkce, které zadáme vstupní hodnoty parametrů a jejím vyhodnocením získáme požadovanou výslednou hodnotu. Statický model lze považovat za bez-stavový. Oproti tomu výstup dynamických modelů je získán s jistým časovým zpožděním oproti vstupu, a tak je výstup modelu ovlivněn nejen jeho aktuálním stavem, ale také stavy předchozími. Stavy modelu mohou být viditelné nebo skryté a ovlivňují pozorovaný výstup v každém simulačním kroku.

V okamžiku, kdy uvažujeme stav modelu, přičemž tento stav se vyvíjí v čase podle pravidel, která popisují chování modelu, hovoříme o simulaci. Simulace následně umožní popsat vývoj modelovaného děje nebo procesu v čase. V případě digitálních modelů je stav modelu uložen v paměti počítače a simulace probíhá nejčastěji v diskrétních časových krocích. V každém kroku simulace počítač uchovává celkový aktuální stav objektu nebo systému, který model popisuje. Tento stav je následně s každým krokem aktualizován podle pravidel modelu. Části modelu mohou reagovat na změnu vstupního stavu okamžitě v dalším kroku simulace nebo s jistým časovým zpožděním v průběhu několika následujících kroků. Data ze simulace jsou pak získána jako posloupnost stavů modelu a jeho částí v jednotlivých krocích simulace. V případě modelů obsahujících mnoho podobných částí se může jednat o agregace stavů těchto částí v časových krocích.

Cíle modelování mohou být různé. Prostředkem k jejich dosažení je však vždy vytvořit s využitím jisté nutné úrovně abstrakce model, který zachytí podstatné vlastnosti modelovaného objektu. Tyto vlastnosti následně v simulaci slouží pro vytvoření zjednodušeného popisu chování originálu, který je modelován.

Existují dva základní přístupy, jak lze model vytvořit. Jedním je indukce z naměřených dat (empiricky) a druhým dedukce s využitím předchozích znalostí pravidel týkajících se chování a struktury modelovaného originálu. V obou případech jsou přijaty předpoklady, při jejichž splnění lze následně navržený model, použitá pravidla (algoritmy) pro práci s hodnotami a výsledky získané simulací považovat za platné.

Předpoklady se mohou týkat přímo návrhu částí modelu nebo pracovních podmínek, za kterých má být model používán. Pracovními podmínkami jsou jakékoliv konkrétní kombinace hodnot parametrů modelu a prostředí, v němž má být model nasazen, a které mohou podstatně ovlivňovat chování modelu a výstupy z něj získané při experimentech pomocí simulace. Dalším předpokladem je správné nastavení vlastních parametrů modelu, které probíhá při kalibraci modelu.

Podle toho, jak uvedené parametry ovlivní nebo neovlivní chování modelu a jeho dynamiku, je lze rozdělit na závislé a nezávislé. U závislých parametrů i jejich malá změna podstatně ovlivňuje dynamiku vývoje simulace modelu. V modelu se mezi vstupem a výstupem modelu uplatňuje zpětná vazba, která může oscilace hodnot parametrů modelu tlumit a má tak stabilizační efekt (negativní zpětná vazba),

nebo posilovat a může tak mít za následek přechod celého modelu do jiného stabilnějšího stavu. Nezávislé parametry mají potom nepatrný nebo malý vliv na celkovou dynamiku vývoje stavu modelu.

Vytvořený model a experimenty s ním provedené v rámci simulací mohou mít funkci vysvětlovací. Data získaná modelováním umožní návrh nebo formulaci vědeckých hypotéz. Na základě potvrzení těchto hypotéz lze formulovat vědecké teorie ověřitelné experimenty nebo potvrditelné matematickými důkazy. Pochopení principů funkce modelu, a tedy přeneseně i objektu, pro který je model vytvořen, umožní predikci, a tím dostatečně přesně předpovědět budoucí stav nebo vývoj systému a výskyt jevů souvisejících s jeho chováním při daném nastavení.

Konečným cílem modelování je získání kompletních znalostí a pochopení modelovaného děje nebo procesu. S využitím modelu a jeho predikce lze potom podle požadavků řídit vývoj stavu systému, pro který byl model vytvořen. Modely vytvářené během procesu modelování jsou prostředky k dosažení tohoto cíle.

## 2.8 Validace a kalibrace

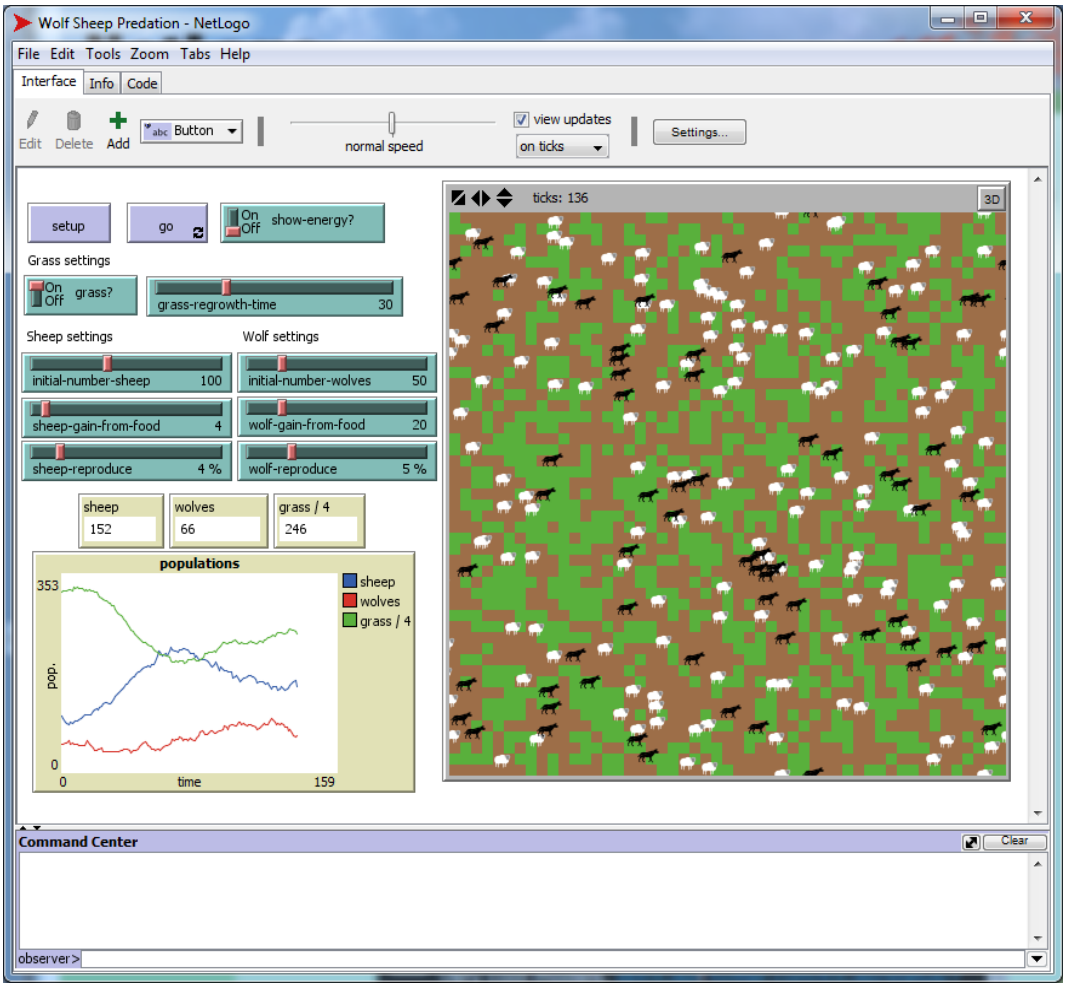
Validní model je takový model, který se chová ve shodě s teorií a vztahy, podle kterých měl být implementován. Je tedy nutné otestovat výstupy modelu a jeho částí, zda jsou ve shodě s testovacími daty získanými měřením nebo analyticky z použitých vztahů. Proces validace je prvním krokem ve fázi testování shody modelu s modelovaným systémem.

Aby bylo možné výsledky modelu použít pro předpovědní účely, je po procesu validace modelu následně nutné model kalibrovat. Kalibrace modelu spočívá v nastavení jeho parametrů tak, aby výsledky generované modelem byly v dostatečné shodě s výsledky získanými pomocí experimentů s reálným systémem. Správně kalibrovaný model poskytuje podobné výsledky jako modelovaný originál.

Po validaci a kalibraci modelu jej lze použít k provádění simulovaných experimentů, které umožní získat výsledky daleko rychleji než experimentování s reálným systémem. Simulované experimenty mohou objevit do té doby neznámé vlastnosti modelu, a tím i případné nové a neznámé vlastnosti originálu, pro který byl model vytvořen. Výsledky simulačních experimentů mohou také sloužit jako podklad pro tvorbu hypotéz a návrh reálných experimentů s cílem podpořit nebo vyvrátit navržené hypotézy.[6]

## 2.9 NetLogo

Jedná se o integrované vývojové prostředí (IDE – *Integrated Development Environment*), které umožňuje díky nízkým požadavkům na uživatele snadno vytvářet agentní a multi-agentní modely, zobrazit jejich stav a následně s těmito modely provádět simulace a experimenty. K popisu modelu využívá vlastní vysokoúrovňový slabě typovaný programovací jazyk, který kombinuje prvky jak deklarativního, tak imperativního programování.[2]




**Obrázek 3: Prostředí NetLoga [5] ve verzi 5.3.1 s ukázkou modelu *Wolf Sheep Predation*.**

NetLogo volně vychází z principů původního jazyka Logo. Jeho minimální požadavky na uživatele bez nutnosti předchozích znalostí jakéhokoliv programování jej činily ideálním prostředkem zejména pro výuku programování u mladších dětí. S tímto záměrem profesor Uri Wilensky [10], matematik, pedagog, informatik, vývojář

[2] V programovacím jazyce NetLoga nalezneme jak řídící konstrukce známé z imperativního programování (globální proměnné, cykly, větvení, skoky), tak konstrukce deklarativního programování (například logické programování, seznamy, vyhledávání). Jeho syntaxe byla oblivněna jazykem Lisp.[9]

a ředitel centra CLL[3] navrhl koncept NetLoga, které umožní být nástrojem pro široké spektrum uživatelů s využitím od výuky programování pro děti až po doménové experty z různých oblastí[4], kteří přicházejí s požadavky na tvorbu modelů, ale nemají k tomu nutné a často značně rozsáhlé znalosti z oboru informatiky a programování.

Z tohoto důvodu jsou tvůrci modelů v NetLogu při jejich vytváření odděleni od nízko-úrovňových konceptů používaných v běžném programování[5]. Výsledkem je usnadnění tvorby modelů pro začínající uživatele. Z osobní zkušenosti jsou předchozí jednostranné znalosti z programování v jiném jazyce nebo paradigmatu v počátečních fázích programování v NetLogu dokonce spíše nevýhodou. Velkou výhodu mají uživatelé, kteří jsou seznámeni s programováním ve funkcionálních jazycích.

Tvorba modelů se zaměřuje na definování sady pravidel pro agenty tvořenými autonomní objekty s definovanými pravidly chování. Při simulaci tito agenti interagují s ostatními ve společném prostředí. Během simulace je chování agentů ovlivněno jejich stavem a stavem jejich bezprostředního okolí. Z pohledu uživatele modelu lze potom pozorovat chování celého systému tvořeného agenty a jejich interakcí.[11]

Zaměření NetLoga je tedy zejména na ověření základního předpokladu o konceptu modelu na základě jeho pozorovaného a zjištěného chování. Dále potom práce s existujícími modely zařazenými do rozsáhlé knihovny modelů, jejichž velkou část navrhl tvůrce NetLoga Uri Wilensky [10]. S agentovými a multi-agentními systémy také souvisí zkoumání fenoménu emergence, kdy chování na mikro-úrovni je přeneseno na chování na makro-úrovni.[6]

Velkým přínosem pro uživatele je potom vlastní reálná zkušenost s modelem a experimentem a dále tvorba nových znalostí využitelných pro pochopení komplexních systémů. Uplatnění nalezne NetLogo při zkoumání modelů ve výuce jak na základním, tak středoškolském i vysokoškolském stupni. V pokročilém nastavení jej lze také použít pro oblast základního výzkumu v oblasti tvorby oborově specifických modelů.

NetLogo je volně šiřitelný software s otevřeným zdrojovým kódem pod licencí GPL, a tak práce s využitím NetLoga musí být také uvolněny pod licencí GPL. V případě, že by toto nebylo možné, existují také komerční licence umožňující jiné

[3] CCL - *Center for Connected Learning and Computer-Based Modeling* nyní sídlící v Bostonu, USA.[11]
[4] Například v oblasti vzdělávání, psychologie, sociologie, ekonomiky, biologie nebo fyziky.[11]
[5] Jedná se zejména o práci s datovými typy a explicitní práci s pamětí.

smluvní podmínky.[8] Prostředí NetLoga zatím nebylo lokalizováno do českého jazyka, nicméně existuje velmi kvalitní a úplný překlad původního anglického průvodce pro uživatele [8] do českého jazyka[6] zveřejněného na webu Robotomie.cz.[14]

S využitím NetLoga bylo od jeho uvedení v roce 1999 do současnosti (2016) publikováno mnoho vědeckých článků a studií, ať už pracovníky CLL nebo nezávislými osobami. Bylo také vytvořeno více než 1000 modelů popisujících komplexní jevy jak v přírodní, tak sociální oblasti.[5]

Implementace vlastního prostředí NetLoga je postavena na programovacích jazycích Java a Scala a je tedy spouštěno pomocí JVM[7]. Pro překlad programového kódu modelů je v jádře NetLoga využíván hybridní překladač/interpret překládající uživatelsky psaný kód na byte-kód JVM, kdy se využívá kombinace vkládání předkompilovaného kódu a přímého volání funkcí JVM.Aktuálně je dostupná i webová verze NetLoga využívající JavaScript místo JVM. Modely lze tak spouštět ve webovém prohlížeči, nicméně je zde jisté omezení oproti verzi pro Javu. Při testech rychlosti simulací v reálném čase došlo přibližně k pětinásobnému zrychlení od první vydané verze NetLoga k aktuální verzi 5.3.1 (2016). Pro zvýšení rychlosti a zvětšení rozsahu simulací je plánován vývoj samostatného kompilátoru jazyka NetLoga.[12]

Simulovaný model je v prostředí NetLoga tvořen agenty, kdy za agenta lze považovat jakoukoliv samostatnou entitu, která je součástí simulačního modelu a je jí možné definovat jisté chování. Tato entita si uchovává svůj vnitřní stav, který ji odlišuje od ostatních entit. Při simulaci je tento vnitřní stav měněn na základě definovaného chování jak podle aktuálního vnitřního stavu entity, tak pomocí informací, které entita získává z lokálního prostředí, ve kterém funguje. Pomocí prostředí potom dochází k interakci mezi jednotlivými entitami, které se označují jako agenti.[6]

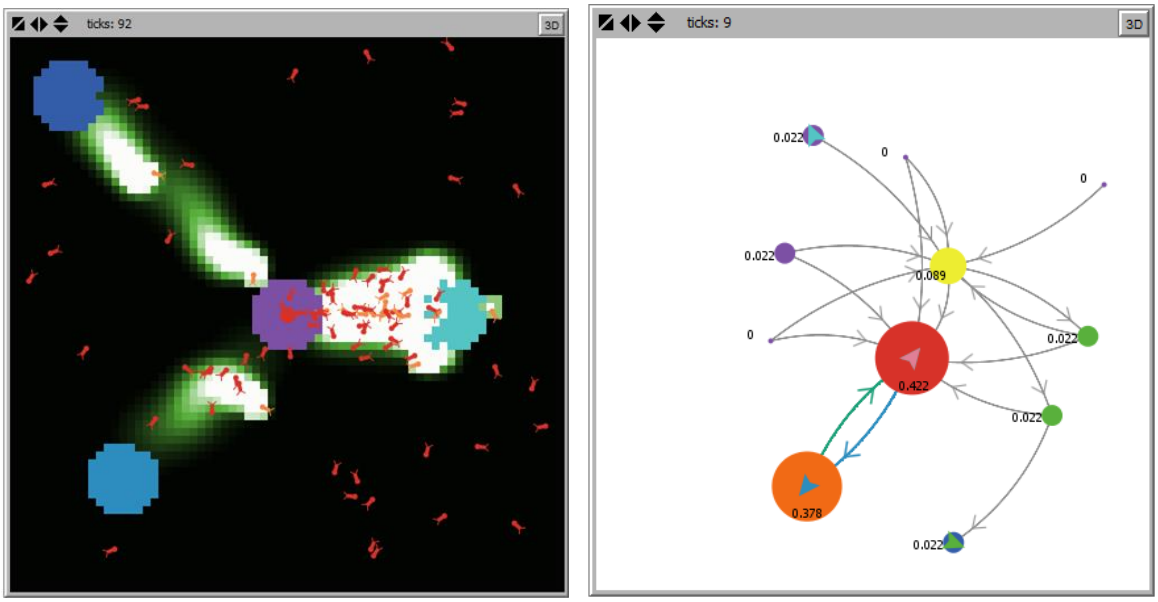
NetLogo poskytuje čtyři základní typy agentů, a to želvy (*turtles*), plochy (*patches*), spojnice (*links*) a pozorovatele (*observer*). Agent pozorovatele je v simulaci vždy pouze jeden a poskytuje instrukce ostatním typům agentů. Slouží tedy k řízení modelu. Průběh a výsledky simulace jsou generovány ostatními agenty a agent

[6] Celá příručka má více než 400 stran a detailně pokrývá všechny oblasti týkající se použití NetLoga. U překladu není uveden autor, ale zmíněná webová stránka je projektem Matematicko-fyzikální fakulty Univerzity Karlovy v Praze a Akademie věd České republiky, což dodává jistou garanci nabízeného obsahu.

[7] JVM - *Java Virtual Machine*, virtuální stroj umožňující spouštění programů vytvořených v jazyce Java.

pozorovatele následně umožní tyto výsledky zprostředkovat uživateli v podobě grafického nebo jiného výstupu. Příkazy lze tomuto agentu zadávat pomocí grafického uživatelského rozhraní modelu obsahujícího ovládací prvky nebo přímo prostřednictvím příkazové řádky modulu *Command Center* umístěné ve spodní části prostředí NetLoga. Tento agent také samozřejmě vykonává kód modelu.[8]

V prostorově orientovaných modelech, kdy se uvažuje s jistou polohou agenta v prostoru, jsou pro tento účel využívány želvy. Prostředí, ve kterém se tito agenti pohybují, je pak tvořeno plochami, které jej rozdělují do pravidelné čtvercové mřížky.[13]




**Obrázek 4: Prostorový model *Ants* (vlevo) a síťový model *PageRang* z knihovny modelů NetLoga.[5]**

U síťových modelů, kde více než fyzické umístnění agenta v daném prostředí hraje roli jeho logické umístění v rámci vztahů k ostatním agentům, tvoří prostředí agenta jeho sousedi, se kterými je propojen pomocí agentů typu spojnice. V případě síťových modelů fyzické umístění agenta může, ale i nemusí mít význam v konkrétním modelu.

# 3 Praktická část

Funkce i procedury mohou mít v NetLogu vedlejší efekty (*side effects*). To znamená, že jejich použití může způsobit změnu stavu modelu. Toto chování je žádané, protože bez něj by nebylo možné realizovat simulaci modelu v čase[8].

Problém může nastat u složitějších modelů, kdy pořadí vykonávaných příkazů může zcela změnit výstupy modelu. Proto je nutné mít tvorbu modelu, jeho návrh a strukturu pod kontrolou. V opačném případě se u složitějších modelů můžeme dostat do nekončící spirály, kdy s každou novou funkcí nebo změnou pořadí ve vykonávání příkazů přestane fungovat předchozí, do té doby fungující část.

Tvorba a úprava funkčního a validního modelu tak předpokládá naprostou znalost zákonitostí a časování mezi jednotlivými částmi modelu. Tato znalost se utváří také v rámci vlastního procesu modelování a vede k hlubšímu pochopení problematiky vzniku pozorovaných hodnot a dat získaných experimentováním. U jednoduchých modelů, jejichž kód lze napsat na několik řádků, je tento úkol podstatně jednodušší, než u modelů komplexnějších.

Z tohoto důvodu je co nejvíce základní funkcionality modelu postupně přesunováno s vývojem jednotlivých verzí do funkcí. Ty jsou navrženy tak, aby, pokud je to možné, neměly vedlejší efekty. To znamená, že dostanou potřebné hodnoty zadány v parametrech na svém vstupu, provedou požadovaný výpočet a výsledek vrátí jako svůj výstup. Pokud tak z jakéhokoliv důvodu dojde ke změně pořadí výpočtů při použití těchto funkcí, je zajištěno, že jakékoliv vedlejší efekty jsou lokalizovány na místo použití těchto příkazů. Další výhodou funkcí je lokalizovaná úprava chování modelu, přizpůsobení funkcí a možnost je skládat do složitějších funkcí dle principů funkcionálního programování. Funkce lze také z modelu vyjmout a použít je beze změny v jiném podobném modelu.

Model deklarativní paměti podle ACT-R následně implementovaný v NetLogu byl navrhován postupně s využitím modelovacího cyklu navrženého pro modely založených na agentech (*agent based models*) [6]. Postupně bylo vytvořeno několik desítek verzí, u nichž byly testovány různé aspekty modelovaného systému.

První verze modelu sloužily k seznámení s principy teorie ACT-R v oblasti deklarativní paměti a jejími očekávanými výstupy. Byly tak postupně modelovány

[8] Během simulace dochází ke změně aktuálního stavu modelu se změnou času.

jednotlivé vlastnosti nyní již finálního modelu. Modely v prvních verzích tak měly spíše funkci vysvětlovací a byly zaměřeny především na validitu procesů probíhajících v modelu. Jedním z těchto základních procesů je zapomínání a degradace paměti, který popisuje mocninná funkce $t^{-d}$. Ta vyjadřuje zhoršování schopnosti paměti vybavit si informace v čase, pokud tyto informace nebyly opakovaně vybavovány.

Později, s větším pochopením teorie ACT-R v oblasti deklarativní paměti a znalostmi z předchozích verzí modelů, bylo přistoupeno ke kalibraci aktuální verze modelu. Validace probíhajících procesů byla již z větší části hotova díky rozsáhlému validačnímu testování předchozích verzí modelu, na kterém byla aktuální verze postavena. Důležité je ale poznamenat, že je nutné provést také validaci finálního modelu, aby byla získána jistota, že jednotlivé komponenty mají validní funkci dle teorie, podle které jsou implementovány.

Celou dobu byl dodržován modelovací cyklus (viz [6]). V tomto cyklu vzniká výsledný model iterativně a postupně po jednotlivých verzích. Jednotlivé fáze modelování v tomto cyklu jsou:

1. Formulace otázky
2. Sběr hypotéz sloužících pro její zodpovězení
3. Očekávané výstupy
4. Návrh struktury modelu
5. Implementace modelu
6. Analýza výstupů z modelu
7. Získané výstupy (případně validace a kalibrace modelu)

S jednotlivými verzemi modelu potom získáváme znalosti pro rozhodnutí nutná do dalších verzí, případně můžeme objevit nové zákonitosti a vztahy mezi částmi modelu, jejichž existence byla do té doby skrytá, a které lépe pomohou v pochopení funkce modelovaného systému.

## 3.1 Návrh modelu

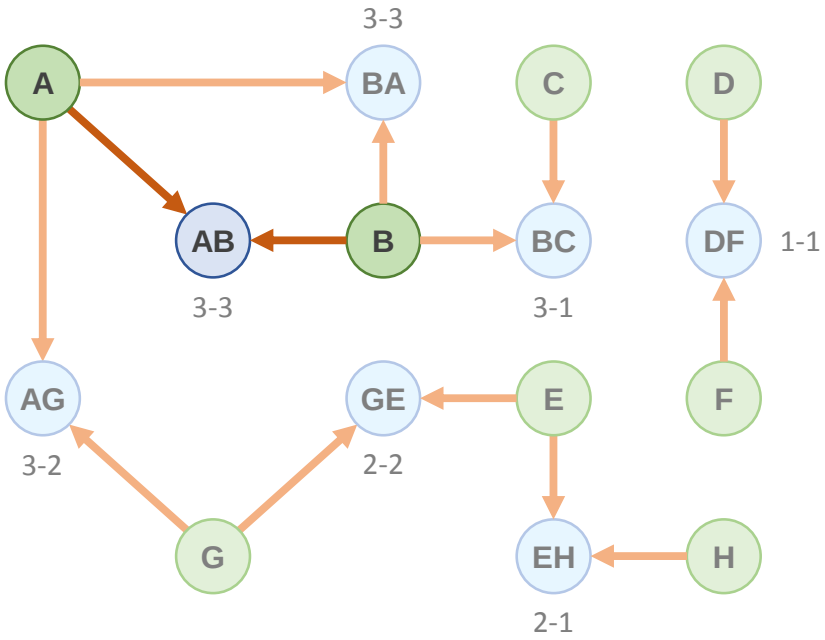
Prvním krokem pro návrh mého modelu bylo tedy seznámit se s obsahem ACT-R teorie dostatečně na to, aby mohla být vytvořena první verze modelu. U tohoto postupu bylo důležité zaměřit se na modelování částí, kterými je výsledný model

a subsystém deklarativní paměti v ACT-R tvořen. Bylo tedy nutné identifikovat základní strukturu tohoto systému.

### 3.1.1 Základní struktura

Už z prvotní analýzy ACT-R bylo patrné, že paměť uchovává informace jako bloky (*chunks*). Bloky v sobě nesou zakódovanou informaci faktického charakteru a mohou být v relaci s jinými bloky, které mohou složit jako kontextový zdroj aktivace. V paměti jsou tedy informace podle ACT-R reprezentovány síťovým modelem.

V navrženém modelu pro NetLogo se tedy budou vyskytovat dva typy agentů, a to agent bloku (*chunk, chunks*) a agent relace mezi bloky, který je bude propojovat (*relation, relations*). Bloky budou v NetLogu reprezentovány želvami (*turtles*) a relace pomocí orientovaných spojnic (*links*). Spojnice potom budou propojovat bloky, které jsou zdrojem aktivace, s bloky, pro které tyto zdroje slouží (viz Obrázek 5).



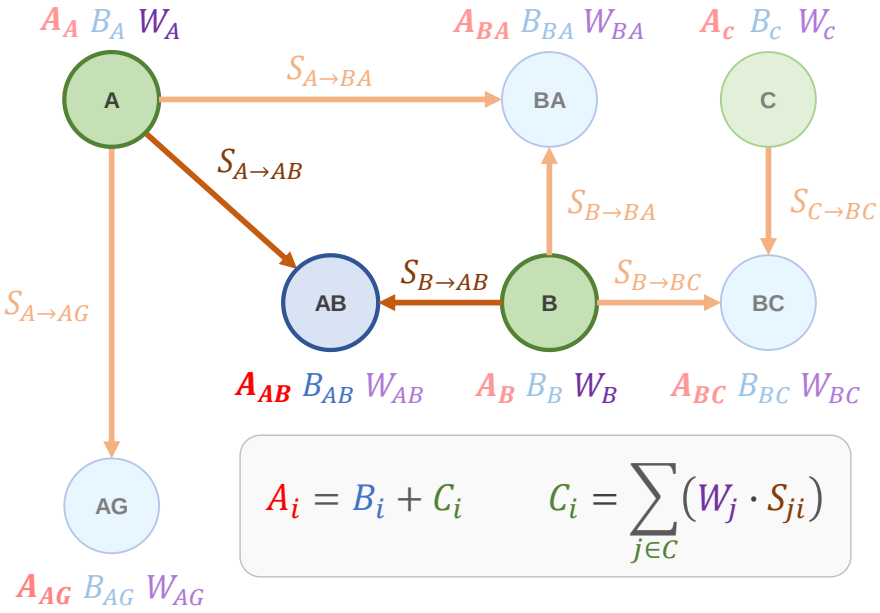

**Obrázek 5: Informace jsou reprezentovány v ACT-R modelu deklarativní paměti jako vrcholy grafu (bloky) a hrany reprezentují relace mezi nimi. Zelené bloky označují zdroje aktivace pro modré bloky. Obsah bloku je symbolicky znázorněn uvnitř bloků a u modrých bloků je zobrazen jejich typ podle Fan experimentu (viz Obrázek 2, s. 13).**

### 3.1.2 Hodnoty a úrovně aktivace

Každý blok má celkovou úroveň aktivace $A_i$, která se skládá ze základní úrovně aktivace $B_i$ a kontextové úrovně aktivace $C_i$. Základní úroveň aktivace blok získá používáním v minulosti, a to součtem hodnot aktivací svých jednotlivých užití

$b_u$. Kontextovou úroveň aktivace získá od relací se souvisejícími bloky a díky váze, která je těmto relacím přisuzována pro danou úlohu a kontext. Úroveň kontextové aktivace je závislá jak na aktuální úrovni aktivace dané relace získané díky minulému a případně opakovanému používání, tak na kontextové váze, pokud je daná relace relevantní pro požadavek na vybavení.

Celková úroveň aktivace $A_i$ bloku $i$ ovlivňuje jak pravděpodobnost vybavení bloku $P_r(i)$, v případě více vybavených bloků potom pravděpodobnost $P(i)$, jaký blok zvolit, tak i čas $T_r(i)$, jak dlouho bude vybavení bloku trvat. Tyto údaje lze potom měřit v experimentech a srovnat s výsledky modelu (viz kapitola Fan experiment, s. 12).




**Obrázek 6: Jednotlivé typy úrovní aktivací v modelu deklarativní paměti podle ACT-R pro bloky a relace mezi nimi. Jedná se o celkovou úroveň aktivace bloku $A_i$, základní úroveň aktivace bloku $B_i$ a kontextovou úroveň aktivace bloku $C_i$, která se určí podle aktivace relace $S_{ji}$ mezi bloky $j$ a $i$ a váhy $W_j$, která je této relaci přiřazena v dané úloze při vybavování bloku.**

Při návrhu prvních verzí síťového modelu vznikl požadavek důsledně rozlišovat, zda pracujeme s **úrovní aktivace**, která může být kladná i záporná, nebo **hodnotou aktivace**, která může být pouze kladné číslo (případně 0 pro ještě nepoužitý blok) a degraduje časem pomocí mocninné funkce $t^{-d}$. Pro úrovně aktivace se proto používají v názvech proměnných velká písmena a pro hodnoty aktivací písmena malá a lze mezi nimi převádět podle níže uvedených vztahů (viz Tabulka 3).

**Tabulka 3: Vztah mezi úrovní aktivace a hodnotou aktivace pro různé typy aktivací.**

| Aktivace | | Úroveň | Hodnota |
|---|---|---|---|
| obor hodnot (pro existující bloky) | | kladné i záporné číslo | vždy číslo větší než 0 |
| **Celková** | (*all activation*) | $A_i = ln(a_i)$ | $a_i = e^{A_i}$ |
| **Základní** | (*base activation*) | $B_i = ln(b_i)$ | $b_i = e^{B_i}$ |
| **Kontextová** | (*context activation*) | $C_i = ln(c_i)$ | $c_i = e^{C_i}$ |
| **Relace mezi bloky *j* a *i*** | (*relation activation*) | $S_{ji} = ln(s_{ji})$ | $s_{ji} = e^{S_{ji}}$ |
| **Užití bloku nebo relace** | (*use activation*)[9] | $X_u = \ln(x_u)$ | $x_u = e^{X_u}$ |

### 3.1.3 Degradace hodnot aktivací

Jak již bylo zmíněno, u **hodnot aktivací** probíhá degradace pomocí mocninné funkce $t^{-d}$, která nejlépe popisuje efekty pozorované v experimentech.[20] Aby bylo možné vypočítat hodnotu aktivace užití $x_u$ k času $t$, rozšíříme vztah získaný z dříve popsaného vztahu v teoretické části (viz Rovnice 14), a to doplněním času, pro který je hodnota aktivace počítána.

$$x_u(t) = x_u(t_u) \cdot (t - t_u)^{-d}$$

**Rovnice 21: Výpočet hodnoty aktivace užití $x_u$ v čase $t$ na základě hodnoty aktivace $x_u$ v čase vytvoření užití $t_u$ a průměrné rychlosti degradace paměti $d$ mezi těmito časy.**

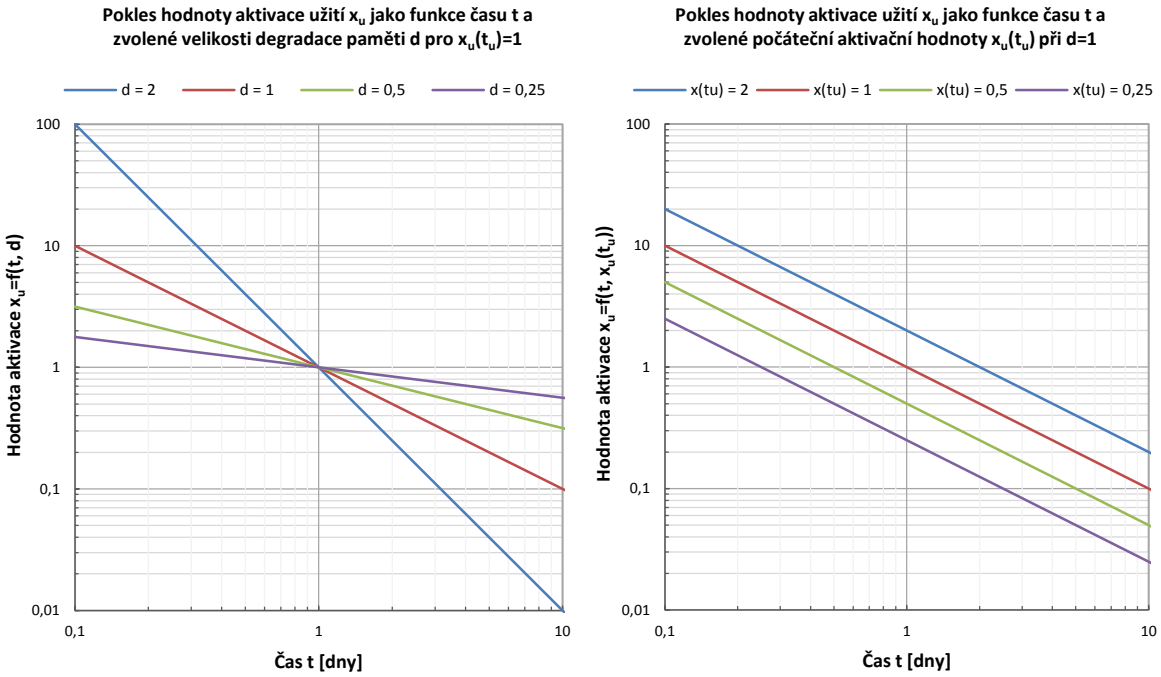
Uvedený vztah je použitý jak pro výpočet jednotlivých hodnot aktivací užití bloku ($b_u$), tak pro výpočet hodnot aktivací užití relací ($s_u$). Za $x_u$ si lze dosadit jeden z těchto typů aktivací užití a hodnota $x_u(t_u)$ vyjadřuje zisk aktivace na konci intervalu vzniku daného užití bloku nebo relace. Rozdíl $(t - t_u)$ představuje čas, který uběhl od vzniku dané aktivace užití bloku nebo relace. Parametr $d$ potom popisuje **průměrnou** hodnotu degradace paměti mezi časem vzniku užití $t_u$ a aktuálním časem $t$. V tabulce níže (viz Tabulka 4) jsou uvedeny hodnoty aktivace pro různou počáteční hodnotu aktivace $x_u(t_u)$ a pro různé rychlosti degradace paměti $d$. Tyto hodnoty budou potom použity pro validaci implementovaného modelu. Základní časovou jednotkou v experimentech, pokud to není uvedeno, jsou většinou dny.

[9] Jedná se o případ, kdy je daný blok nebo relace použita ať už při vybavování nebo při učení bloku či relace. Používá se následně pro výpočet základní hodnoty aktivace bloku $b_i$ (*base activation*) a při výpočtu hodnoty aktivace relace $b_{ji}$ (*relation activation*).

| $x_u(t_u)$ | 2 | 1 | 0,5 | 0,25 | 2 | 2 | 2 | 2 | 0,5 | 0,5 | 0,5 | 0,5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $d$ | 1 | 1 | 1 | 1 | 2 | 1 | 0,5 | 0,25 | 2 | 1 | 0,5 | 0,25 |
| $t$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ | $x_u(t)$ |
| 1/10 | 20,0000 | 10,0000 | 5,0000 | 2,5000 | 200,000 | 20,0000 | 6,3246 | 3,5566 | 50,0000 | 5,0000 | 1,5811 | 0,8891 |
| 1/2 | 4,0000 | 2,0000 | 1,0000 | 0,5000 | 8,0000 | 4,0000 | 2,8284 | 2,3784 | 2,0000 | 1,0000 | 0,7071 | 0,5946 |
| 1 | 2,0000 | 1,0000 | 0,5000 | 0,2500 | 2,0000 | 2,0000 | 2,0000 | 2,0000 | 0,5000 | 0,5000 | 0,5000 | 0,5000 |
| 2 | 1,0000 | 0,5000 | 0,2500 | 0,1250 | 0,5000 | 1,0000 | 1,4142 | 1,6818 | 0,1250 | 0,2500 | 0,3536 | 0,4204 |
| 4 | 0,5000 | 0,2500 | 0,1250 | 0,0625 | 0,1250 | 0,5000 | 1,0000 | 1,4142 | 0,0313 | 0,1250 | 0,2500 | 0,3536 |
| 8 | 0,2500 | 0,1250 | 0,0625 | 0,0313 | 0,0313 | 0,2500 | 0,7071 | 1,1892 | 0,0078 | 0,0625 | 0,1768 | 0,2973 |

**Tabulka 4: Hodnoty aktivací $x_u$ užití bloků a relací při degradaci v čase $t$ pro různá nastavení počáteční hodnoty aktivace užití $x_u(t_u)$ a rychlosti degradace paměti $d$. Čas vzniku užití je pro všechny vypočtené hodnoty nastaven na $t_u = 0$.**

U poklesu hodnot aktivací si lze povšimnout, že hodnota aktivace užití nejdříve klesá velmi strmě bez ohledu na počáteční velikost $x_u(t_u)$ v závislosti na velikosti degradace paměti $d$. Čím je ale čas od vzniku aktivační hodnoty delší, tím je absolutní pokles aktivace pomalejší, což je důsledek použití uvedené mocninné funkce. Hodnota aktivace užití tak asymptoticky klesá k hodnotě 0 a rychlost tohoto poklesu se zpomaluje. Tento efekt je dokumentován následujícím grafem (viz Graf 6).




**Graf 6: Pokles hodnoty aktivace užití $x_u$ bloku nebo relace při změně průměrné rychlosti degradace paměti $d$ nebo při změně hodnoty počáteční aktivace $x(t_u)$ získané vznikem užití bloku nebo relace.**

Protože je průběh poklesu mocninný, je zvoleno pro osu času $t$ i pro osu hodnoty aktivace užití $x_u(t)$ logaritmické měřítko. V tomto případě má funkce $f = x_u(t)$ lineární charakter, kde hodnota degradace paměti $d$ mění sklon vzniklé funkce

a hodnota počáteční aktivace $x_u(t_u)$ představuje konstantní hodnotu, která je přičítána k aktuální hodnotě $x_u(t)$ a posouvá tak funkci vertikálně. Je však nutné poznamenat, že jak změna hodnoty $d$, tak změna hodnoty $x_u(t_u)$ se projevuje nelineárně na změně sklonu nebo posunu funkce (viz Graf 6).

#### 3.1.3.1 Simulace poklesu hodnoty aktivace

Jak již bylo dříve uvedeno v teoretické části (viz Simulace jako metoda, s. 31), funguje simulační model v krocích. Stav modelu a jeho komponent na konci jednoho kroku simulace je pomocí funkcí a pravidel modelu mapován, a tedy přenesen, na stav na konci dalšího kroku simulace. Díky znalosti pravidel a počátečního stavu lze tak simulovat vývoj stavu děje nebo systému v čase.

V případě mého modelu bylo potřebné odvodit funkci, která dokáže vypočítat hodnotu aktivace užití $x_u$ v čase $t_2$, a tedy hodnotu ${x_u}^{(t_2)}$, na základě následujících známých parametrů a jejich hodnot:

- $t_1$ – čas, ke kterému známe hodnotu aktivace užití $x_u$
- $t_2$ – čas, ke kterému potřebujeme vypočítat hodnotu aktivace užití $x_u$
- ${x_u}^{(t_1)}$ – hodnota aktivace užití platná k času $t_1$
- $t_u$ – čas, kdy vznikla hodnota aktivace užití $x_u$
- $d$ – průměrná rychlost degradace paměti mezi časy $t_1$ a $t_2$

Výsledný vztah byl odvozen tak, že nejdříve byl parametr $d$ nastaven na hodnotu 1 a čas vzniku aktivační hodnoty $t_u$ na 0. Následně byly spočítány hodnoty v jednotlivých časových krocích od vzniku užití (viz Tabulka 4). Potom byly spočítány rozdíly v hodnotách mezi jednotlivými kroky a induktivně byl odvozen následující vztah na základě předpokladu, že potřebujeme spočítat hodnotu aktivace užití $x_u$ v čase $t_2$ (tedy ${x_u}^{(t_2)}$ ) z hodnoty $x_u$ v čase $t_1$ (tedy ${x_u}^{(t_1)}$ ). Potom musí platit, že ${x_u}^{(t_2)} < {x_u}^{(t_1)}$, z čehož plyne, že ${x_u}^{(t_1)}$ musí poklesnout o jistou část ${x_u}^{(t_1)}$. Tato část je dána poměrem časů $t_1$ a $t_2$. Tyto předpoklady vedly k odvození následujícího vztahu.

$$ {x_u}^{(t_2)} = {x_u}^{(t_1)} - {x_u}^{(t_1)} \cdot \left(1 - \left(\frac{t_1}{t_2}\right)\right) $$

**Rovnice 22: Základní výpočet hodnoty aktivace užití $x_u$ k času $t_2$ ze známé hodnoty aktivace užití platné k času $t_1$ pro $d = 1$ a $t_u = 0$.**

Do tohoto vztahu bylo nutné zahrnout faktor degradace paměti. Čím je degradace paměti větší, tím je řádově větší pokles aktivace a naopak. Jelikož poměr $t_1/t_2$ vyjadřuje, na kolik má poklesnout hodnota aktivace užití ${x_u}^{(t_2)}$ vzhledem

k hodnotě aktivace užití ${x_u}^{(t_1)}$, bude tento poměr umocněn faktorem degradace paměti $d$.

$$ {x_u}^{(t_2)} = {x_u}^{(t_1)} - {x_u}^{(t_1)} \cdot \left(1 - \left(\frac{t_1}{t_2}\right)^d\right) $$

**Rovnice 23: Výpočet hodnoty aktivace užití $x_u$ k času $t_2$ ze známé hodnoty aktivace užití platné k času $t_1$ zahrnující degradaci paměti $d$ pro $t_u = 0$.**

Posledním krokem bylo do vztahu zahrnout čas $t_u$ vzniku aktivace užití $x_u$. V případě, kdy $t_1$ vyjadřuje aktuální (nebo minulou) hodnotu aktivace užití $x_u$ a čas $t_2$ vyjadřuje budoucí (nebo aktuální) hodnotu aktivace užití $x_u$, lze čas $t_2$ uvažovat jako aktuální čas simulace, ke kterému je potřeba aktualizovat hodnotu aktivace užití z minulého kroku simulace, a tedy platnou k času $t_1$. Od obou časů je tedy nutné odečíst čas $t_u$ z důvodu, že absolutní rychlost poklesu hodnoty aktivace je na tomto času závislá. Bude tedy provedena korekce časů $t_1$ a $t_2$ o velikost času $t_u$ vzniku hodnoty aktivace užití. Výsledný vztah vypadá následovně:

$$ {x_u}^{(t_2)} = {x_u}^{(t_1)} - {x_u}^{(t_1)} \cdot \left(1 - \left(\frac{t_1 - t_u}{t_2 - t_u}\right)^d\right) $$

**Rovnice 24: Výpočet hodnoty aktivace užití $x_u$ k času $t_2$ ze známé hodnoty aktivace užití platné k času $t_1$ a zahrnující jak degradaci paměti $d$, tak i čas vzniku hodnoty aktivace užití $t_u$.**

Odvozený vztah, který umožňuje iterativně počítat hodnotu aktivace v jednotlivých krocích simulace na základě znalosti hodnoty z předchozího kroku, byl důkladně validován a testován výpočty v tabulkovém procesoru, aby bylo ověřeno, že jeho výsledky jsou za všech okolností shodné se vztahem analyticky popisujícím pokles hodnoty aktivace (viz Rovnice 21).

Nesrovnatelnou výhodou oproti analytickému vztahu je však možnost do výpočtu v simulaci zahrnout nelineární změnu degradace paměti $d$ v závislosti na čase. K časovému úseku mezi časy $t_1 - t_u$ a $t_2 - t_u$ může být provedena korekce základní hodnoty degradace paměti $d$ o jistý koeficient, který bude lépe modelovat děje probíhající při degradaci paměti. Analytický vztah potom umožňuje modelovat pouze průměrnou rychlost degradace paměti.

Kvůli iterativní povaze odvozeného vztahu (viz Rovnice 24) může teoreticky v simulaci docházet k zaokrouhlovacím chybám při výpočtech, které se projeví po velkém počtu kroků odchylkou oproti výsledkům analytického výpočtu. Tento předpoklad byl testován při validaci modelu (viz Tabulka 12, s. 93).

### 3.1.4 Nelineární degradace paměti

Rychlost degradace paměti může být konstantní, jak navrhuje teorie ACT-R, nebo i proměnná tak, aby lépe popsala modelovaný systém a výsledky získané experimenty. Za tohoto předpokladu mohou různé typy hodnot aktivací (bloků nebo relací) degradovat různými rychlostmi v závislosti na čase, který uběhl od vytvoření dané aktivace užití, a podle nastavených parametrů modelu.

Většina modelů paměti typicky rozlišuje kvalitativně různé typy pamětí podle doby uchování informací a rychlosti zapomínání. Prvním typem je paměť krátkodobá (pracovní), která má nízkou úroveň degradace, protože běžně nezapomínáme informace, se kterými aktuálně pracujeme. Tuto roli plní v teorii ACT-R dočasné paměti (*buffers*)[2]. Střednědobá paměť má potom rychlejší degradaci. Posledním typem je paměť dlouhodobá s pomalejší degradací. Degradace se tedy v průběhu času pro danou hodnotu aktivace užití může potenciálně měnit v závislosti na tom, jaký čas uplynul od vytvoření této hodnoty aktivace.

V teorii ACT-R se toto řeší různým nastavením parametru $d$ podle toho, jaký typ paměti modelujeme. Výchozí hodnotou je hodnota $d = 0{,}5$. Pro modelování dlouhodobé paměti se používá hodnota nižší, přičemž nejlepších výsledků v modelech pro heuristické rozhodování lze dosáhnout při hodnotách v intervalu $d = 0{,}32$ až $d = 0{,}34$. U nižších hodnot degradace (paměť pomaleji zapomíná) se dosahované výsledky již zhoršují (viz s. 21).

Při experimentech jak modelovat tuto proměnnou degradaci v čase, kdy je nejdříve její hodnota malá, ale za všech okolností kladná, následně roste, dosáhne maxima a potom asymptoticky klesá k zadané úrovni, byla vlastním experimentováním objevena mocninná funkce, která má přesně požadované chování a je formálně velice snadno vyjádřitelná. Získaný průběh je však komplexní a splňuje výše uvedené předpoklady.

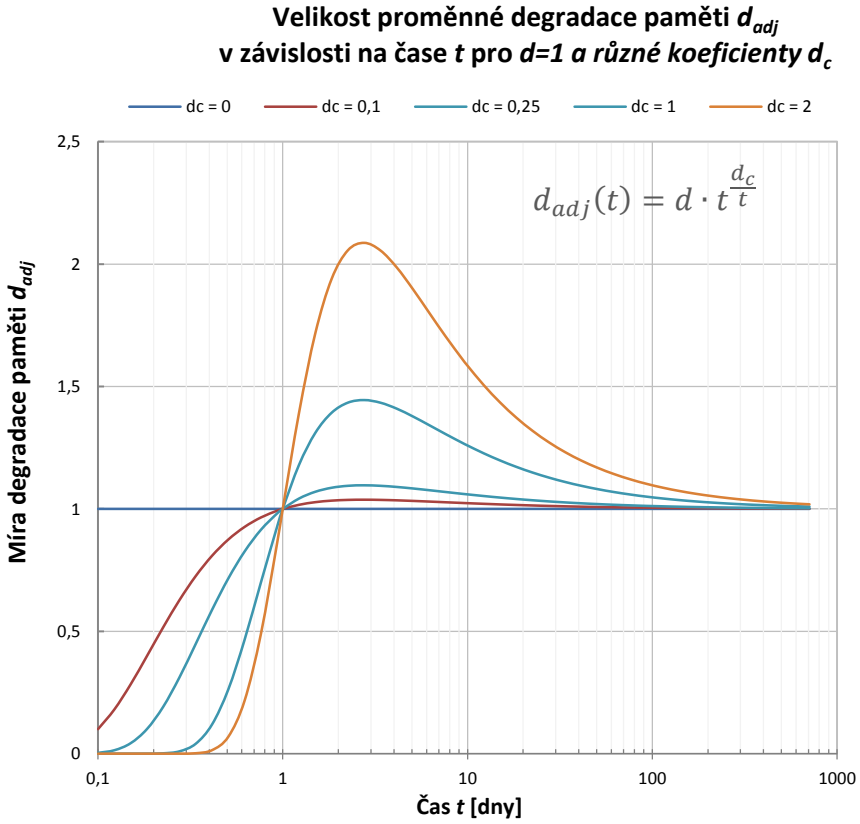
$$d_{adj}(t) = d \cdot t^{\frac{d_c}{t}}$$

**Rovnice 25: Rovnice pro korekci degradace paměti $d$ v závislosti na čase $t$, který uplynul od vytvoření hodnoty aktivace.**

Do vztahu dosazujeme cílovou hodnotu degradace paměti jako parametr $d$, který má stejný význam jako výše uvedený parametr degradace paměti v analytickém

modelu. Výsledná hodnota korigované degradace (*adjusted*) paměti $d_{adj}(t)$ se k této stanovené hodnotě $d$ se zvyšujícím se časem asymptoticky přibližuje shora.

Čas $t$ potom vyjadřuje čas, který uběhl od vytvoření hodnoty aktivace. Pro jeho výpočet tedy platí následující vztah: $t = t_p - t_u$, kde $t_p$ je aktuální čas simulace a $t_u$ je čas vytvoření dané hodnoty aktivace užití. Parametr $d_c$ představuje koeficient (*memory decay coefficient*), který řídí velikost korekce a je zadávaným parametrem modelu. Uvedený vztah je zobrazen níže jako graf funkce pro různé hodnoty koeficientu $d_c$ s pevně nastavenou hodnotou degradace paměti $d = 1$ (viz Graf 7).




**Graf 7: Průběh proměnné nelineární degradace paměti v čase *t*.**

Uvedený vztah umožňuje jak nastavit konstantní hodnotu degradace paměti při volbě $d_c = 0$, se kterou momentálně pracuje ACT-R, tak při kladné hodnotě experimentovat s různým nastavením modelu.

Pro zachování přesnosti výpočtu bude nutné dodržet dostatečně krátký simulační krok pro velmi krátké časy 0,1 až 1 a následně pro časy v rozmezí 1 až 10. Důvodem je, že degradace aktivačních hodnot je aproximována lineárně v rámci simulačního kroku. Dále se hodnota $d_{adj}$ asymptoticky přibližuje ke zvolené hodnotě $d$ udávající degradaci dlouhodobé paměti.

Možnost uvedeným vztahem (viz Rovnice 25) nelineárně ovlivňovat rychlost degradace paměti a tím i rychlost degradace hodnot aktivací užití bloků a relací se při experimentování s vytvořeným modelem ukázala jako klíčová a zásadní. Díky této možnosti lze model kalibrovat podle dat získaných z reálných experimentů a dosáhnout pozoruhodně vysoké míry shody s těmito daty.

### 3.1.5 Aktivace bloků a relací

Všechny hodnoty aktivací bloků i relací degradují časem (viz kapitola Degradace hodnot aktivací, s. 43). V případě použití bloku nebo relace je k této celkové hodnotě aktivace daného bloku nebo relace připsán jistý zisk díky tomuto užití. Tento zisk má potom za následek pomalejší pokles celkové hodnoty aktivace bloku nebo relace v budoucnu.

Podle uvedeného vztahu pro výpočet celkové aktivace bloku (viz Rovnice 13, s. 20) v ACT-R je nutné provést součet všech hodnot aktivací užití, které byly získány minulými užitími daného bloku nebo relace. V případě mého modelu tak musí blok i relace uchovávat záznamy o jednotlivých užitích. Ty obsahují informace o aktuální hodnotě aktivace daného užití $x_u$ a času $t_u$, ke kterému dané užití bloku vzniklo. Tento čas je důležitý pro správný výpočet degradace hodnoty aktivace užití.

$$x^{(t)} = \sum_{u=1}^{n} x_u{}^{(t)}$$

**Rovnice 26: Celková hodnota aktivace bloku nebo relace $x$ k času $t$ je dána součtem hodnot aktivací jednotlivých užití $x_u$ aktualizovaných k tomuto času $t$.**

Pro výpočet celkové hodnoty aktivace bloku $b_i$ nebo relace $s_{ji}$ provedeme součet všech hodnot aktivací užití aktualizovaných k danému času simulace. Následně lze tuto **hodnotu** aktivace převést na **úroveň** aktivace podle vztahu $X = \ln x$, kde $x$ je hodnota aktivace a $X$ potom úroveň aktivace (viz kapitola Hodnoty a úrovně aktivace na straně 41).

Záznamy o užitích bloku nebo relace musí být uloženy v seznamu, který si v modelu blok nebo relace uchovává a jehož hodnoty aktivací jsou synchronizovány k jednomu časovému okamžiku $t$. Tímto okamžikem je potom aktuální (případně minulý) čas simulace.

Při zvýšení hodnoty simulačního čítače dojde k posunu simulace do dalšího časového kroku $t_2$ a je nutné aktualizovat hodnoty aktivací v seznamu užití k tomuto novému času simulace z původního času $t_1$. Z tohoto důvodu je nutné, aby blok také

uchovával čas $x_{u\text{-}time} = t_1$, ke kterému jsou hodnoty aktivací v seznamu užití platné, aby mohla být korektně spočítána, společně s využitím času $t_u$ vzniku aktivace, jejich aktualizace k novému času simulace $t_2$.

Aktualizace hodnot aktivací v seznamu užití probíhá podle vztahu odvozeného dříve (viz Rovnice 24) a má za následek pokles aktivace v čase, a tím i postupnou degradaci této hodnoty. Každá hodnota aktivace v seznamu užití bloku nebo relace potom degraduje samostatně. Součtem těchto degradací potom získáme celkovou degradaci bloku nebo relace.

V případě užití bloku nebo relace, ať už při učení nebo vybavování, vznikne v seznamu užití nový záznam o užití. Seznam užití potom uchovává všechny záznamy o všech předchozích užitích bloku nebo relace. Počáteční hodnotu aktivace užití tohoto nového záznamu lze nastavit jako parametr v modelu, který popisuje zisk aktivace vzniklý daným užitím bloku nebo relace.

### 3.1.6 Časový krok

V prvních verzích modelu byla zvolena jako základní časová jednotka 1 sekunda, která je základní SI fyzikální jednotkou pro měření času. Při této volbě však vycházel příliš vysoký počet potřebných simulačních kroků. Model je tak navržen se záměrem, aby bylo možné časovou délku simulačního kroku měnit podle potřeby. Tímto způsobem je umožněno simulovat změny v modelu, které se odehrávají jak ve zlomcích simulačního kroku, tak i v jeho násobcích při procesu získávání a ztráty aktivace.

Možnost měnit délku simulačního kroku umožnil odvozený vztah (viz Rovnice 24, s. 46) pro pokles aktivace, který je schopen z aktuální hodnoty aktivace v čase $t_1$ spočítat degradovanou hodnotu aktivace v libovolně vzdálený časový okamžik $t_2$. To za předpokladu, že platí $t_2 > t_1$ a míra degradace paměti $d$ má pro tento časový interval konstantní hodnotu. Simulace tedy může za tohoto předpokladu přejít do libovolného času $t_2$, pokud před tímto časem nedošlo k události užití bloku nebo relace. Má-li simulace běžet trvale a reagovat na události z vnějšího prostředí, je nutné omezit maximální délku simulačního kroku na jistou hodnotu.

Při kalibraci modelu se však vyskytl problém. Model byl kalibrován podle dat z experimentů provedených v rámci teorie ACT-R (viz Fan experiment, s. 12) a tyto experimenty probíhají nejčastěji po jednotlivých dnech. Základní zvolený časový krok

však byl v délce jedné sekundy. Model byl kalibrován a získané výsledky byly odpovídající. Bylo tak nutné změnit rozsah zobrazovaného času, kdy základní jednotkou bude jeden den místo jedné sekundy. Časy kratší než jeden den budou potom zlomky simulačního kroku. Díky původní volbě navrhnout model s proměnným a zlomkovým čítáním času simulace bylo snadné upravit základní časový krok.

Pro účely zobrazování je potom nově počet simulačních kroků vynásoben číslem 86400, které vyjadřuje počet sekund v jednom dni. Tímto je získán celkový počet sekund pro výpočet času simulace a jeho zobrazení v pro člověka čitelném formátu (hodiny, dny, měsíce). Pokud chceme posunout čas simulace o jednu minutu, provedeme simulační krok o velikosti přibližně $6{,}95 \cdot 10^{-4}$. Posunu o tři měsíce odpovídá simulační krok o velikosti devadesát. Převody mezi běžnými časovými intervaly, jejich délka v sekundách a velikost v simulačních krocích jsou uvedeny v následující tabulce (viz Tabulka 5).

| Interval | Počet sekund | Simulační krok |
|---|---|---|
| 1 sekunda | 1 | 1,1574E-05 |
| 30 sekund | 30 | 3,4722E-04 |
| 1 minuta | 60 | 6,9444E-04 |
| 5 minut | 300 | 3,4722E-03 |
| 10 minut | 600 | 6,9444E-03 |
| 30 minut | 1 800 | 2,0833E-02 |
| 1 hodina | 3 600 | 4,1667E-02 |
| 3 hodiny | 10 800 | 1,2500E-01 |
| 6 hodin | 21 600 | 2,5000E-01 |
| 12 hodin | 43 200 | 5,0000E-01 |

| Interval | Počet sekund | Simulační krok |
|---|---|---|
| 1 den | 86 400 | 1 |
| 7 dnů | 604 800 | 7 |
| 30 dnů | 2 592 000 | 30 |
| 3 měsíce | 7 776 000 | 90 |
| 1 rok | 31 104 000 | 360 |
| 5 roků | 155 520 000 | 1800 |
| 10 roků | 311 040 000 | 3600 |
| 20 roků | 622 080 000 | 7200 |
| 40 roků | 1 244 160 000 | 14400 |
| 80 roků | 2 488 320 000 | 28800 |

**Tabulka 5: Vztah mezi časovým intervalem, jeho velikostí v sekundách a velikostí časového kroku.**

### *3.1.6.1 Výpočet délky simulačního kroku*

Při nelineární degradaci paměti $d$ (viz Nelineární degradace paměti, s. 47) dochází k proměnné rychlosti degradace hodnoty aktivace užití v čase simulace v závislosti na době, která uplynula od vytvoření této aktivace užití. Tuto dobu lze spočítat pomocí vztahu $t = t_{sim} - t_u$, kde $t_{sim}$ je aktuální čas simulace a $t_u$ je čas, kdy došlo ke vzniku hodnoty aktivace užití.

Díky povaze použitého vztahu pro degradaci paměti (viz Rovnice 25, s. 47) je nutné, aby simulační krok postupoval pro $t$ od 0 do cca 10 v dostatečně krátkých intervalech. Čím budou simulační kroky kratší, tím bude aproximace zadané funkce

přesnější a přesnější budou také výsledky výpočtu degradace hodnoty aktivace k danému času.

Výpočet délky simulačního kroku vychází z předpokladu, že čím vyšší je počáteční hodnota aktivace jakéhokoliv užití, tím rychlejší bude její absolutní pokles v důsledku degradace (viz Degradace hodnot aktivací, s. 43). Výpočet délky simulačního kroku je nepřímo úměrný velikosti této aktivační hodnoty. Při experimentování s modelem byl odvozen následující vztah, kterým získáme dostatečně přesné výsledky při nastavení parametru $d_c$ do hodnoty přibližně 4.

$$t_2 = t_1 + \Delta t^{(t_1)}$$

$$\Delta t^{(t_1)} = \left({x_{max}}^{(t_1)^{-1}} \cdot 0{,}05\right) + 0{,}05$$

**Rovnice 27: Výpočet velikosti simulačního kroku $\Delta t$ pro posun času simulace z času $t_1$ do času $t_2$.**

Uvedený vztah vyjadřuje, že délka simulačního kroku bude vypočítána na základě maximální aktivační hodnoty užití $x_{max}$, která se v modelu pro daný simulační krok nachází. Čím je absolutní velikost této hodnoty vyšší, tím bude krok kratší, aby byl zajištěn přesnější výsledek výpočtu. Hodnota 0,05 přičítaná k výsledku potom vyjadřuje, že krok bude vždy minimálně cca 72 minut dlouhý (1/20 dne).

Výsledek výpočtu je potom v implementaci modelu zaokrouhlován na jedno desetinné místo. Pokud je hodnota $x_{max} = 0$, žádný z bloků ještě nebyl naučen a je zvolen krok $\Delta t^{(t_1)} = 1$. Délka vypočteného kroku musí být vždy menší než maximální délka kroku $\Delta t_{max}$, v opačném případě platí $\Delta t^{(t_1)} = \Delta t_{max}$. Pro účely validace modelu lze velikost simulačního kroku nastavit na konstantní hodnotu.

Při volbě jiných konstant ve výpočtu simulačního kroku je nutné zvolit jiné hodnoty pro kalibraci modelu. Čím jsou hodnoty těchto konstant menší, tím citlivěji model reaguje na změny kalibračních hodnot, avšak jejich absolutní velikost se příliš nemění. Důvodem větší citlivosti je tedy zpětná vazba, která se postupně uplatňuje v jednotlivých krocích výpočtu modelu mezi aktuální velikostí aktivace a velikostí jejího poklesu.

### 3.1.7 Simulační cyklus

Prvním krokem je netypicky posunutí času simulace o zadaný časový krok a k tomuto kroku je aktualizován stav modelu a jeho komponent (bloky a relace). Když je agent, jehož deklarativní paměť modelem simulujeme, v režimu učení (*Learning*), je vybrán jeden z bloků podle typu zvolené strategie

(kalibrace - *Calibration*, náhodná - *Random* nebo nejmenší aktivace - *Least learned*). Pokud je agent v režimu odpočinku (*Relaxing*), není učen ani vybavován žádný blok a tedy bude tento krok přeskočen. Nakonec je aktualizován pohled na data modelu a výstupy v podobě grafů.

Model je navržen tak, aby bylo možné posunout čas o libovolně zvolený časový krok. Tímto krokem může být zlomek základní velikosti časového kroku jednoho dne (například sekunda, minuta, hodina) nebo zároveň násobek základního časového kroku (například 58 hodin, 10 dní, 20 měsíců). Díky tomu model umožní pružně simulovat jak intenzivní období s velkým výskytem událostí v krátkých intervalech, tak období dlouhé nečinnosti, během kterého dochází k zapomínání a dlouhodobému poklesu aktivačních hodnot.

Při posunu simulačního času máme k dispozici dva časy, aktuální čas simulace $t_1$ a cílový čas simulace $t_2$, ke kterému chceme aktuální čas simulace posunout. Pro zahájení posunu si tedy vypočítáme aktuální velikost simulačního kroku $\Delta t^{(t_1)}$ podle dříve uvedeného vztahu (viz Rovnice 27). Dokud platí, že $t_1$ je menší než $t_2 - \Delta t^{(t_1)}$, opakujeme následující postup, pokud chceme zobrazit průběh procesu poklesu aktivačních hodnot. Čas simulace zvyšujeme o aktuální spočítanou velikost simulačního kroku, a tedy platí $t_1 \coloneqq t_1 + \Delta t^{(t_1)}$. Následně provedeme aktualizaci stavu modelu a jeho částí k tomuto novému času $t_1$ a spočítáme novou velikost simulačního kroku $\Delta t^{(t_1)}$ pro tento čas. Uvedený postup se opakuje, dokud platí podmínka $t_1 < t_2 - \Delta t^{(t_1)}$. Na závěr je proveden krok z posledního času $t_1$ do času $t_2$.

Uvedený postup je používán proto, aby bylo možné zobrazit proces přechodu stavu modelu z času $t_1$ do času $t_2$ v grafech zobrazujících stav modelu a v náhledu modelu. Ještě důležitější je potom fakt, že při nelineární degradaci paměti $d$ v čase (pro parametr $d_c > 0$, viz Nelineární degradace paměti, s. 47) je tento výpočet nelineárně závislý na aktuálním čase, a tedy i na velikosti kroku (viz Výpočet délky simulačního kroku, s. 51).

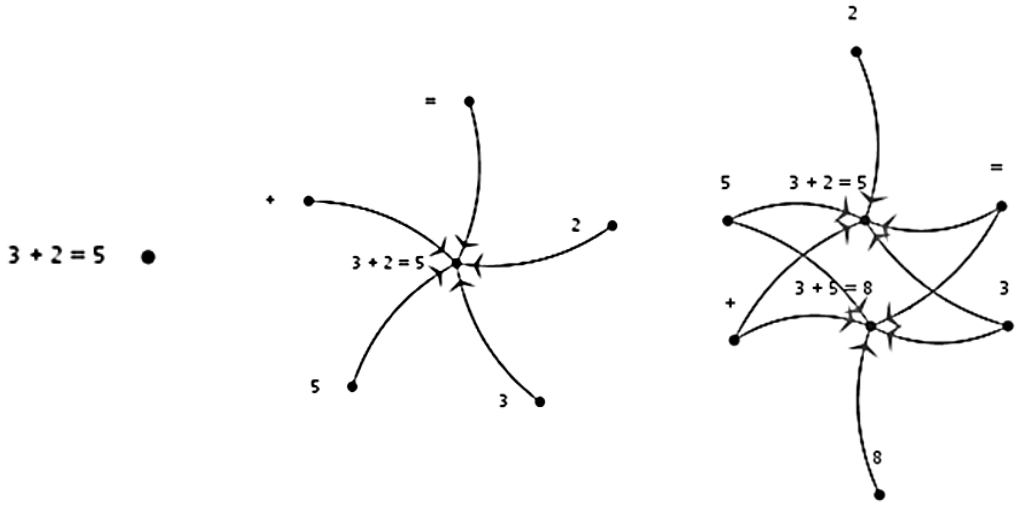
### 3.1.8 Tvorba bloků a relací

Model disponuje dvěma režimy tvorby bloků (*chunks*), které si má agent, jehož paměť modelujeme, zapamatovat a později případně vybavit. V modelu pak bude pro každou faktickou unikátní informaci vytvořen jeden blok. Například pokud

zadáme vytvoření bloků s obsahem `A + B` a `B + A`, potom budou vytvořeny dva bloky. Pokud bychom však zadali vytvoření bloků `+ A B` a `+ A B`, bude vytvořen pouze jeden blok.

První režim tvorby bloků vychází z prvních verzí modelu a představuje tvorbu bloků bez kontextu. V tomto režimu si každý blok nese svůj obsah a není propojován se zdroji aktivace. Jedná se tedy o bezkontextový režim a celková úroveň aktivace bloků *A* se počítá pouze z jejich základní aktivace *B*. Tento režim sloužil u prvních verzí modelu k testování a lze jej stále použít k validaci modelu nebo v případě, že máme v záměru modelovat učení faktů, které spolu nesouvisí.

V kontextovém režimu jsou potom bloky propojovány se zdroji aktivace prostřednictvím relací. Zdroje aktivace tvoří bloky s částí obsahu vytvářeného bloku. Například blok obsahující `3 + 5 = 8` bude vytvořen jako jeden blok obsahující informaci `3 + 5 = 8` a následně, pokud dosud neexistují, budou vytvořeny bloky `3`, `+`, `5`, `=` a `8` představující zdroje aktivace. Pokud zdroj aktivace již existuje, bude nový blok propojen s tímto zdrojem. Jakmile potom vytvoříme nový blok `3 + 2 = 5` v kontextovém režimu, využije tento blok existující bloky jako zdroje aktivace. Celou situaci zobrazuje následující obrázek (viz Obrázek 7).




**Obrázek 7: Ukázka vytvořených bloků a vazeb mezi nimi v implementovaném modelu. Vlevo samostatný blok bez zdrojů aktivace. Uprostřed jeden blok 3+2=5 se svými zdroji aktivace. Vpravo dva bloky 3+2=5 a 3+5=8 sdílející společné zdroje aktivace.**

### 3.1.9 Simulační scénáře

Vytvářet v modelu bloky ručně by bylo z pohledu experimentů příliš zdlouhavé, nicméně je to možné prostřednictvím příkazu pro vytvoření bloku

zadaného v NetLogu v rozhraní *Command Center*. Pro typické úlohy, jako je validace, kalibrace a testování modelu s různým nastavením, jsou k dispozici simulační scénáře (*Scenarios*). Tyto scénáře automaticky vytvoří zadanou situaci, to znamená jak bloky, tak případně v kontextovém režimu relace mezi nimi.

Bylo navrženo několik typů scénářů, které umožní provádět validaci modelu, následně potom kalibraci a experimenty s několika typy úloh a různými strategiemi učení. Dostupné scénáře v navrženém a implementovaném modelu jsou uvedeny v následující tabulce.

| Simulační scénář | Popis |
|---|---|
| **Doporučené využití převážně pro validaci modelu** | |
| **Test základní aktivace** (*Test base activation*) | Model vygeneruje jeden blok. V tomto scénáři lze validovat degradaci základní úrovně aktivace (viz Rovnice 13, s. 20) a další nastavení ovlivňující základní degradaci bloku. |
| **Test kontextové aktivace** (*Test context activation*) | Podobný scénář jako test základní aktivace, k jednomu vygenerovanému bloku jsou však vygenerovány i zdroje aktivace (v implementovaném modelu se jedná o tři zdroje). Lze tak sledovat jak degradaci aktivace bloků, tak degradaci aktivací relací a všechny související parametry se základní i kontextovou aktivací. |
| **Doporučené využití převážně pro kalibraci modelu** | |
| **Fan *a*-*b* dle ACT-R** (*ACT-R fan a-b calibration*) | Scénář vytvoří blok s různým fan číslem (viz Fan experiment, s. 12), aktuálně model obsahuje scénáře generující bloky s fan čísly 1-1, 3-1, 2-2 a 3-3. Tyto scénáře lze použít zejména pro kalibraci modelu. |
| **Doporučené využití převážně pro experimentování s modelem** | |
| **Sada samostatných bloků** (*Set of unrelated chunks*) | Je vytvořen zadaný počet bloků, které nemají žádné zdroje aktivace. Jedná se tedy o samostatné bloky a lze testovat různé strategie učení a vliv na jejich základní a díky absenci kontextu zároveň i na celkovou aktivaci. |
| **ACT-R fan experiment** (*ACT-R fan experiment*) | Je vygenerováno několik bloků s různými fan čísly. Počet bloků má pouze prezentační charakter a není shodný se skutečným experimentem provedeným Johnem Andersonem. |
| **Čísla od 0 do 9 vpřed** (*Numbers 0 to 9 next*) | Vytvoří sadu bloků obsahující informace o pořadí čísel, jedná se o bloky `0 další 1`, `1 další 2`, `2 další 3` a další bloky. |
| **Čísla od 0 do 9 vpřed/vzad** (*Numbers 0 to 9 next/prev*) | Scénář je podobný předchozímu scénáři, ale jsou doplněny bloky obsahující informace o zpětném pořadí čísel. Tedy například bloky `8 předchozí 7`, `7 předchozí 6` a další podobné bloky. |
| **Sčítání *a* to *b*** (*Addition a to b*) | Scénář tvoří sada bloků obsahující faktické informace o sčítání dvou čísel v rozsahu 0 až 5 nebo 0 až 10. |
| **Násobení a to b** (*Multiplication a to b*) | Scénář tvoří sada bloků obsahující faktické informace o násobení dvou čísel v rozsahu 0 až 5 nebo 0 až 10. |

| | |
|---|---|
| **Vlastní scénář** (*Custom*) | Jedná se o prázdný scénář umožňující přímou tvorbu bloků příkazem `create-chunk` a zadávaných pomocí rozhranní *Command Center*. Například lze vytvořit blok obsahující informaci `A B` pomocí `create-chunk "A B"`. |

**Tabulka 6: Přehled navržených a implementovaných scénářů v modelu sloužící k jeho snadnější validaci, kalibraci a k usnadnění prováděných experimentů.**

### 3.1.10 Strategie učení

V modelu byly implementovány základní strategie učení, jež umožní vybrat jeden z bloků, který má být v průběhu simulačního cyklu naučen, a tudíž mu má být vytvořen záznam o užití pro tento blok. Následkem toho je bloku zvýšena hodnota i úroveň aktivace (viz Aktivace bloků a relací, s. 49). Případně lze implementovat a testovat vlastní strategie výběru bloku podle různých kritérií. Pokud probíhá učení intenzivněji, je během jedné iterace simulačního cyklu vybráno a naučeno více bloků, nebo je zvýšen základní zisk aktivace při procesu učení bloku.

Implementovanými strategiemi učení v modelu jsou kalibrace (*Calibration*), náhodná (*Random*), nejméně naučeno (*Least Learned*) a pod limitem vybavení (*Bellow Threshold*). Popis a princip strategií je uveden v tabulce (viz Tabulka 7).

| Strategie učení | Popis |
|---|---|
| Kalibrace (*Calibration*) | Pomocí této strategie lze kalibrovat model pro různé vybrané scénáře a nastavení modelu, pokud potřebujeme mít volbu vybraného bloku pod kontrolou a zobrazit údaje o aktivaci tohoto bloku v grafu aktivace. Tato strategie umožní zobrazit úroveň celkové aktivace $A$, základní aktivace $B$, a kontextové aktivace $C$ vybraného bloku (*Chunk activation level*) v grafu aktivací (viz Úroveň aktivace bloku, s. 18). |
| Náhodná (*Random*) | Vybere zcela náhodně jeden z bloků, který má být naučen. Blok může, ale nemusí být naučen předem. |
| Nejméně naučeno (*Least Learned*) | Vybere blok s nejnižší úrovní aktivace. Nenaučené bloky mají úroveň aktivace 0. V okamžiku, kdy není naučen ani jeden blok, bude vybrán jeden z těchto nenaučených bloků. Pokud některé bloky již byly naučeny, ale jejich úroveň aktivace klesla do záporných hodnot, budou přednostně vybrány tyto bloky. Až v okamžiku, kdy mají všechny bloky úroveň aktivace nad 0, je teprve naučen nový blok. |
| Pod limitem (*Bellow Threshold*) | Strategie je podobná strategii Nejméně naučeno (*Least Learned*) s tím rozdílem, že pro výběr bloků není použita úroveň aktivace 0, ale úroveň nutná jako pro limit vybavení (*Threshold*) bloku (viz Čas vybavení bloku, s. 14). |

**Tabulka 7: Navržené a implementované strategie učení v NetLogo modelu deklarativní paměti.**

Volba bloku se liší podle toho, zda jsme v kontextovém nebo bezkontextovém simulačním scénáři. V bezkontextovém scénáři se nepoužívají relace mezi bloky a je vybrán jeden blok podle zvolené strategie. V kontextovém scénáři se relace mezi

bloky používají. Bloky v kontextovém režimu mají tedy zdroje aktivace podle teorie ACT-R (viz Kontextová úroveň aktivace bloku, s. 23), kterými jsou v navrženém simulačním modelu jiné bloky. Pro výběr bloku v kontextovém režimu jsou tedy použity pouze bloky, které mají zdroje aktivace. Bloky, které tvoří zdroje aktivace, nejsou ve výběru zahrnuty.

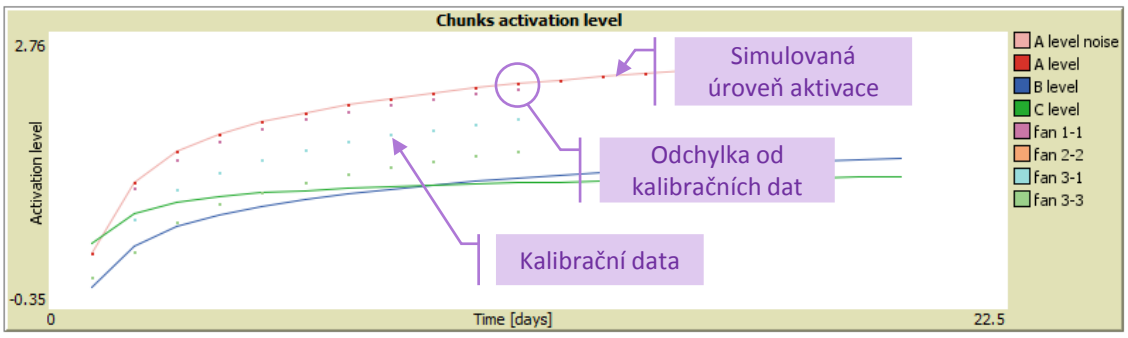
Pro modelování různé frekvence učení bloků (například více bloků v sadě v jeden den) lze v každé iteraci simulačního cyklu vybrat více bloků podle zvolené strategie. Pokud je agent, jehož deklarativní paměť modelujeme, v klasickém režimu učení, je mezi jednotlivými bloky nastaven časový posun přibližně třicet sekund, což je ekvivalentní přibližně hodnotě $34{,}722 \cdot 10^{-5}$ simulačního kroku, aby bylo simulováno rozložení bloků v čase podobně jako v reálném případě. V kalibračním režimu se tento časový posun nerealizuje a frekvence učení je zpravidla nastavena na hodnotu jedna, tj. jeden blok během jedné iterace simulačního cyklu.

### 3.1.11 Validace a kalibrace modelu

Proces validace a kalibrace byl popsán v teoretické části (viz Validace a kalibrace, s. 34). Pro usnadnění těchto úkonů souvisejících s používáním modelu je model vybaven několika procedurami, které usnadní provádění validace a kalibrace. V navrženém a implementovaném modelu je připraveno několik kalibračních sad. Procedury po výběru kalibrační sady snadno umožní nastavit parametry modelu pro základní shodu s kalibračními daty. Následně lze provádět validační a kalibrační experimenty s modelem v různých scénářích.

Kalibrační sady obsahují jak výchozí nastavení modelu pro danou sadu, tak kalibrační data, která jsou platná pro danou sadu a daný typ scénáře (například fan 1-1, fan 1-3 a další). Kalibrační data byla získána ze statistických modelů, které aproximovaly reálná data z experimentů. Jedná se o fan experimenty z let 1985 [21] a 1999 [15] (viz Fan experiment, s. 12).

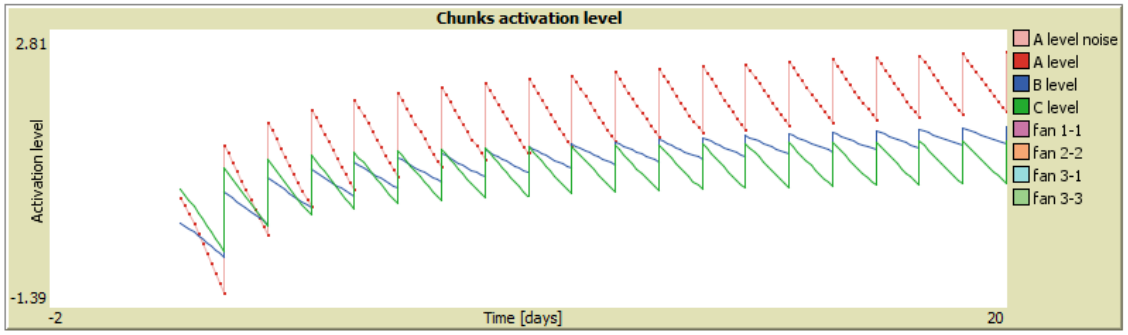
Kalibrační data, představující kalibrované hodnoty aktivace bloku pro daný typ bloku (například fan 1-1, fan 1-3 a další), se zobrazují společně s modelovanými aktivacemi bloku během kalibračního režimu v grafu aktivací bloku. Cílem je nastavit model tak, aby aktivace modelovaného bloku dosáhly stejného průběhu jako kalibrační data pro daný typ bloku. Postup, jak toho dosáhnout, je popsán v kalibrační proceduře (viz Kalibrace modelu, s. 94).




**Graf 8: Idealizovaný průběh aktivace, kdy se nezobrazuje pokles mezi jednotlivými dny.**

Během kalibrace je model přepnut do kalibračního režimu, během kterého nejsou zobrazovány poklesy v hodnotách aktivace mezi jednotlivými dny (viz Graf 8). Tyto poklesy jsou však simulovány, pouze se nezobrazují. Tento pokles zhoršuje čitelnost shody modelované aktivace bloku s kalibračními daty při procesu kalibrace (viz Graf 9), a proto je vhodné jeho zobrazování vypnout.

Je vidět, že simulovaná celková aktivace *A* bloku (červeně) typu fan 1-1 je vyšší, než kalibrační data pro tento typ bloku (viz Graf 8). Zobrazovány jsou jak základní úroveň aktivace *B* (modře), tak kontextová úroveň aktivace *C* (zeleně). Jejich grafickým součtem získáme celkovou úroveň aktivace *A*.




**Graf 9: Zobrazení poklesu aktivace mezi jednotlivými dny více odpovídající skutečnému průběhu úrovně aktivace bloku, degradující mezi jednotlivými dny (shodný průběh bez poklesu viz Graf 8).**

Po dosažení shody s kalibračními daty je možné kalibrační režim vypnout a začít experimenty s modelem. V tomto režimu se v grafu zobrazují průměrné hodnoty aktivací bloků a pokles jejich aktivace mezi jednotlivými užitími.

## 3.2 Implementace modelu v NetLogu

V jednotlivých podkapitolách je nejdříve popsána datová struktura modelu obsahující informace o použitých typech agentů, dále informace o globálních proměnných a proměnných, které mají jednotlivé typy agentů. Následuje přehled funkcí a procedur implementovaných v modelu, jejich význam a členění do sekcí pro

účely modelu. V závěru kapitoly je popsáno grafické uživatelské rozhraní modelu a význam jednotlivých parametrů implementovaného modelu.

Celou implementaci modelu podle uvedeného návrhu tvoří přibližně 1250 řádků základního kódu včetně komentářů. Bez komentářů a prázdných řádků zvyšujících čitelnost má celý kód modelu přibližně 930 řádků. Dalších přibližně 50 až 100 řádků kódu slouží pro vykreslování dat modelu v grafech.

### 3.2.1 Datová struktura a proměnné

Model obsahuje základní typy agentů v podobě bloků `chunks` a relací `relations`. Tyto relace propojují bloky s jejich zdroji aktivace. Zdroje aktivace jsou představovány jinými bloky (viz Základní struktura, s. 41). Není tedy přímo rozlišováno na úrovni struktury modelu, zda daný blok je klasickým blokem, který si potřebujeme vybavit, nebo zdrojem aktivace pro tento blok. Relace mezi bloky jsou orientované. Pomocnými typy agentů jsou potom popisky k blokům `chunks-labels` a spojnice `llabels`, které tyto popisky připojují k těmto blokům (viz Ukázka kódu 1).

```
breed [ chunks chunk ]                          ; bloky (chunks) v paměti
directed-link-breed [ relations relation ]      ; relace mezi bloky

breed [ chunks-labels chunk-label ]             ; popisky k blokům
directed-link-breed [ llabels llabel ]          ; relace pro připojení popisků
```

**Ukázka kódu 1: Typy agentů.**

Agenti v modelu pracují se sadou globálních proměnných uchovávajících informace o celkovém stavu modelu. Některé proměnné slouží pro nastavení vykreslovaného vzhledu agentů. Další proměnné uchovávají dočasné výsledky, které jiní agenti používají ke svým výpočtům. Není proto nutné, aby každý agent musel provádět svůj dotaz a získávat stejný výsledek jako ostatní agenti. Tímto jsou tak sníženy nároky na výpočty modelu.

Speciální pozornost zaslouží proměnná `activation-max-value`, uchovávající maximální hodnotu aktivace užití bloku nebo relace v celém modelu k aktuálnímu časovému kroku a sloužící pro výpočet délky dalšího simulačního kroku, a také proměnná `tick-delta-max` omezující maximální velikost simulačního kroku (viz Výpočet délky simulačního kroku, s. 51). Pro celkový přehled globálních proměnných implementovaného modelu viz Ukázka kódu 2.

```
globals [
  learned-chunks                ; množina již naučených bloků
  learned-chunks-count          ; počet naučených bloků
  view-current-version?         ; typ zobrazované verze (tisk nebo obrazovka)
  view-color                    ; hodnota 1 pro tisk a -1 pro režim obrazovky
  chunk-size-min                ; minimální velikost bloku pro účely zobrazení
  chunk-size-range              ; rozsah velikost bloku pro účely zobrazení
  relation-thickness-min        ; minimální tloušťka relace pro účely zobrazení
  relation-thickness-range      ; rozsah tloušťky relace pro účely zobrazení
  chunk-view-config             ; konfigurace zobrazení bloků
  chunk-view-config             ; konfigurace zobrazení relací
  chunks-a-sum                  ; celkový součet základních aktivací bloků
  chunks-prob-choice-avg        ; průměrná pravděpodobnost volby bloku
  activation-max-value          ; maximální hodnota aktivace pro výpočet kroku
  tick-delta-max                ; maximální délka simulačního kroku
  calibration-data-sets         ; proměnná uchovávající data kalibračních sad
  learner-strategies            ; strategie učení pro modelovaného agenta
]
```

**Ukázka kódu 2: Explicitně deklarované globální proměnné.**

Nejvíce informací v modelu nese agent bloku. Ačkoliv by některé tyto informace mohly být dopočítávány, tak z důvodu zjednodušení a zrychlení implementace modelu jsou tyto hodnoty přímo uchovávány v proměnných. Hodnoty aktivací jsou zapisovány v modelu malými písmeny `a`, `b`, `c`, úrovně aktivace potom velkými písmeny a doplněny slovem *level* (úroveň), tedy pomocí názvů `A-level`, `B-level`, `C-level`. Důvodem pro toto pojmenování je fakt, že NetLogo nerozlišuje velikost písmen v pojmenováních (identifikátorech).

Pro více informací o hodnotách a úrovních aktivací a vzájemných souvislostech viz Hodnoty a úrovně aktivace, s. 41. Pro informace o výpočtech aktivací bloku potom viz Aktivace bloků a relací, s. 49. Přehled všech proměnných bloku a jejich význam viz Ukázka kódu 3.

```
chunks-own [
  learned? content          ; zda byl blok naučen a obsah bloku
  b-u-list b-u-time         ; seznam aktivací užití bloku a čas platnosti seznamu
  a b c                     ; hodnoty aktivací bloků
  A-level-noise             ; celková úroveň aktivace A zahrnující šum
  A-level B-level C-level   ; úrovně aktivací bloku
  C-weight                  ; váha bloku při kontextovém vybavování
  prob-choice               ; pravděpodobnost výběru bloku jako (a / sum a)
  prob-choice-relative      ; relativní pravděpodobnost výběru bloku (a / avg a)
  time-learned time-last-used   ; čas vytvoření a posledního užití bloku
  retrieve-probability          ; pravděpodobnost vybavení bloku
  retrieve-time                 ; čas nutný k vybavení bloku v případě výběru
  learned-count                 ; kolikrát byl blok naučen (vybaven)
]
```

**Ukázka kódu 3: Deklarované proměnné bloků.**

Stav relací je podstatně menší než stav bloků. Lze ho vyjádřit pomocí informace, zda byla či nebyla daná relace již použita. Následuje seznam užití relace `s-u-list` obsahující informace o jednotlivých hodnotách užití dané relace a čas

`s-u-time`, ke kterému jsou aktivace v tomto seznamu platné. Pro celkový přehled všech proměnných relací viz Ukázka kódu 4.

```
relations-own [
  known? used-last-time  ; zda byla relace již použita a čas posledního užití
  s-u-list s-u-time      ; seznam aktivací užití relace a čas platnosti seznamu
  s S-level              ; celková hodnota a úroveň aktivace dané relace
]
```

**Ukázka kódu 4: Deklarované proměnné relací.**

### 3.2.2 Funkce a procedury

Všechny funkce a procedury implementovaného modelu lze rozdělit do několika sekcí, které tvoří v implementaci modelu jistou hierarchii. Ta určuje, jak jsou funkce a procedury navzájem využívány. Funkce a procedury na vyšší úrovni zpravidla používají ty na nižší úrovni. Pro přehled všech funkcí a procedur viz Tabulka 8.

**Tabulka 8: Přehled hierarchie funkcí implementovaného modelu. Nejvyšší úroveň má nejvyšší číslo.**

| Úroveň modelu | Funkce a procedury |
|---|---|
| **6** Řízení simulace (viz s. 77) | `setup`, `go [ time-step ]`<br>`advance-time [ advance-ticks ]`, `tick-delta-advance`<br>`update-model`, `update-view`,<br>`default-global-context`<br>`reset-all-parameters`, `update-max-act-value [ val ]` |
| **5** Kalibrace a kalibrační data (viz s. 76) | `default-calibration`, `calibrate-model`<br>`plot-pen-calibration-data [ pen ]`,<br>`plot-calibration-data [ set-name fan ]`<br>`calibration-set-data [ set-name fan]`, `create-calibration-data-sets` |
| **4** Scénáře a tvorba bloků (viz s. 74) | `scenario-addition-0-5`, `scenario-addition-0-20`,<br>`scenario-multiplication-0-5`, `scenario-multiplication-0-10`,<br>`create-chunks-arithmetic [operator max-x max-y]`<br>`scenario-numbers-order-forward`, `scenario-numbers-order-both`<br>`scenario-anderson-fan-experiment`,<br>`scenario-random-unrelated-chunks`<br>`scenario-fan-1-1`, `scenario-fan-3-1`, `scenario-fan-2-2`,<br>`scenario-fan-3-3`, `scenario-fan-setup`<br>`scenario-test-context-activation`, `scenario-test-base-activation`,<br>`create-chunk [ chunk-string ]` |
| **3** Funkce a procedury relací (viz s. 71) | `relation-update-view`, `view-relation-init`<br>`relation-use [ gain ]`, `relation-update-activation`<br>`fan-diff`, `fan-number`, `relation-init` |
| **2** Funkce a procedury bloků (viz s. 65) | `chunk-learned`, `chunk-use [ gain ]`<br>`chunk-update-view`, `view-chunk-init`, `chunk-update-retrieve`,<br>`chunk-update-activation`, `chunk-update-A-level`,<br>`chunk-update-C-level`, `chunk-update-B-level` |

| | |
|---|---|
| | `chunk-init [ chunk-content ]`<br>`chunk-attach-label`, `chunk-set-label [ caption ]`<br>`chunk-is-activation-source?` |
| **1** Univerzální funkce<br>(viz s. 62) | `update-u-list [ u-list t-use t-update d]`<br>`activation-value-decay [x-t1 t1 t2 tu d]`<br>`string-to-list [ string ]`<br>`time [ ticks-count ]`, `num-to-str` |

Funkce budou v této podkapitole uvedeny v logické návaznosti od těch na nejnižších k těm na nejvyšších úrovních hierarchie. Číslování nadpisů jednotlivých úrovní tak odpovídá číslům úrovní, ve kterých se dané funkce nacházejí. Z důvodu zmenšení prezentovaného rozsahu implementace jsou některé méně důležité komentáře z funkcí a procedur vynechány a jejich zápis je zkrácen.

### *3.2.2.1 Univerzální funkce*

Při vytváření nových bloků v modelu se objevil problém, že je nutné provést rozdělení textového řetězce na jednotlivá slova. Pro následující zpracování je vhodné, aby tato slova byla uložena v seznamu, který lze postupně zpracovat položku po položce. NetLogo [5] obsahuje mnoho základních příkazů pro práci s řetězci (*strings*), nicméně funkce pro jeho rozdělení do seznamu podle zadaného znaku není obsažena. Existují doplňky ve formě rozšiřujících modulů, které umožní použít funkce pro manipulaci s řetězci, například pomocí regulárních výrazů.[8]

Z důvodu vyhnutí se návaznosti na rozšiřující moduly byla implementována vlastní funkce `string-to-list` (viz Ukázka kódu 5). Tato funkce vrací seznam obsahující slova v zadaném řetězci s tím, že oddělovačem pro slova je minimálně jedna mezera. Mezer však může být i více. Mezery na začátku i na konci řetězce jsou potom ignorovány.

```
to-report string-to-list [ string ]
  let pos 0                                ; aktuální pozice při prohledávání
  let add 0                                ; pozice od které se počítá slovo
  let result []                            ; proměnná pro výstupní seznam
  while [ pos < length string ] [
    if (item pos string = " ") [
      if (add < pos) [ set result lput (substring string add pos) result ]
      set add pos + 1
    ]
    set pos pos + 1
  ]
  if (add < pos) [ set result lput (substring string add pos) result ]
  report result
end
```

**Ukázka kódu 5: Funkce `string-to-list` pro převod řetězce na seznam.**

Základní časovou jednotkou pro simulaci je jeden den. Časová délka prováděných simulačních kroků však může být i v řádech sekund. Na druhé straně časový rozsah simulace může potom být od několika dní po mnoho let simulovaného času. Z tohoto důvodu vyvstala tedy nutnost snadnější orientace v aktuálním čase simulace pro uživatele modelu.

NetLogo v základním nastavení neobsahuje funkce pro formátování data a času, proto bylo nutné pro tento účel vytvořit vlastní funkci. Tato funkce umožňuje zobrazit čas simulace v pro člověka čitelném formátu s využitím známých časových jednotek, jako jsou minuty, hodiny, dny a měsíce. Pro zjednodušení funkce byla délka měsíce určena na 30 dní, což může způsobit při časových úsecích mnoha let jistou odchylku zobrazovaného času od reálného kalendářního času.

Navržená funkce `time` v kódu modelu (viz Ukázka kódu 6) má pak jeden parametr vyjadřující převáděný počet dní získaný z hodnoty simulačního čítače s využitím funkce `ticks`. Výstupem je textový řetězec ve formátu `xxY-xxM-xxD xxh:xxm:xxs`, kde za hodnoty `x` jsou dosazeny sekundy (`s`), minuty (`m`), hodiny (`h`), dny (`D`), měsíce (`M`) a roky (`Y`).

Pro implementaci byl zvolen funkcionální přístup. Funkce potom provádí vlastní převod pomocí funkcí `fput`, `map` a `reduce` pro práci se seznamy. Díky tomu bylo možné se vyhnout imperativnímu přístupu s využitím složitého větvení a cyklů. Výsledkem je kratší a přehlednější implementace při zachování plné funkcionality. Případná úprava výstupního formátu řetězce je pak velice jednoduchá, a to pomocí změn parametrů zadaných v pomocných seznamech `intervals`, `units` a `separators`.

```
to-report time [ ticks-count ]                    ; ticks-count je počet dní
  let tc-rem int (ticks-count * (24 * 60 * 60))   ; převod dnů na sekundy
  let intervals [ 60 60 24 30 12 ]                ; pomocné seznamy pro zpracování
  let units [ "Y" "M" "D" "h" "m" "s" ]
  let separators [ "-" "-" " " ":" ":" ""]
  let time-values []
  foreach intervals [   ; převést sekundy do jednotek času (minuty, hodiny, atd.)
    set time-values fput (tc-rem mod ?) time-values
    set tc-rem int (tc-rem / ?)
  ]
  set time-values fput tc-rem time-values
  set time-values map [ num-to-str ? ] time-values        ; doplnit vedoucí nuly
  set time-values (map [ word ?1 ?2 ] time-values units)       ; + jednotky
  set time-values (map [ word ?1 ?2 ] time-values separators)  ; + oddělovače
  report reduce [ word ?1 ?2 ] time-values                ; převést na řetězec
end
```

**Ukázka kódu 6: Funkce `time` pro převod zadaného počtu dní do formátu Y-M-D h:m:s.**

Pro zachování konstantní délky výstupního řetězce a tím zvýšení jeho čitelnosti jsou hodnoty 0 až 9 s jedním řádem doplněny ve funkci `time` po jejich výpočtu vedoucí nulou před dalším zpracováním. K tomuto účelu byla navržena jednoduchá pomocná funkce `num-to-str`, která zadanou hodnotu převede na textový řetězec a v případě potřeby doplní tento řetězec vedoucí nulou.

```
to-report num-to-str [ val ]
  report ifelse-value (val < 10) [ word "0" val ] [ (word val) ]
end
```

**Ukázka kódu 7: Funkce `num-to-str` vrací zadanou hodnotu jako řetězec a pro 0 až 9 doplní vedoucí nulu.**

Výpočet degradace hodnoty aktivace bloků a relací je implementován pomocí funkce `activation-value-decay`. Použitý vztah byl představen při návrhu modelu v kapitole Simulace poklesu hodnoty aktivace na straně 45. Tato funkce přijímá všechny potřebné údaje pomocí svých parametrů. Pro zjednodušení implementace a efektivnější chod modelu má však jeden vedlejší efekt (*side effect*), a tím je použití funkce `update-max-act-value`, která aktualizuje globální proměnnou uchovávající maximální hodnotu aktivace v celém modelu. Koeficient degradace paměti $d$ je také zadáván jako jeden z parametrů a případně je podle nastavení modelu provedena jeho nelineární korekce (viz Nelineární degradace paměti, s. 47).

```
to-report activation-value-decay [x-t1 t1 t2 tu d]
  if (decay-adj?) [    ; je-li vyžadována nelineární korekce degradace paměti
    let t t2 - tu                          ; vypočítat čas pro výpočet korekce
    set d d * t ^ (decay-coef / t)         ; provést korekci zadané degradace
  ]
  let x-t2 x-t1 - x-t1 * (1 - ((t1 - tu) / (t2 - tu)) ^ d)   ; výpočet poklesu
  update-max-act-value x-t2                ; aktualizuje maximální aktivaci
  report x-t2
end
```

**Ukázka kódu 8: Funkce `activation-value-decay` umožňující spočítat pokles hodnoty aktivace.**

Jak bloky, tak relace vyžadují aktualizaci aktivačních hodnot ve svých seznamech užití (viz Aktivace bloků a relací, s. 49). Aktualizací seznamu užití je potom realizována degradace aktivačních hodnot v čase. Pro tento účel byla v modelu implementována funkce `update-u-list` (viz Ukázka kódu 9), která umožní pomocí dříve uvedené funkce `activation-value-decay` právě tuto činnost. Vstupem pro funkci je seznam užití `u-list`. Dále čas `t-use`, ke kterému jsou hodnoty v seznamu užití platné, a čas `t-update`, ke kterému chceme hodnoty aktivací aktualizovat. Zadává se také základní hodnota degradace paměti `d` pro výpočet. Důvodem je, že bloky a relace mohou mít podle nastavení modelu různou rychlost degradace.

```
to-report update-u-list [ u-list t-use t-update d]
  ; aktualizovat hodnoty aktivací z času t-use (t1) na čas t-update (t2)
  let u-list-update
    map [ activation-value-decay (first ?) t-use t-update (item 1 ?) d] u-list
  ; nastavit bloku nově spočítané aktivace
  report (map [ replace-item 0 ?1 ?2 ] u-list u-list-update)
end
```

**Ukázka kódu 9: Funkce `update-u-list` pro aktualizaci seznamu užití bloku nebo relace.**

Manipulace se vstupním seznamem je prováděna pomocí funkcí NetLoga `map`, `first`, `item` a `replace-item` pro práci se seznamy. Seznamy v NetLogu jsou neměnitelné (*immutable*), a tedy nelze měnit hodnoty, které jsou v nich uloženy. Je však možné provádět operace se seznamy pomocí funkcí, které vrací vždy nově vytvořený seznam. Takto získaný seznam lze poté použít jako novou hodnotu pro následné přiřazení do proměnné. Pokud by z výkonnostních důvodů seznamy v modelu nestačily, lze použít v rozšíření NetLoga vektory, které mají měnitelný (*mutable*) obsah.[13]

#### 3.2.2.2 Funkce a procedury bloků

Při implementaci modelu vznikl požadavek na rozlišení, zda je daný blok zdrojem aktivace pro jiné bloky či nikoliv (viz Obrázek 5, s. 41). Pro tento účel byla navržena jednoduchá funkce bloku `chunk-is-activation-source?`. Pokud je zavolána blokem, vrací `true`, pokud má blok nějaké výchozí relace.

```
to-report chunk-is-activation-source?
  report [ count my-out-relations > 0 ] of self
end
```

**Ukázka kódu 10: Funkce `chunk-is-activation-source?` pro zjištění, zda je blok zdrojem aktivace.**

Procedury pro popisky bloku `chunk-attach-label` a `chunk-set-label` řeší popisky hodnot pro bloky, které jsou realizovány jako samostatní agenti. Lze tak nastavit vzdálenost a umístění popisku, což NetLogo se základními dostupnými prostředky neumožňuje. Jedná se tedy o technické řešení, které se přímo netýká problematiky zabývající se obsahem modelu, a proto zde nebude uvedeno.

Při inicializaci modelu je pro výchozí nastavení hodnot bloku použita procedura `chunk-init`. Procedura má parametr `chunk-content`, který nastaví obsah nově vytvářenému bloku. Blok potom čeká v nastaveném výchozím stavu, dokud nebude naučen.

```
to chunk-init [ chunk-content ]
  set learned? false            ; blok zatím nebyl naučen
  set content chunk-content     ; nastavit obsah bloku
  set b-u-list []               ; seznam hodnot aktivací užití bloku je prázdný
  setxy 0 0                     ; pozice bloku je uprostřed pohledu
  chunk-attach-label            ; k bloku je připojen popisek
end
```

**Ukázka kódu 11: Procedura `chunk-init` pro inicializaci.**

Pro aktualizaci celkové základní hodnoty aktivace $b$ a úrovně aktivace $B$ předchozího užití bloku (viz Aktivace bloků a relací, s. 49) je implementována procedura bloku `chunk-update-B-level`. Aktualizace probíhá k současnému času simulace (`ticks`) a využívá k tomu dříve uvedenou funkci `update-u-list`. Následkem aktualizace je degradace aktivačních hodnot bloku zadaným výchozím faktorem degradace bloku `chunk-dec-rate`, který může být při výpočtu pro každou z hodnot aktivace užití upraven (viz Nelineární degradace paměti, s. 47).

Pokud dochází k synchronní aktualizaci všech bloků pro daný simulační krok, jeví se proměnná bloku `b-u-time` jako nepotřebná a bylo by ji možné nahradit globální proměnnou společnou pro všechny bloky. Z důvodů dřívějších verzí modelu byla tato proměnná pro bloky zatím ponechána.

```
to chunk-update-B-level
  if(learned? and ticks > b-u-time) [    ; když byl blok naučen a čas posunut
    ; aktualizovat seznam hodnot aktivací užití bloku
    set b-u-list (update-u-list b-u-list b-u-time ticks chunk-dec-rate)
    set b reduce + map [first ?] b-u-list    ; aktivace je součet aktivací užití
    set B-level ln b                         ; úroveň aktivace bloku B=ln(b)
    set b-u-time ticks                       ; čas platnosti hodnoty aktivace b
  ]
end
```

**Ukázka kódu 12: Procedura `chunk-update-B-level` pro aktualizaci základní aktivace bloku.**

Kontextová aktivace bloku (jak úroveň `C-level`, tak hodnota `c`) se aktualizuje pomocí procedury `chunk-update-C-level`. Výpočet je v implementaci prováděn iterativně, kdy je nejdříve provedena aktualizace aktivace relace a následně je tato aktualizovaná hodnota použita pro výpočet části kontextové aktivace a přičtena k celkové nové úrovni kontextové aktivace bloku.

Na závěr je na novou spočítanou úroveň nastavena kontextová úroveň aktivace bloku i hodnota této aktivace (viz Hodnoty a úrovně aktivace, s. 41). Pro požadovaný výsledek je nutné před výpočtem nastavit váhy $W_j$ jednotlivých bloků, které jsou zdrojem aktivace (viz Kontextová úroveň aktivace bloku, s. 23).

```
to chunk-update-C-level
  let C-new 0                    ; nová hodnota kontextové aktivace
  ask my-in-relations [          ; iterovat přes všechny relace ze zdrojů aktivace
    relation-update-activation     ; aktualizovat stav aktivace relace
    set C-new (C-new + ([ C-weight ] of end1) * S-level)
  ]
  set C-level C-new              ; nastavit novou úroveň aktivace
  set c exp C-new                ; vypočítat hodnotu aktivace c=e^(C-new)
end
```

**Ukázka kódu 13: Procedura `chunk-update-C-level` pro aktualizaci kontextové aktivace bloku.**

Pro výpočet celkové úrovně `A-level` a hodnoty `a` aktivace bloku se používá procedura `chunk-update-A-level`. Před vlastním výpočtem aktivace je provedena aktualizace základní a kontextové aktivace bloku podle dříve uvedených procedur. Pokud má být modelován také šum v aktivačních hodnotách, je hodnota `A-level-noise` doplněna o vygenerovanou hodnotu šumu podle nastavení modelu. Pro více informací viz Rovnice 7, s. 17, v kapitole Pravděpodobnost vybavení bloku.

```
to chunk-update-A-level
  chunk-update-B-level             ; aktualizovat základní úroveň aktivace bloku
  chunk-update-C-level             ; aktualizovat kontextovou úroveň aktivace
  set A-level B-level + C-level    ; celková úroveň aktivace bloku
  set a exp A-level                ; celková hodnota aktivace bloku a=e^(A-level)
  set A-level-noise A-level        ; aktivace zahrnující šum aktivačních hodnot
  if (activation-noise?) [         ; když má být zahrnut šum provést výpočet šumu
    let sd-noise (s-activation-noise * chunk-retrieve-treshold) / sqrt(3)
    set A-level-noise ( A-level + random-normal 0 sd-noise)
  ]
end
```

**Ukázka kódu 14: Procedura `chunk-update-A-level` pro aktualizaci celkové aktivace bloku.**

Jako rozhraní oddělující procedury na vyšších úrovních implementace od procedur této úrovně je navržena procedura `chunk-update-activation`. Jejím úkolem je aktualizovat aktivaci bloku, pokud byl blok v minulosti naučen, a tedy použit. Tuto informaci uchovává proměnná bloku `learned?`.

```
to chunk-update-activation
  if (learned?) [ chunk-update-A-level ]   ; když byl blok naučen aktualizovat
end
```

**Ukázka kódu 15: Procedura `chunk-update-activation` pro aktualizaci aktivace bloku.**

Hodnoty jako jsou pravděpodobnost vybavení bloku a čas, jak dlouho toto vybavení bude trvat, jsou vypočteny na základě celkové aktivační úrovně *A* bloku. Nejdříve tedy musí být aktualizována tato úroveň a až potom lze spočítat tyto údaje (viz kapitoly Čas vybavení bloku a Pravděpodobnost vybavení bloku na straně 14 a 16). Pro tento výpočet slouží procedura `chunk-update-retrieve`. Procedura následně z důvodu efektivity používá globální proměnné `chunks-a-sum`

a `chunks-prob-choice-avg`, aby každé volání nemuselo provádět svůj výpočet. Je nutné tedy hodnoty těchto proměnných aktualizovat před voláním této procedury.

```
to chunk-update-retrieve
  if (learned?) [
    if (matching-mode = "exact") [          ; režim přesné shody
      let t chunk-retrieve-treshold           ; limit pro vybavení
      let s-noise s-activation-noise          ; aktivační šum
      set retrieve-probability 1 / ( 1 + exp( - (A-level - t) / s-noise))
      let F-time time-scaling-const           ; časové měřítko
      let f-act activation-scaling-const      ; aktivační měřítko
      set retrieve-time F-time * exp (- A-level-noise * f-act)
    ]
    set prob-choice a / chunks-a-sum       ; pravděpodobnosti výběru dle aktivace
    set prob-choice-relative prob-choice / chunks-prob-choice-avg
  ]
end
```

**Ukázka kódu 16: Procedura `chunk-update-retrieve` aktualizuje údaje týkající se vybavení bloku.**

Pro konfiguraci vzhledu bloku se používá procedura `view-chunk-init` spuštěná při inicializaci modelu. Konfigurace je v podobě seznamu zapsána do globální proměnné `chunk-view-config`. V ukázce je uveden pouze výběr z celkového počtu šestnácti hodnot, které lze u bloku sledovat. Hodnoty konstant pro korekci zobrazení byly zjištěny experimentováním s modelem.

```
to view-chunk-init
  set chunk-view-config [
    ; konstanta pro korekci zobrazení, zda provést korekci a typ hodnoty
    [ 0.20 false "Content" ] [ 0.20 false "An | Activation level (noise)" ]
    [ 1.00 true "a | Total activation value" ] [ 0.15 true "Choice probability" ]
    ...
    [-1.50 true "Retrieve time" ] [ 1E-2 false "Time since first learned"]
    [-0.20 true "Time since last use" ] [ 1E-3 false "Learning count" ]
  ]
end
```

**Ukázka kódu 17: Procedura `view-chunk-init` pro konfiguraci vzhledu bloků v pohledu modelu.**

Poslední procedurou, která se zabývá procesem aktualizace stavu bloku, je procedura `chunk-update-view` (viz Ukázka kódu 18), která má za úkol zobrazit stav bloku v náhledu modelu v závislosti na typu vybrané hodnoty bloku. Aktuální hodnoty bloku jsou jako seznam spojeny s konfigurací zobrazení pomocí funkce `map` a následně je povedeno filtrování vzniklého seznamu pomocí funkce `filter`. Tím je získán maximálně jeden záznam (`x`) obsahující aktuální konfiguraci zobrazení pro hodnotu vybraného typu (`chunk-value-display`), v opačném případě je použito výchozího zobrazení.

Pokud je zvolena hodnota číselného typu, je tato hodnota podle konfigurace `conf` daného typu hodnoty mapována na hodnotu pro zobrazovací účely `view-val`. Vzniklá hodnota je následně pomocí logistické (*sigmoid*) funkce $1/(1+e^{-x})$

a s využitím měřítka z konfigurace `view-scale` mapována na odstín barvy a číslo vyjadřující velikost bloku `size`. Pokud je vybrána hodnota nečíselného typu (například hodnota typu `"Content"`), je jako výchozí hodnota pro zobrazení zvolena hodnota základní aktivace bloku s případným šumem `A-level-noise`.

U nenaučených bloků je nastaven výchozí vzhled. Naučené bloky, které jsou zdroji aktivace, jsou zelené (`turquoise`), ostatní bloky modré (`blue`). Bloky, které byly použity do jednoho dne od aktuálního času simulace, používají pro zobrazení kruhovou značku se žlutou tečkou (tvar `"block-used"`), ostatní bloky potom kruhovou značku bez tečky (tvar `"block-default"`). Pokud je zobrazovaná hodnota u bloku číselná, je zaokrouhlena funkcí `precision` pro účely zobrazení na zadaný počet míst podle nastavení `view-val-precision`. Popisky pro bloky jsou jako samostatní agenti, aby bylo možné nastavit vzdálenost a umístění popisku, a pro nastavení textu popisku se používá tedy procedura `chunk-set-label`.

```
to chunk-update-view
  let t ticks                     ; aktuální čas simulace
  let chunk-values (list content A-level-noise A-level B-level C-level C-weight
                         a b c prob-choice prob-choice-relative
                         retrieve-probability retrieve-time
                         (t - time-learned) (t - time-last-used) learned-count)
  let x filter [item 2 ? = chunk-value-display]
        (map [ lput ?2 ?1  ] chunk-view-config chunk-values)

  let val 0                       ; zobrazovaná hodnota u bloku
  let view-val 0                  ; hodnota použita pro zobrazení bloku
  let view-scale 1                ; měřítko hodnoty použité pro zobrazení
  if-else (length x > 0) [
    let conf item 0 x               ; konfigurace vzhledu jako první položka x
    let do-ln? item 1 conf          ; zda má být hodnota logaritmována
    set view-scale item 0 conf      ; koeficient měřítka hodnoty
    set val item 3 conf             ; velikost hodnoty
    set view-val ifelse-value (is-number? val and do-ln? and val > 0)
                              [ ln val ] [ val ]
  ][ set val "" ]                   ; neznámá položka - nic se nezobrazí
  if-else (not learned?) [        ; nenaučený (neznámý blok) - základní vzhled
    set shape "block-default"
    set color (gray + 2.5 * view-color)
    set label-color (gray + 1 * view-color)
    set size chunk-size-min
  ][
    let chunk-color ifelse-value (chunk-is-activation-source?)
                        [ turquoise ][ blue ]
    set shape ifelse-value ((t - time-last-used) < 1)
                        [ "block-used" ][ "block-default" ]
    let viz-value ifelse-value (is-number? val) [ view-val ] [ A-level-noise ]
    ;; výpočet barvy a velikosti bloku používá logistickou funkci 1/(1+e^-x)
    let ch-cb (chunk-color + 2 * view-color)      ; základní barva
    let ch-cr 3                                   ; barevný rozsah odstínů
    set color
      (ch-cb - ch-cr * (1 / (1 + exp (- viz-value * view-scale))) * view-color)
    let r-sm chunk-size-min                       ; minimální velikost
    let r-sr chunk-size-range                     ; rozsah velikosti
    set size r-sm + r-sr * (1 / (1 + exp (- viz-value * 2 * view-scale)))
  ]
```

```
  ; je-li zobrazovaná hodnota číselná - zaokrouhlit pro účely zobrazení
  if (is-number? val) [ set val precision val view-val-precision ]
  chunk-set-label ifelse-value (show-values?) [ val ] [ "" ]
end
```

**Ukázka kódu 18: Funkce `chunk-update-view` aktualizuje zobrazení bloku v náhledu modelu.**

Při užití bloku je vytvořen v daném bloku záznam o jeho užití (viz Aktivace bloků a relací, s. 49). Toto v modelu zajišťuje procedura `chunk-use`, které se předává počáteční aktivační hodnota tohoto užití pomocí parametru `gain`. Díky tomu lze případně realizovat různé zisky aktivační hodnoty užitím bloku (viz Sub-symbolické učení, s. 20).

```
to chunk-use [ gain ]
  let use-log (list gain (ticks - 1))            ; vytvořit záznam o užití
  if (b-u-time > 0) [ chunk-update-B-level ]     ; aktualizovat seznam užití
  set b-u-list lput use-log b-u-list             ; přidat záznam do seznamu užití
  set b ifelse-value (b-u-time = 0) [ gain ] [ b + gain ]  ; hodnota aktivace
  set B-level ln b;                                        ; úroveň aktivace
  set b-u-time ticks                        ; čas platnosti základní aktivace
  set time-last-used ticks                  ; nastavit čas posledního užití
  update-max-act-value gain                 ; provést aktualizaci max. aktivace
  set learned-count learned-count + 1       ; zvýšit počítadlo užití bloku
end
```

**Ukázka kódu 19: Procedura `chunk-use` provede akci užití bloku a tím zvýšení jeho základní aktivace.**

Pro reprezentaci procesu učení daného bloku je v modelu implementována procedura `chunk-learned`. Pokud se jedná o nový, nenaučený blok, je tento blok převeden mezi naučené bloky a do množiny naučených bloků.

Je-li zvolena možnost použití kontextové nápovědy, je spočítán počet zdrojů aktivace a těmto zdrojům je nastavena kontextová váha (viz Kontextová úroveň aktivace bloku, s. 23). U bloků, které jsou zdroji aktivace, je provedeno také jejich naučení, a tím je zvýšena jejich úroveň aktivace. Následně získají zisk aktivace i relace, které propojují aktuální blok s těmito zdroji aktivace. Velikost zisku hodnoty aktivace relace řídí nastavený parametr modelu `relation-gain`. Pokud je kontextová nápověda vypnuta, nejsou uvedené akce vykonány.

Na závěr je provedeno užití vlastního bloku, který je učen s nastaveným ziskem `chunk-gain`. Jak hodnota aktivace zisku bloku, tak případně zisk relace je násoben koeficientem `learning-intensity`, který vyjadřuje intenzitu procesu učení, která se odráží ve vyšší hodnotě zisku. Tímto koeficientem lze modelovat relativní intenzitu učení podle nastavené kalibrace modelu.

```
to chunk-learned
  if (not learned?) [                  ; když je blok zatím neznámý
    set learned? true                                    ; blok byl naučen
    set time-learned ticks                               ; čas naučení bloku
    set learned-chunks turtle-set chunks with [learned?]  ; přidat mezi naučené
  ]
  if (use-context-cue?) [              ; když má být použita kontextová nápověda
    let n count in-relation-neighbors   ; zjistit počet zdrojů aktivace
    ask in-relation-neighbors [         ; pro každý zdroj aktivace
      set C-weight 1 / n                  ; nastavit váhu bloku
      chunk-learned                       ; naučit blok zdroje aktivace
    ]
    ask my-in-relations                   ; zisk aktivace pro relace
      [ relation-use relation-gain * learning-intensity ]
  ]
  chunk-use chunk-gain * learning-intensity   ; zisk aktivace užitím bloku
end
```

**Ukázka kódu 20: Procedura `chunk-learned` realizující zisk aktivace bloku a souvisejících komponent.**

### 3.2.2.3 Funkce a procedury relací

Implementace procedur relací je podstatně jednodušší než u bloků, které mají podobnou strukturu jako procedury bloků. První uvedenou procedurou je procedura pro inicializaci relace `relation-init` použitá pro inicializování nově vytvářených relací při inicializaci modelu.

```
to relation-init
  set known? false            ; nová (neznámá) relace ještě nebyla použita
  set s-u-list []             ; seznam užití relace je prázdný
end
```

**Ukázka kódu 21: Procedura `relation-init` pro inicializaci počátečního stavu nové relace.**

Ve dvou procedurách je nutné provádět výpočet fan čísla bloku, ze kterých daná relace vychází. Jedná se o bloky, které jsou zdrojem aktivací pro jiné bloky v kontextu a jsou tedy i zdrojem relací (viz Obrázek 2, s. 13). Pro tento účel byla navržena funkce `fan-number`, která spočítá počet použitých odchozích relací pro zadaný blok, a tedy tím i aktuální fan číslo daného bloku.

```
to-report fan-number [ chunk ]
  let fan 0
  ask chunk [ set fan count my-out-relations with [ known? ] ]
  report fan
end
```

**Ukázka kódu 22: Funkce `fan-number` pro výpočet Fan čísla zadaného bloku.**

U bloků, které mají asymetrická fan čísla (například bloky typu fan 1-3 nebo fan 3-1), je pro korekci aktivace relace nutné z důvodu této asymetrie spočítat velikost korekce aktivace při jejím zisku i degradaci. Funkce `fan-fiff` spočítá, jaký je rozdíl mezi fan čísly pro danou relaci. Toto číslo lze následně použít pro uvedenou korekci.

```
to-report fan-diff [ relation ]
  let total 0
  ask [end2] of relation [
    ask in-relation-neighbors [
      set total total + count my-out-relations with [ known? ] ] ]
  let n-out1 0
  ask [end1] of relation [ set n-out1 count my-out-relations with [ known? ] ]
  let n-out2 total - n-out1
  report abs (n-out2 - n-out1)
end
```

**Ukázka kódu 23: Funkce `fan-diff` vrací rozdíl fan čísel zdrojů aktivace bloku, kde je relace zdrojem.**

Aktualizace aktivace relace se provádí podobně jako u bloků pomocí procedury `relation-update-activation`. Ekvivalentem pro bloky je procedura `chunk-update-B-level` (viz Ukázka kódu 12, s. 66). Na rozdíl od procedury bloku je v proceduře pro relaci nejdříve vypočteno fan číslo `fan` bloku, ze kterého aktualizovaná relace vychází

Následně je provedena podle `fan` hodnoty vypočítaná korekce základní degradace relací `relation-deg-rate`. Implementovaný vztah byl objeven na základě zkoumání výsledků simulací a návrhů jednotlivých verzí modelu a po kalibraci modelu nejjednodušeji a nejpřesněji popisuje výsledky získané v reálných experimentech. Poté je aktualizován s využitím korigované hodnoty degradace relace `d` seznam jednotlivých užití relace `s-u-list`. Na závěr je aktualizována celková hodnota `s` a úroveň `S-level` aktivace užití relace a je nastaven čas `s-u-time`, ke kterému je tato aktivace platná.

```
to relation-update-activation
  if(known? and not empty? s-u-list and ticks > s-u-time) [
    let fan fan-number end1           ; výpočet fan čísla zdrojového bloku relace
    let diff fan-diff self            ; rozdíl fan čísel bloku end1
    let d relation-dec-rate * ( fan-decay-coef ^ (fan - 1) )  ; korekce degradace
    set d d * (diff-decay-coef) ^ (diff)            ; korekce rozdílem fan čísel
    set s-u-list (update-u-list s-u-list s-u-time ticks d)  ; aktualizace užití
    set s reduce + map [first ?] s-u-list       ; aktualizovat hodnotu aktivace
    set S-level ln s                            ; aktualizovat úroveň aktivace
    set s-u-time ticks                          ; nastavit čas aktualizace
  ]
end
```

**Ukázka kódu 24: Procedura `relation-update-activation` pro aktualizaci aktivace relace.**

Užití relace je velmi podobné užití bloku (viz Ukázka kódu 19, s. 70). Tato akce je realizována voláním procedury `relation-use`. Prvním krokem je výpočet fan čísla bloku, ze kterého relace vychází. Základní zisk hodnoty aktivace relace je potom korigován identickým mocninným vztahem s využitím fan čísla stejně, jako je korigován faktor degradace relace paměti v předchozí proceduře.

Pokud má blok nenulovou hodnotu `s-u-time`, znamená to, že již byla relace v minulosti někdy užita, a tedy seznam užití není prázdný a je nutné jej aktualizovat. Následně je vytvořen záznam o užití relace a tento záznam je vložen do aktualizovaného seznamu užití. Dále zbývá aktualizovat celkovou hodnotu `s`, úroveň `S-level` aktivace užití relace a časy, ke kterému je toto užití platné. Na závěr je nutné případně aktualizovat maximální hodnotu aktivace v modelu použitou pro výpočet velikosti simulačního kroku (viz Výpočet délky simulačního kroku, s. 51).

```
to relation-use [ gain ]
  if(not known?) [ set known? true ] ; zařadit relaci mezi použité (naučené)
  let fan fan-number end1            ; výpočet fan čísla zdrojového bloku relace
  let diff fan-diff self             ; rozdíl fan čísel bloku end1
  let s-gain gain * ((fan-gain-coef) ^ (fan - 1))     ; korekce vlivem fan čísla
  set s-gain s-gain * (diff-gain-coef) ^ (diff)       ; korekce rozdílem fan čísel
  let use-log (list s-gain (ticks - 1) )              ; vytvořit záznam o užití
  if (s-u-time > 0) [ relation-update-activation ]    ; aktualizace seznamu užití
  set s-u-list lput use-log s-u-list                  ; vložit záznam do seznamu
  ; když je relace nová je aktivace rovna zisku, jinak základní + zisk
  set s ifelse-value (s-u-time = 0) [ s-gain ] [ s + s-gain ]
  set S-level ln s;                   ; aktualizovat úroveň aktivace relace
  set s-u-time ticks                  ; nastavit čas aktualizace aktivace
  set used-last-time ticks            ; čas posledního použití relace
  update-max-act-value s-gain         ; aktualizace globální hodnoty aktivace
end
```

**Ukázka kódu 25: Procedura `relation-use` provede akci užití relace a tím zvýšení její základní aktivace.**

Zobrazení relací je řešeno stejně jako u bloků dvěma procedurami. Procedura `view-relation-init` nastaví konfiguraci zobrazení při inicializaci modelu. Následně potom procedura pro aktualizaci stavů relací `relation-update-view` provede převod vybrané hodnoty pro zobrazení na barvu a tloušťku relace v pohledu modelu. Ukázka této procedury je z důvodu podobné části v dříve uvedené proceduře zkrácena. Pro více informací viz Ukázka kódu 17 a Ukázka kódu 18 na stranách 68 a 70.

```
to view-relation-init
  set relation-view-config [
    ; konstanta pro korekci zobrazení, zda provést korekci a typ hodnoty
    [ 0.25 false "S | Activation level" ] [ 0.25 true "s | Activation value" ]
    [-4E-2 false "Time since last use" ]]
end
```

**Ukázka kódu 26: Procedura `view-relation-init` pro konfiguraci vzhledu relací v pohledu modelu.**

```
to relation-update-view
  let relation-values (list S-level s (ticks - used-last-time))  ; hodnoty relace
  let x filter [ item 2 ? = relation-value-display ]             ; výběr hodnot
        (map [ lput ?2 ?1  ] relation-view-config relation-values)
  ...
  if-else (s = 0) [                          ; nepoužitá relace - výchozí vzhled
    set thickness relation-thickness-min          ; minimální tloušťka
    set label-color (gray + 3 * view-color)       ; šedá barva pro popisek
    set color (gray + 3 * view-color)             ; šedá barva pro tvar relace
    set shape "activation"                        ; výchozí tvar
  ][
```

```
    set shape ifelse-value ((ticks - used-last-time) < 1) ; použitá relace
              [ "activation-used" ][ "activation" ]       ; do 1 dne má jiný tvar
    let l-c-b (brown + 1 * view-color)      ; základní barva popisku
    let c-r 2.5                             ; rozsah barev popisku a relace
    set label-color                         ; výpočet barvy sigmoid funkcí
      (l-c-b - c-r * ( 1 / ( 1 + exp ( - view-val * view-scale))) * view-color)
    let c-b (orange + 4 * view-color)       ; základní barva relace
    set color
      (c-b - c-r * ( 1 / ( 1 + exp ( - view-val * view-scale))) * view-color)
    let th-min relation-thickness-min       ; minimální tloušťka relace
    let th-range relation-thickness-range   ; rozsah tlouštěk relace
    set thickness
      th-min + th-range * ( 1 / ( 1 + exp ( - view-val * 1.5 * view-scale)))
  ]
  if (is-number? val) [ set val precision val view-val-precision ]
  set label ifelse-value (show-values?) [ val ] [ "" ]
end
```

**Ukázka kódu 27: Procedura `relation-update-view` aktualizuje zobrazení relací v náhledu modelu.**

#### 3.2.2.4 Scénáře a tvorba bloků

Na této úrovni jsou implementovány funkce a procedury zabývající se vytvářením bloků a relací mezi nimi podle předem nastavených scénářů (viz Simulační scénáře, s. 54). Základní procedurou je procedura `create-chunk`, která vytvoří blok se zadaným obsahem `chunk-string`.

Implementován je kontextový a bezkontextový režim (viz Tvorba bloků a relací, s. 53). V kontextovém režimu jsou vyhledány bloky obsahující slova ze zadaného řetězce. Pokud bloky s daným obsahem existují, je vytvořena pouze relace z daného bloku k nově vytvářenému bloku. Pokud zdroj aktivace jako jiný blok neexistuje, je nejdříve vytvořen a inicializován nový blok jako zdroj aktivace a následně je vytvořena relace z tohoto bloku.

```
to create-chunk [ chunk-string ]
  if(not any? chunks with [ content = chunk-string ]) [  ; pouze unikátní blok
    create-chunks 1 [                                   ; vytvořit nový blok
      chunk-init chunk-string                           ; nastavit mu zadaný obsah
      let context-chunks string-to-list chunk-string    ; rozdělit obsah na slova
      if (context-mode? and (length context-chunks > 1)) [
        let activation-chunk nobody                     ; zdroj aktivace
        foreach context-chunks [                        ; pro každé slovo obsahu
          set activation-chunk chunks with [content = ?]  ; vyhledat blok zdroje
          if-else (any? activation-chunk) [             ; když existuje
            ask activation-chunk [ create-relation-to myself [ relation-init ] ]
          ][                                            ; jinak neexistuje
            hatch 1 [                                   ; vytvoř nový zdroj
              chunk-init ?                              ; inicializuj
              create-relation-to myself [ relation-init ]   ; vytvoř relaci
            ]
          ] ;; rychle rozmístit stávající použité nebo nové zdroje aktivace
          repeat 50 [ layout-spring chunks relations 0.06 8 1 ]
        ]  ;; rozmístit řádně bloky
        repeat 1500 [ layout-spring chunks relations 0.06 8 1 ]
  ]]]
end
```

**Ukázka kódu 28: Procedura `create-chunk` vytvoří nový blok a propojí jej relacemi se zdroji aktivace.**

Proceduru pro vytváření bloku potom využívají všechny procedury pro tvorbu scénářů (viz Tabulka 6, s. 56). Jsou implementovány tři typy scénářů, a to validační, kalibrační (viz Validace a kalibrace, s. 34) a experimentální. Validační scénáře slouží pro formální testování korektnosti modelu. Jedná se o proceduru `scenario-test-base-activation` pro testování základní aktivace a o proceduru `scenario-test-context-activation` testující kontextovou aktivaci.

```
to scenario-test-context-activation
  set context-mode? true
  create-chunk "fireman thanked chef"
  ask chunks [ setxy 0 0 ]
  repeat 3000 [ layout-spring chunks relations 0.1 10 1 ]
end

to scenario-test-base-activation
  set context-mode? false
  create-chunk "fireman"
end
```

**Ukázka kódu 29: Procedury pro validační scénáře modelu.**

Kalibrační scénáře jsou tvořeny připravenými bloky a relacemi mezi nimi, které společně tvoří bloky s různým fan číslem (viz Obrázek 2, s. 13). Aby fan efekt fungoval, je nutné všechny relace nastavit jako známé (`known? = true`). Kalibrační scénáře je nutné vytvářet v kontextovém režimu (`context-mode? = true`).

```
to scenario-fan-3-3                ; scénář fan 3-3
  create-chunk "A B"    create-chunk "B C"    create-chunk "B D"
  create-chunk "A E"    create-chunk "A F"    scenario-fan-setup
end

to scenario-fan-1-1                ; scénář fan 1-1
  create-chunk "A B"    scenario-fan-setup
end

to scenario-fan-setup
  ask relations [ set known? true ]      ; nastavit relace jako známé
  ask chunks [ setxy 0 0 ]
  repeat 15000 [ layout-spring chunks relations 0.1 6 1 ]
  ask chunks [ setxy (xcor * 0.8) (ycor * 0.8) ]
end
```

**Ukázka kódu 30: Vybrané procedury pro tvorbu scénářů obsahující bloky s různým fan číslem.**

Scénáře pro experimenty jsou vytvářeny tak, že je nejdříve složen řetězec, který následně tvoří obsah vytvářeného bloku. Různé řetězce lze skládat opakovaně s obměnou hodnot pomocí cyklů. Jedná se tedy spíše o technické řešení, a proto bude uveden pouze jeden z těchto scénářů, neboť ostatní procedury jsou velice podobné. Jedná se o scénář *Čísla od 0 do 9 vpřed* (viz Tabulka 6, s. 56), který realizuje procedura `scenario-numbers-order-forward`.

```
to scenario-numbers-order-forward
  let x 0
  while [ x < 10 ] [
    create-chunk (word x " next " (x + 1))
    repeat 3000 [ layout-spring chunks relations 0.06 4 1 ]
    set x x + 1
  ]
end
```

**Ukázka kódu 31: Procedura `scenario-numbers-order-forward` tvorby scénáře *Čísla od 0 do 9 vpřed*.**

### 3.2.2.5 Kalibrace a kalibrační data

V implementaci modelu byla navržena sada procedur pro snadnou kalibraci parametrů modelu (viz Validace a kalibrace modelu, s. 57). Kalibrační data jsou uložena strukturovaně v seznamech. Na první úrovni seznamu je uveden název sady. Druhá úroveň seznamu obsahuje fan číslo, pro které je sada platná, a s touto sadou související kalibrační data. Tato data mají podobu seznamu uspořádaných dvojic čísla dne a aktivační úrovně pro daný den. Sady jsou vytvářeny pomocí procedury `create-calibration-data-sets` při prvním požadavku na získání kalibračních dat příslušnou funkcí. Vzhledem k množství kalibračních dat je v ukázce zobrazena pouze hlavní struktura procedury.

Celkem byly v implementaci připraveny čtyři kalibrační sady na základě dat z reálných experimentů. Úrovně aktivací v sadě je potom nutné převádět pomocí parametrů časového $F$ a aktivačního $f$ měřítka (viz Rovnice 4, s. 16), což provádí model automaticky dle parametrů `time-scaling` a `activation-scaling`.

```
to create-calibration-data-sets
  set calibration-data-sets [
    ...
    ;; podle Anderson J., 2004, An Integrated theory of the Mind
    ;; úrovně aktivací z naměřených časů A=ln(F/t)*(1/f) pro F=1.2975 a f=0.3165
    [ "Activation by time data (1974)" [
     [ "1-1"  [[ 1 0.0000] [ 2 0.9954] [ 3 1.3348] [ 4 1.5333] [ 5 1.6741]
                [ 6 1.7834] [ 7 1.8726] [ 8 1.9481] [ 9 2.0135] [10 2.0712]]]
     ...
     [ "3-3"  [[ 1 0.0000] [ 2 0.1828] [ 3 0.5812] [ 4 0.8142] [ 5 0.9795]
                [ 6 1.1077] [ 7 1.2125] [ 8 1.3011] [ 9 1.3779] [10 1.4455]]]
    ]]
    ...
  ]
end
```

**Ukázka kódu 32: Část procedury `create-calibration-data-sets` pro vytvoření kalibračních dat.**

Pro vlastní získání kalibračních dat jako seznamu je použita funkce `calibration-data-set`. Tato funkce má dva parametry, a to název kalibrační sady `set-name` a fan číslo `fan`, pro které chceme data jako seznam dvojic čísla dne a aktivační úrovně získat. Pokud vyžádaná kalibrační data neexistují, je funkcí vrácen prázdný seznam. Seznam v globální proměnné obsahující kalibrační data je filtrován

na dvou úrovních pomocí funkce `filter`. Nejdříve je vybrána daná sada a potom data související se zadaným fan číslem.

```
to-report calibration-data-set [ set-name fan]
  if (calibration-data-sets = 0) [ create-calibration-data-sets ]
  let select-data-set filter [ first ? = set-name ] calibration-data-sets
  if(length select-data-set > 0) [
    let fan-sets item 1 first select-data-set
    if(length fan-sets > 0) [
       let select-fan-set filter [ first ? = fan ] fan-sets
       if (length select-fan-set > 0) [
         report item 1 first select-fan-set
       ]
    ]
  ]
  report []       ; kalibrační sada nenalzena tedy vrátit prázdný seznam
end
```

**Ukázka kódu 33: Funkce `calibration-data-set` vracející kalibrační data zadané sady a fan čísla.**

Pro vykreslení kalibračních dat do grafu v režimu kalibrace se používá procedura `plot-pen-calibration-data`, kterou používají jednotlivá pera grafu pojmenována fan čísly a v režimu kalibrace vykreslují kalibrační hodnoty jako body.

```
to plot-pen-calibration-data [ pen ]
  if (not calibration-mode?) [ stop ]     ; vykreslit data jen v režimu kalibrace
  let data calibration-data-set calibration-set pen     ; získat kalibrační data
  if (empty? data) [ stop ]               ; zastavit je-li datová sada prázdná
  set-current-plot "Activation level"           ; nastavit aktuální graf
  set-current-plot-pen (word "fan " pen)        ; nastavit pero pro vykreslování
  foreach data [ plotxy (item 0 ?) (item 1 ?) ] ; vykreslit kalibrační data
end
```

**Ukázka kódu 34: Procedura `plot-pen-calibration-data` pro vykreslení kalibračních dat do grafu.**

Základní nastavení modelu tak, aby byl ve shodě s kalibračními daty, se provádí procedurou `calibrate-model`. Tato procedura nastaví všech třináct parametrů modelu podle hodnot, které byly předem navrženy jako výchozí. Tyto údaje jsou získány pomocí parametru `cal-data` jako seznam.

```
to calibrate-model [ cal-data ]
  set decay-coef item 0 cal-data            ; korekce degradace paměti
  set chunk-gain item 1 cal-data            ; zisk aktivace bloku jeho užitím
  set chunk-dec-rate item 2 cal-data        ; míra degradace aktivace bloku
  set relation-gain item 3 cal-data         ; zisk aktivace užitím relace
  set relation-dec-rate item 4 cal-data     ; míra degradace aktivace relace
  set fan-gain-coef item 5 cal-data         ; vliv fan efektu na zisk aktivace
  set fan-decay-coef item 6 cal-data        ; vliv fan efektu na degradaci
  set diff-gain-coef item 7 cal-data        ; vliv rozdílu fan čísla na zisk
  set diff-decay-coef item 8 cal-data       ; vliv rozdílu fan čísla na degradaci
  set time-scaling item 9 cal-data            ; časové měřítko
  set activation-scaling item 10 cal-data     ; aktivační měřítko
  set chunk-retrieve-treshold item 11 cal-data ; limit pro vybavení
  set s-activation-noise item 12 cal-data     ; šum v aktivačních hodnotách
end
```

**Ukázka kódu 35: Procedura `calibrate-model` pro nastavení hodnot parametrů modelu při kalibraci.**

Nastavení výchozí kalibrace modelu podle předem připravených hodnot se provádí pomocí procedury `default-calibration`. Procedura nastaví parametry

modelu do výchozího nastavení podle vybrané kalibrační sady tak, aby mohla být jednoduše ověřena shoda výsledků modelu s kalibračními daty.

```
to default-calibration
  set calibration-mode? true          ; přepnout do kalibračního módu
  set decay-adj? true                 ; zapnout nelineární korekci degradace
  set context-mode? true              ; použít kontextový režim
  set activation-noise? false         ; vypnout šum
  set chunks-in-one-step 1            ; v jeden den jen jeden blok
  set learning-intensity 1            ; intenzita učení (relativní zisk aktivace)
  let model-setup [                   ; hodnoty pro nastavení dle kalibračních sad
    ["Model of activation by time data (1974)"
     [ 3.1416 0.985 0.385 1.380 0.452 0.690 0.952 0.972 0.962
       1.2975 0.3165 0.93 0.40 ]]
    ...
    ["Validation"
     [ 0.0000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
       1.2940 0.1862 0.93 0.40 ]]
  ]
  calibrate-model item 1 first        ; kalibrovat model podle zvolené konfigurace
                  filter [ first ? = calibration-set ] model-setup
end
```

**Ukázka kódu 36: Procedura `default-calibration` pro nastavení modelu dle vybrané kalibrační sady.**

#### 3.2.2.6 Řízení simulace

Procedura `update-max-act-value` slouží pro případnou aktualizaci maximální hodnoty aktivace. Tato aktivace se potom používá pro výpočet případné velikosti simulačního kroku (viz Výpočet délky simulačního kroku, s. 51). Procedura je používána procedurami na nižších úrovních modelu z důvodu celkové jednodušší implementace, i když je tím porušena hierarchie volání funkcí mezi úrovněmi modelu.

```
to update-max-act-value [ val ]
  ; pokud je zadaná hodnota větší než aktuální maximum je tak novým maximem
  if(val > activation-max-value) [ set activation-max-value val ]
end
```

**Ukázka kódu 37: Procedura `update-max-act-value` pro aktualizaci maximální hodnoty aktivace.**

Nastavení celého modelu do výchozího stavu provádí procedura `reset-all-parameters`, která je spuštěna kliknutím na příslušné tlačítko v uživatelském rozhraní. Na závěr je provedeno také výchozí nastavení parametrů modelu podle zvolené kalibrační sady pomocí procedury `default-calibration` (viz Ukázka kódu 36).

```
to reset-all-parameters
  set enable-global-context? false  set tick-delta 1
  set update-view? true             set show-values? true
  set view-print? true              set update-plots? true
  set use-context-cue? true         set label-dir 330
  set label-dist 1                  set plot-scale 30
  set view-val-precision 5          set learner-strategy "Calibration"
  set learner-mode "Learning"
  set relation-value-display "S | Activation level"
  set chunk-value-display "An | Activation level (noise)"
  default-calibration          ; výchozí kalibrace podle vybrané kalibrační sady
end
```

**Ukázka kódu 38: Procedura `reset-all-parameters` pro nastavení celého modelu do výchozího stavu.**

Pro experimentování s kontextovou aktivací bloků byla navržena procedura `default-global-context`, která nastaví všem blokům výchozí kontextovou váhu podle jejich aktuální celkové úrovně aktivace včetně případného šumu v této úrovni. Součet všech vah je potom $\sum_j W_j = 1$, aby byla dodržena podmínka pro omezení součtu celkové kontextové váhy (viz Kontextová úroveň aktivace bloku, s. 23).

```
to default-global-context
  let a-total sum [exp A-level-noise] of chunks
  if (a-total > 0) [
    ask learned-chunks [ set C-weight (exp A-level-noise) / a-total ]
  ]
end
```

**Ukázka kódu 39: Procedura `default-global-context` pro nastavení výchozí váhy všem blokům.**

Aktualizace pohledu celého modelu během simulace se provádí pomocí procedury `update-view`. Ta umožní nastavit tmavé nebo světlé zobrazení použitelné pro tisk. Pokud není model v režimu kalibrace, je zobrazovaný graf aktivace v čase průběžný. Následně je pomocí procedur `chunk-update-view` a `relation-update-view` aktualizován vzhled bloků a relací dle zvoleného nastavení.

```
to update-view
  if (view-current-version? != view-print?) [      ; tmavé nebo světlé zobrazení
    set view-color ifelse-value (view-print?) [ 1 ] [ -1 ]
    ask patches [ set pcolor (gray + 4.9 * view-color) ]
    set view-current-version? view-print?
  ]
  if (not calibration-mode?) [                  ; mimo kalibrační mód
    set-current-plot "Activation level"         ; nastavit graf aktivace
    set-plot-x-range (ticks - plot-scale) ticks  ; průběžné posouvání grafu
  ]
  ask chunks [ chunk-update-view ]              ; aktualizace zobrazení bloků
  ask relations [ relation-update-view ]        ; aktualizace zobrazení relací
end
```

**Ukázka kódu 40: Procedura `update-view` pro aktualizaci pohledu na relace a bloku v modelu.**

Před zobrazením stavu modelu pomocí procedury `update-view`, je celý stav modelu aktualizován procedurou `update-model`. Nejdříve dochází k aktualizaci

aktivací relací a následně bloků, protože hodnota aktivace bloků je ovlivněna hodnotou aktivace souvisejících relací (viz Obrázek 5, s. 41).

```
to update-model
  ask relations [ relation-update-activation ]   ; první aktualizovat relace
  ask chunks [ chunk-update-activation ]         ; následně aktualizovat bloky
  set chunks-a-sum sum [a] of chunks             ; celková základní aktivace
  if (chunks-a-sum > 0) [
    set chunks-prob-choice-avg 1 / count learned-chunks    ; rovnoměrné rozdělení
    ask chunks [ chunk-update-retrieve ]   ; aktualizace pravděpodobností bloků
  ]
  if(update-view?) [ update-view ]      ; aktualizovat pohled je-li vyžadováno
  if(update-plots?) [ update-plots ]    ; aktualizovat grafy je-li vyžadováno
end
```

**Ukázka kódu 41: Procedura `update-model` pro aktualizaci celkového stavu modelu.**

Pro výpočet délky následujícího simulačního kroku (viz Výpočet délky simulačního kroku, s. 51) slouží funkce `tick-delta-advance`, která implementuje výpočet uvedený v návrhu modelu. Parametr `lock-tick-delta?` slouží k validaci a umožňuje uzamknout velikost kroku na zadanou hodnotu.

```
to-report tick-delta-advance
  if(lock-tick-delta?) [ report tick-delta ]         ; uzamčená velikosti kroku
  let t ifelse-value (activation-max-value != 0)
          [ precision (((1 / activation-max-value) * 0.05) + 0.05) 1 ] [ 1 ]
  if-else (t > tick-delta-max )     ; když je krok větší než maximální krok
    [ set tick-delta tick-delta-max ]    ; nastavit maximální krok
    [ set tick-delta t ]                 ; nastavit krok o spočítané velikosti
  report tick-delta
end
```

**Ukázka kódu 42: Funkce `tick-delta-advance` vrací velikost následujícího simulačního kroku.**

Posun času simulace je realizován pomocí procedury `advance-time`, jejíž parametr `advance-ticks` umožní zadat časový interval, o kolik má být čas simulace posunut. V kalibračním režimu, kdy nás nezajímá vizualizace poklesu aktivačních hodnot, je čas simulace okamžitě posunut o zadaný interval.

Mimo kalibrační režim je vizualizován i pokles aktivačních hodnot mezi jednotlivými posuny simulace v čase. Simulace postupuje v tomto režimu o vypočítaný krok podle funkce `tick-delta-advance`. Pokud je velikost kroku řízena maximální aktivační hodnotou v modelu, potom je pokles aktivace zobrazen dostatečně plynule bez velkých skoků mezi jednotlivými kroky. Nicméně na absolutní konečnou přesnost výpočtu aktivačních hodnot nemá velikost simulačního kroku téměř vliv a uplatňují se jen velmi malé zaokrouhlovací chyby související s výpočtem.

```
to advance-time [ advance-ticks ]
  if-else (calibration-mode?) [       ; pro kalibrační režim
   tick-advance advance-ticks              ;posunout se přímo o zadaný čas
  ][                                  ; v běžném režimu zobrazit pokles aktivace
    let start-time ticks                    ; čas počátku časového posunu
    let end-time start-time + advance-ticks ; čas na konci časového posunu
    let time-step tick-delta-advance        ; spočítat velikost simulačního kroku
    while [ ticks < end-time - time-step ][ ; dokud lze posouvat simulaci o krok
      tick-advance time-step                ; provést posun simulačního kroku
      if (enable-global-context?) [ default-global-context ]
      update-model                          ; aktualizovat stav simulace
      set time-step tick-delta-advance      ; spočítat novou velikost kroku
      set activation-max-value 0            ; maximální hodnotu aktivace na 0
    ]
    tick-advance end-time - ticks           ; posunout na konec intervalu
  ]
end
```

**Ukázka kódu 43: Procedura `advance-time` provede posun času simulace o zadaný interval.**

Hlavní procedurou simulačního cyklu (viz Simulační cyklus, s. 52) je procedura `go` běžná v jiných NetLogo modelech. Tato procedura je navíc doplněna netypicky parametrem `time-step`, který udává, o kolik má být posunut čas simulace před provedením souvisejících akcí uvnitř procedury.

```
to go [ time-step ]
  advance-time time-step           ; posunout čas simulace o zadaný krok
  if (learner-mode = "Learning") [ ; když je modelovaný agent v režimu učení
    let select-chunk nobody          ; nastavit vybíraný blok na žádný
    repeat chunks-in-one-step [      ; počet bloků v jednom sezení (délka učení)
      let pick-task item 1 first
                       filter [ first ? = learner-strategy ] learner-strategies

      set select-chunk one-of ifelse-value (context-mode?)
            [ (runresult pick-task (chunks with [ count my-in-relations > 0])) ]
            [ (runresult pick-task (chunks))]

      if (select-chunk = nobody) [
        set select-chunk one-of ifelse-value  (context-mode?)
            [ chunks with [ count my-in-relations > 0 and not learned?] ]
            [ chunks with [ not learned? ] ]
      ]
      if(select-chunk != nobody) [        ; když byl vybrán nějaký blok
        ask chunks [ set C-weight 0 ]       ; kontextová váha všech bloků na 0
        ask select-chunk [ chunk-learned ]  ; naučit vybraný blok
      ]
      ; mimo strategii kalibrace provést časový posun o 30 sekund mezi bloky
      if(learner-strategy != "Calibration") [ advance-time 34.722E-5 ]
    ]
  ]
  update-model                            ; aktualizovat stav modelu
  if(update-view?) [ update-view ]        ; aktualizovat pohled je-li vyžadováno
  if(update-plots?) [ update-plots ]      ; aktualizovat grafy je-li vyžadováno
end
```

**Ukázka kódu 44: Procedura `go` pro realizaci hlavního simulačního cyklu modelu.**

Nejdříve je tedy netypicky posunut čas simulace pomocí procedury `advance-time`. K tomuto posunutému času jsou provedeny související akce aktualizace a vizualizace stavu modelu. Díky tomu lze vidět model takový, jaký je v okamžiku před výpočtem poklesu aktivačních hodnot. Toto je velmi užitečné pro

potřeby validace a kalibrace, neboť díky tomu vidíme, v jakém stavu je blok po získání aktivační hodnoty jeho užitím. V rámci procedury je potom, pokud je agent v režimu učení, implementován výběr zvolené strategie učení (viz Strategie učení, s. 56).

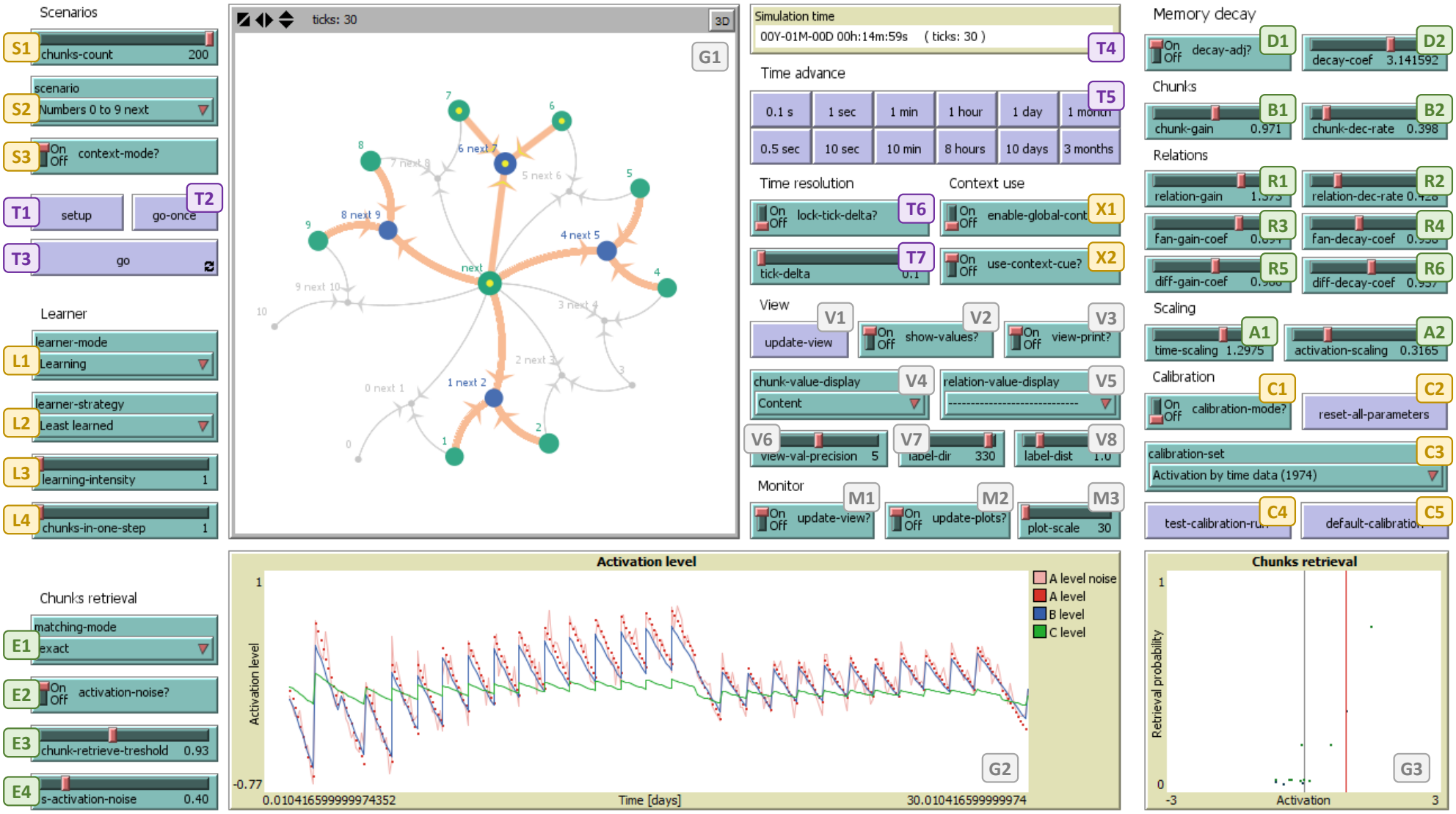
Poslední představenou procedurou je procedura `setup`, která inicializuje model a nastaví počáteční stav simulace. Zajímavou částí této procedury je vytvoření seznamů strategií učení a scénářů. Tyto seznamy obsahují úlohy (*tasks*), které budou v případě výběru spuštěny. Tímto je možné vyhnout se složitému kódu modelu založenému na větvení podle různých podmínek.

```
to setup
  clear-all                              ; vše smazat
  setup-plots                            ; inicializovat grafy
  set chunk-size-min 0.5                 ; minimální velikost bloku
  set chunk-size-range  2                ; rozsah velikosti bloku
  set relation-thickness-min 0.01        ; minimální tloušťka relace
  set relation-thickness-range 1         ; rozsah tloušťky relace
  reset-ticks                            ; resetovat časovač simulace
  set tick-delta-max 30                  ; maximální krok simulace je 30 dnů
  view-chunk-init                        ; inicializace zobrazení bloků
  view-relation-init                     ; inicializace zobrazení relací
  set learner-strategies  (list          ; seznam s úlohami pro strategie
    (list "Calibration" (task [ ? with-min [who]]))
    (list "Random" (task [ ? ]))
    (list "Least learned" (task [ ? with [A-level-noise < 0 and learned?]
                                  with-min [A-level-noise]]))
    (list "Bellow Threshold"
      (task [ ? with [ A-level-noise < chunk-retrieve-treshold and learned?]
                with-min [ A-level-noise ]])))
  let scenarios (list                    ; seznam úloh simulačních scénářů
    (list "Test base activation" (task [ scenario-test-base-activation ]))
    (list "Test context activation" (task [ scenario-test-context-activation ]))
    (list "Set of unrelated chunks" (task [ scenario-random-unrelated-chunks ]))
    (list "ACT-R fan 1-1 calibration" (task [ scenario-fan-1-1 ]))
    ...
    (list "Multiplication 0 to 10" (task [  scenario-multiplication-0-10 ])))
  ; spustit vybraný scénář
  run item 1 first filter [ first ? = scenario ] scenarios
  set learned-chunks no-turtles              ; žádné bloky nejsou naučeny
  set view-current-version? not view-print?  ; aktualizace typu zobrazení
  update-view                                ; aktualizovat zobrazení modelu
  update-plots                               ; aktualizovat grafy
end
```

**Ukázka kódu 45: Procedura `setup` pro nastavení počátečního stavu modelu.**

### 3.2.3 Grafické uživatelské rozhraní

Pro běžnou práci s modelem slouží v NetLogu uživatelské rozhraní modelu (*Interface*). Toto rozhraní umožní pohodlně obsluhovat nastavení modelu a zobrazit graficky výstupy simulace. V této kapitole je popsáno rozhraní implementovaného modelu deklarativní paměti podle teorie ACT-R a význam jednotlivých ovládacích prvků. Na tyto prvky je následně odkazováno v kapitolách souvisejících s validací, kalibrací a experimentováním s modelem.




**Obrázek 8: Grafické uživatelské rozhraní implementovaného modelu deklarativní paměti podle teorie ACT-R s označením komponent a parametrů modelu.**

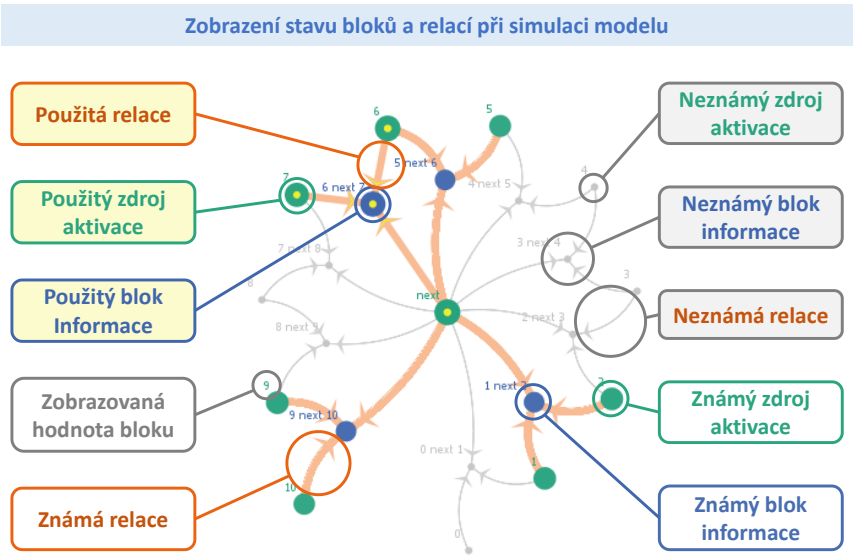
Komponenty uživatelského rozhraní (viz Obrázek 8) jsou logicky seskupeny podle významu s přihlédnutím k využití dostupného prostoru. V rámci těchto skupin jsou pro účely odkazování se z textu označeny jednotlivé komponenty uživatelského rozhraní písmenem skupiny a číslem. Toto označení tvoří jednoduchý unikátní identifikátor daného prvku, na který je v následujících kapitolách odkazováno.

**Tabulka 9: Přehled všech komponent uživatelského rozhraní implementovaného modelu a jejich význam.**

| | Skupina | Komponenty a parametry | | |
|---|---|---|---|---|
| | **Nastavení modelu** | | | |
| S | **Scénáře** (*Scenarios*) (viz s. 56) | {S1} | `chunks-count` | počet vytvářených pro scénář *Sada samostatných bloků* |
| | | {S2} | `scenario` | výběr simulačního scénáře (viz s. 74) |
| | | {S3} | `context-mode?` | zapíná nebo vypíná kontextový mód (viz s. 74) |
| L | **Modelovaný agent** (*Learner*) (viz s. 52, 56) | {L1} | `learner-mode` | režim, ve kterém se agent nachází |
| | | {L2} | `learner-strategy` | výběr strategie učení pro agenta (viz s. 81) |
| | | {L3} | `learning-intensity` | relativní velikost zisku aktivace při učení dle kalibrace |
| | | {L4} | `chunks-in-one-step` | počet bloků naučených během iterace simulačního cyklu |
| C | **Kalibrace modelu** (*Calibration*) (viz s. 57) | {C1} | `calibration-mode?` | zda je/není model v režimu kalibrace (viz s. 81) |
| | | {C2} | `reset-all-parameters` | nastaví celýmodel do výchozího stavu (viz s. 79) |
| | | {C3} | `calibration-set` | výběr kalibrační sady (viz s. 78) |
| | | {C4} | `test-calibration-run` | spustí proceduru kalibrace |
| | | {C5} | `default-calibration` | nastavení modelu dle vybrané sady (viz s. 78) |
| X | **Užití kontextu** (*Context use*) (viz s. 41) | {X1} | `enable-global-context?` | povolení globálního kontextu (viz s. 79) |
| | | {X2} | `use-context-cue?` | použití relací a kontextu (viz s. 71) |
| | **Parametry modelu** | | | |
| D | **Nastavení paměti** (*Memory decay*) (viz s. 47) | {D1} | `decay-adj` | povolení nelineární korekce degradace paměti (viz s. 64) |
| | | {D2} | `decay-coef` | koeficient nelineární degradace paměti (viz s. 64) |
| B | **Nastavení bloků** (*Chunks*) (viz s. 49) | {B1} | `chunk-gain` | zisk hodnoty aktivace užitím bloku (viz s. 71) |
| | | {B2} | `chunk-dec-rate` | míra degradace hodnoty aktivace bloku (viz s. 66) |
| R | **Nastavení relací** (*Relation*) (viz s. 13, 49) | {R1} | `relation-gain` | zisk hodnoty aktivace užitím relace (viz s. 71) |
| | | {R2} | `relation-deg-rate` | míra degradace hodnoty aktivace relace (viz s. 72) |
| | | {R3} | `fan-gain-coef` | korekce zisku relace dle fan čísla zdroje (viz s. 73) |
| | | {R4} | `fan-decay-coef` | korekce degradace relace dle fan čísla zdroje (viz s. 72) |
| | | {R5} | `diff-gain-coef` | korekce zisku relace díky rozdílu fan čísel (viz s. 72) |
| | | {R6} | `diff-decay-coef` | korekce degradace relace díky rozdílu fan čísel (viz s. 72) |

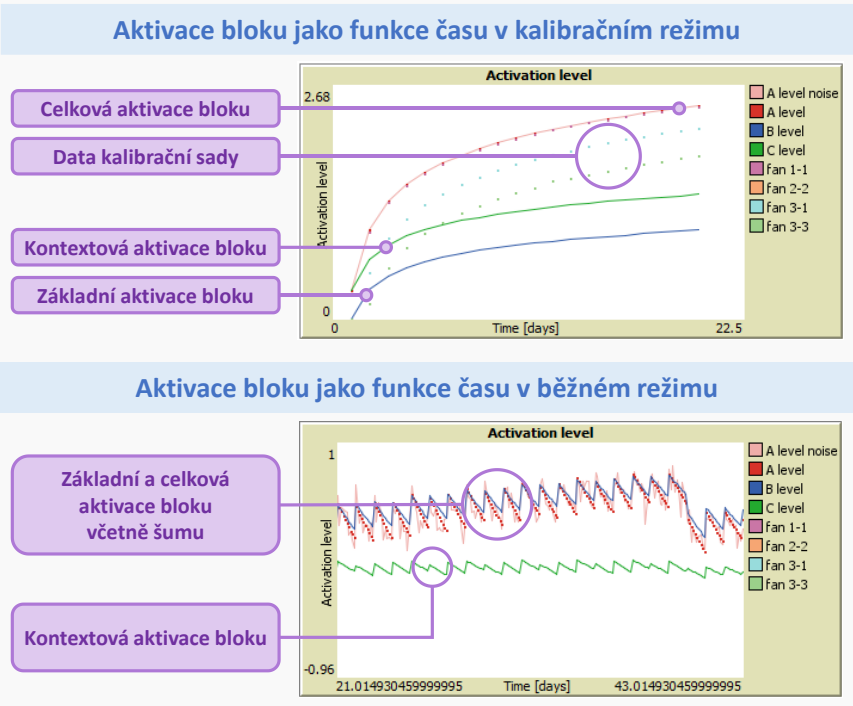
| | | | | |
|---|---|---|---|---|
| A | **Měřítka převodu** (*Scaling*) (viz s. 14) | {A1} | `time-scaling` | časové měřítko převodu aktivační hodnoty (viz s. 76) |
| | | {A2} | `activation-scaling` | aktivační měřítko převodu aktivační hodnoty (viz s. 76) |
| E | **Vybavení bloků** (*Chunks retrieval*) (viz s. 16, 70) | {E1} | `matching-mode` | mód používaný pro výběr bloku |
| | | {E2} | `activation-noise?` | povolení šumu v hodnotách aktivace |
| | | {E3} | `chunk-retrieve-treshold` | limit pro vybavení bloku (viz s. 82) |
| | | {E4} | `s-activation-noise` | úroveň šumu v aktivačních hodnotách |
| | **Řízení a časování simulace** | | | |
| T | **Počáteční stav** (viz s. 31) | {T1} | `setup` | nastavení simulace modelu do výchozího stavu (viz s. 82) |
| | **Krokování** (viz s. 50, 52) | {T2} | `go-once` | jeden průchod simulačního cyklu s krokem jednoho dne (viz s. 81) |
| | | {T3} | `go` | opakovaný průchod simulačního cyklu s krokem jednoho dne (viz s. 81) |
| | **Simulační čas** (*Simulation time*) | {T4} | | zobrazuje celkový čas simulace od jejího začátku ve snadno čitelném formátu s využitím běžných časových jednotek (viz s. 63) |
| | **Posun času** (*Time advance*) (viz s. 51) | {T5} | | provede jeden průchod simulačního cyklu s vybranou velikostí simulačního kroku (viz s. 81) |
| | **Časové rozlišení** (*Time resolution*) (viz s. 51, 70) | {T6} | `lock-tick-delta?` | povolit uzamčení velikosti simulačního kroku |
| | | {T7} | `tick-delta` | aktuální velikost simulačního kroku |
| | **Zobrazení výstupů** | | | |
| V | **Nastavení pohledu** (*View*) (viz s. 42, 86) | {V1} | `update-view` | provede aktualizaci stavu pohledu (viz s. 79) |
| | | {V2} | `show-values?` | zobrazování hodnot u bloků a relací (viz s. 70) |
| | | {V3} | `view-print?` | verze pro tisk nebo obrazovku (viz s. 79) |
| | | {V4} | `chunk-value-display` | hodnoty zobrazované u bloků (viz s. 70) |
| | | {V5} | `relation-value-display` | hodnoty zobrazované u relací (viz s. 74) |
| | | {V6} | `view-val-precision` | počet řádů u zobrazovaných hodnot (viz s. 70, 74) |
| | | {V7} | `label-dir` | směr umístění popisku bloků (viz s. 65) |
| | | {V8} | `label-dist` | vzdálenost umístění popisku bloků (viz s. 65) |
| M | **Sledování stavu** (*Monitors*) (viz s. 45, 80) | {M1} | `update-view?` | povolení provádění aktualizace pohledu modelu |
| | | {M2} | `update-plots?` | povolení vykreslování grafů |
| | | {M3} | `plot-scale` | velikost časového okna zobrazeného v grafu (viz s. 79) |
| G1 | **Pohled** (*View*) (viz s. 52) | {G1} | | informativně zobrazuje aktuální stav bloků a relací při simulaci (viz Obrázek 9) |

Aktualizace pohledu provádí NetLogo automaticky. V případě vypnutí volby `update-view?` nejsou vykonány vizualizační procedury, což se může projevit celkovým zrychlením běhu simulace. Kdykoliv a při jakékoliv změně zobrazovaných údajů pomocí ovládacích prvků {V2} až {V8} lze provést aktualizaci pohledu pomocí prvku {V1}, tedy tlačítka `update-view`.



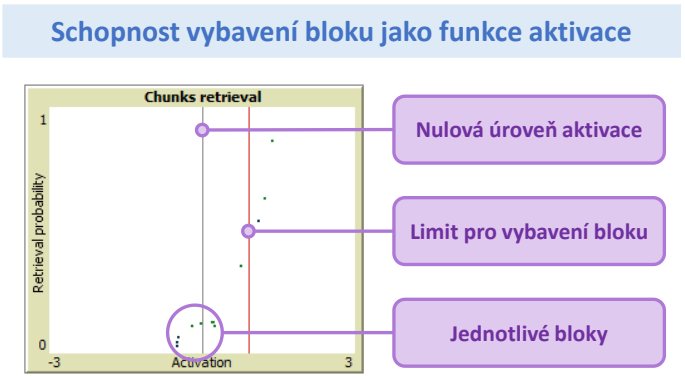

**Obrázek 9: Vizualizace bloků a relací v pohledu modelu.**

| | | |
|---|---|---|
| G2 | **Aktivace bloků** (*Activation level*) (viz s. 49) | {G2} zobrazuje aktivaci bloků jako funkci času<br>V režimu kalibrace nejsou zobrazovány poklesy aktivačních hodnot mezi jednotlivými iteracemi simulačního cyklu. Funkce má proto stejný průběh jako funkce aktivací z reálných experimentů, kdy je zjišťována hodnota aktivace v daný den při jejím nepřímém měření v experimentech (viz Obrázek 10). V reálné situaci však dochází k poklesu aktivačních hodnot, což model umožní zobrazit při vypnutí kalibračního režimu pomocí vypnutí volby `calibration-mode?` {C1}. |
| G3 | **Vybavení bloků** (*Chunks retrieval*) (viz s. 21) | {G3} zobrazuje pravděpodobnost vybavení bloku jako funkci aktivační hodnoty<br>Graf funkce slouží pro přehled, jaká je pravděpodobnost vybavení bloků. Tato |




**Obrázek 10: Funkce aktivace v závislosti na času simulace.**

pravděpodobnost se odvíjí od aktuální celkové aktivace bloku a dále od nastavení limitu pro vybavení `chunk-retrieve-treshold` a střední úrovně šumu v aktivačních hodnotách `s-activation-noise`. Tyto parametry potom ovlivňují také velikost šumu v aktivačních hodnotách bloků.




**Obrázek 11: Funkce vybavení bloku v závislosti na jeho aktivaci a nastavených parametrech modelu.**

## 3.3 Validace modelu

Úvodní informace pro validaci modelu byly popsány v teoretické části (Validace a kalibrace, s. 34), následně zapracovány v praktické části do návrhu kalibračních a validačních procedur (Validace a kalibrace modelu, s. 57). V této podkapitole jsou popsány kroky, pomocí kterých lze otestovat formální správnost implementovaného modelu podle navržených vztahů a pravidel. Pro zkrácení a zpřehlednění se popis kroků odkazuje na použité komponenty modelu (viz Obrázek 8) v jednotlivých krocích testovacích případů.

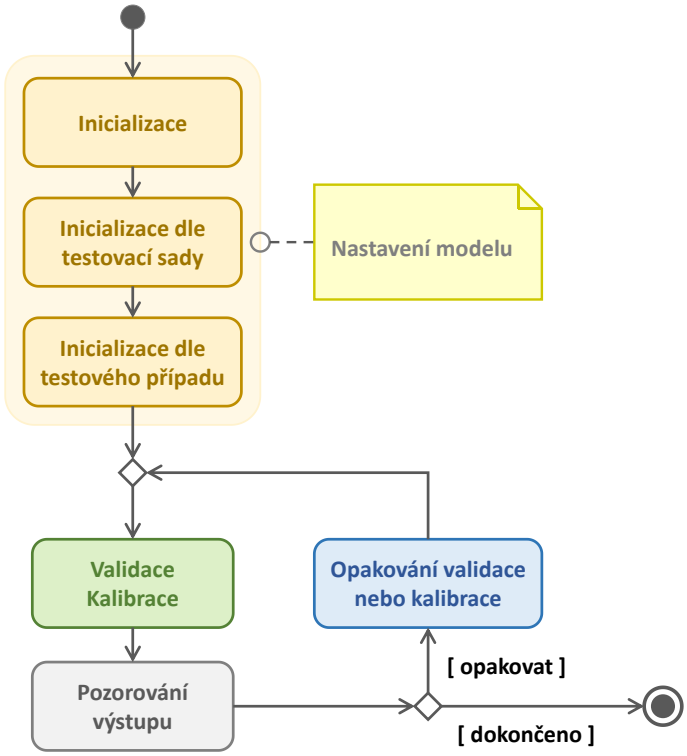
Je uveden postup validace nejdůležitějších částí modelu a použitých vztahů. Kompletní a důkladný popis validace celého modelu by byl pro potřeby diplomové práce příliš rozsáhlý, avšak byl při testování modelu fyzicky proveden. Uživatel s dostatečnými znalostmi o fungování modelu, které získá mimo jiné z této podkapitoly, může sám provést případné zbývající validační kroky.

### 3.3.1 Degradace aktivačních hodnot

Základním vztahem, jehož validitu je potřebné v implementovaném modelu zajistit, a ze kterého celý model vychází, je rovnice degradace aktivačních hodnot (viz podkapitola Degradace hodnot aktivací, s. 43 a Rovnice 24, s. 46). Tento vztah se používá pro degradaci hodnot aktivací užití bloků i relací, které jsou následně základem pro výpočet všech ostatních hodnot v modelu. Je tedy nutné důkladně otestovat funkci tohoto vztahu v implementovaném modelu při různém nastavení

a zjistit případnou velikost odchylek od hodnoty, kterou pro tuto hodnotu předpovídá analytická funkce podle teorie ACT-R (viz Rovnice 14, s. 20).

Jako výchozí zdroj pro testovací data může posloužit Tabulka 4 na straně 44, obsahující vypočítané hodnoty aktivací užití $x_u$ pro různé časy $t = t_{sim} - t_u$, které uběhly od vzniku této hodnoty aktivace užití. Počáteční velikost hodnoty pak vyjadřuje parametr $x_u(t_u)$, a tedy hodnotu aktivace užití $x_u$ v čase jejího vzniku $t_u$. Rychlost poklesu aktivační hodnoty potom řídí nastavení parametru $d$ vyjadřující míru degradace paměti. Testovací případy jsou uvedeny z důvodu přehlednosti v tabulce, a pro zkrácení zápisu se odkazují na komponenty uživatelského rozhraní modelu (viz Tabulka 9, s. 84). Vlastní testy jsou prováděny podle uvedených kroků v následujícím schématu:




**Obrázek 12: UML diagram aktivity validace a kalibrace.**

Pro zkrácení zápisu testovacích a kalibračních testů jsou nejdříve uvedeny kroky pro globální nastavení validace nebo kalibrace. Následují inicializační kroky v rámci sady testů a na závěr kroky pro nastavení konkrétního testového případu. Po nastavení modelu lze přejít k vlastním krokům validace nebo kalibrace a pozorování výstupu. Pokud je nutné validaci nebo kalibraci opakovat, jsou uvedeny pouze nutné kroky pro opakování testu.

**Tabulka 10: Inicializační procedura pro validaci modelu.**

1. {C3} Vybrat kalibrační sadu `"Validation"`.
2. {C2} Nastavit celý model do výchozího stavu dle zvolené kalibrační sady `"Validation"`.
3. {V4} Vybrat zobrazování hodnoty aktivace bloku `"b | Base activation value"`.
4. {V5} Vybrat zobrazování hodnoty aktivace relace `"s | Activation value"`.
5. {IN} Inicializační procedura pro validační sadu

**Tabulka 11: Sada testů pro validaci vztahu pro degradaci aktivace bloku.**

**Procedury pro sadu testů degradace aktivace bloku**

**Inicializace sady**

1. {S2} Vybrat simulační scénář `"Test base activation"`.
2. {T6} Nastavit uzamčení simulačního kroku na **ON**.
3. {T1} Nastavit simulaci modelu do výchozího stavu.
4. {IN} Provést inicializaci daného testu

**Kroky validace**

1. {T2} Posunout simulaci o jeden průchod simulačního cyklu.
2. {L1} Přepnout modelovaného agenta do režimu odpočinku `"Relaxing"`.
3. {T2} Opakovat průchody simulačního cyklu a pozorovat hodnoty.

**Opakování testu**

1. {L1} Přepnout modelovaného agenta do režimu učení `"Learning"`.
2. {T1} Nastavit simulaci modelu do výchozího stavu.
3. {VA} Provést kroky validace.

**Pozorování výstupu**

- {G1} Sledujeme, zda jsou jednotlivé hodnoty v náhledu modelu ve shodě s daty získanými analytickým vztahem pro jednotlivé časové kroky. Čas lze sledovat pomocí {T4}. Kroky jsou po jednotlivých dnech.
- {G2} Je zobrazována aktivační úroveň bloku, v tomto konkrétním případě podle vztahu $A(t) = \ln(x_u(t))$.

**Test 1 : Pokles základní hodnoty aktivace $b_i$ bloku při hodnotách $d = 1$ a $x_u(t_u) = 1$**

**Výsledky dle vztahu** $x(t) = \frac{x_u(t_u)}{t^d}$

| $t$ [dny] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_u(t)$ | 1 | 0,5 | 0,3333 | 0,25 | 0,2 | 0,1667 | 0,1429 | 0,125 | 0,1111 | 0,1 |

**Inicializace testu**

- pouze inicializace dle validační sady

**Test 2 : Pokles základní hodnoty aktivace $b_i$ bloku při hodnotách $d = 1$ a $x_u(t_u) = 2$**

**Výsledky dle vztahu** $x(t) = \frac{x_u(t_u)}{t^d}$

| $t$ [dny] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_u(t)$ | 2 | 1 | 0,6667 | 0,5 | 0,4 | 0,3333 | 0,2857 | 0,25 | 0,2222 | 0,2 |

**Inicializace testu**

1. {B1} Nastavit zisk hodnoty aktivace užitím bloku `chunk-gain` na `2`

**Test 3 : Pokles základní hodnoty aktivace $b_i$ bloku při hodnotách $d = 0,5$ a $x_u(t_u) = 1$**

**Výsledky dle vztahu** $x(t) = \frac{x_u(t_u)}{t^d}$

| $t$ [dny] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_u(t)$ | 1 | 0,7071 | 0,5774 | 0,5 | 0,4472 | 0,4082 | 0,3780 | 0,3536 | 0,3333 | 0,3162 |

| | |
|---|---|
| **Inicializace testu** | 2. {B2} Nastavit míru degradace paměti `mem-deg-rate` na `0,5` |

Rychlost degradace $d$ je poloviční oproti Testu 1 ($T1$). Počet kroků (včeně kroku užití) nutných k tomu, aby aktivační hodnota klesla na stejnou úroveň, jako v $T1$ je druhá mocnina počtu kroků oproti $T1$. Například pro $x_{T1}(t) = 0{,}25$ jsou podle $T1$ nutné $4$ kroky od času vzniku užití bloku (první krok), kdy pro $x_{T2}(t) = 0{,}25$ bude nutné $16$ kroků. Pokud snížíme hodnotu degradace ještě o polovinu na hodnotu $d = 0{,}25$, bude potřebné $n^4$ kroků a tedy $256$ kroků na pokles na hodnotu $0{,}25$. Naopak při dvojnásobné rychlosti degradace paměti oproti $d = 1$, tedy pro $d = 2$ bude mít aktivace hodnotu $0{,}25$ již ve druhém kroku, ale stále platí mocninný vztah, tentokrát však s hodnotou $n^{1/2} = \sqrt[2]{n}$ pro počet kroků v T1. Při validaci degradace paměti lze tedy experimentovat s různým poklesem, nejdříve jej vypočítat analyticky a následně ověřit modelem simulačně.

**Test 4 : Pokles základní hodnoty aktivace $b_i$ bloku při hodnotách $d = 0,437$ a $x_u(t_u) = 1,285$**

**Výsledky dle vztahu** $x(t) = \frac{x_u(t_u)}{t^d}$

| $t$ [dny] | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_u(t)$ | 1,285 | 0,9492 | 0,7951 | 0,7011 | 0,6360 | 0,5873 | 0,5490 | 0,5179 | 0,4919 | 0,4698 |

| | |
|---|---|
| **Inicializace testu** | 1. {B1} Nastavit zisk hodnoty aktivace užití bloku `chunk-gain` na `1,285`<br>2. {B2} Nastavit míru degradace paměti `mem-deg-rate` na `0,437` |
| **Poznámka** | Byly zvoleny náhodně různé hodnoty d a $x_u(t_u)$, aby byla otestována kombinace různé aktivace a degradace paměti s hodnotami s pohyblivou řádovou čárkou. |

**Test 5 : Validace funkce posunu času** (*Time advance*) {T5}

Lze použít uvedená nastavení degradace paměti z **Testů 1 až 4** a pomocí opakovaného posunu simulačního času tlačítky {T5} se pokusíme složit výsledný časový interval tak, aby zobrazovaný čas simulace {T4} odpovídal některému z časů uváděných v tabulce hodnot získaných podle analytického vztahu pro konkrétní test. Například lze stisknout tlačítko 8 hodin (*8 hours*) 3krát pro posun o jeden den nebo stisknout tlačítko 1 hodina (*1 hour*) 24krát. Případně si lze provést vlastní analytický výpočet a potom otestovat validitu modelu posunem o zvolený čas. Nesmíme však zapomenout korektně nastavit degradace aktivace $d$ a počáteční hodnotu aktivace $x_u(t_u)$ podle parametrů zvoleného testu.

**Test 6 : Validace proměnné velikosti simulačního kroku**

| | |
|---|---|
| **Inicializace testu** | 1. {T6} Nastavit uzamčení simulačního kroku na **OFF**.<br>2. {C1} Nastavit kalibrační režim na **OFF**. |
| **Popis testu** | Test provádíme s identickým nastavením jako **Test 5**, inicializaci doplníme o uvedené dva kroky. Pozorujeme, že v grafu aktivační úrovně {G2} je přesněji zobrazen průběh jejího poklesu. Vypočtené hodnoty i přes větší počet iterací, a tedy i aktualizací stavu modelu provedených pro každý zobrazovaný bod, musí přesně odpovídat hodnotám získaným pomocí analytického vztahu. |

**Test 7 : Opakované použití bloku**

Test provádíme se zvoleným nastavením podle **Testu 1 až 6** tak, že během jednotlivých kroků validace

modelovaného agenta ponecháme v režimu učení nastavením {L1} na hodnotu `"Learning"`. Během testu lze případně vypnout režim kalibrace nastavením {C1} na **OFF**. Zobrazí se tak poklesy aktivace při jednotlivých iteracích simulačního cyklu. Pomocí příkazu `print [ b-u-list ] of chunk 0` zadaného v *Command center* po každé iteraci simulačního cyklu lze sledovat hodnoty jednotlivých užití bloku a jejich pokles a tím degradaci se zvyšujícím se časem. Hodnoty jsou seřazeny od nejstaršího užití po nejnovější. Například po pěti iteracích simulačního cyklu v nastavení podle **Testu 1** seznam užití bloku obsahuje tyto hodnoty: `[[0.2 0] [0.25 1] [0.3333333333333333 2] [0.5 3] [1 4]]`. První hodnota ze dvojice je vždy hodnota aktivace daného užití bloku, druhou hodnotou je čas vzniku dané hodnoty (čas, kdy došlo k užití bloku). V pohledu modelu {G1} je potom zobrazován součet těchto hodnot. Jako celková základní aktivace bloku, ta je potom převedena na celkovou aktivaci v {G2}. V grafu vybavení bloku {G3} lze sledovat růst pravděpodobnosti vybavení. Doporučuji také vyzkoušet kratší časové posuny v iteracích simulačního cyklu pomocí tlačítek {T5} pro posun času simulace. V každé iteraci simulačního cyklu si lze vybrat, jaký režim zvolíme pro agenta pomocí volby v komponentě {L1}.

**Test 8 : Validace funkce intenzity učení**

Při tomto testu lze měnit hodnotu parametru `learning-intensity` komponenty {L3} která ovlivňuje relativní zisk počáteční aktivace užití bloků a relací. Při nastavení na hodnotu 2, bude relacím a blokům připisován dvojnásobek počáteční aktivace nastavené pomocí `chunk-gain` v {B1} a `relation-gain` v {R1}. Toto lze otestovat v libovolném dříve uvedeném testu, případně pomocí **Testu 7**

**Test 9 : Validace funkce počtu učených bloků v jedné iteraci simulačního cyklu**

Parametr `chunks-in-one-step` komponenty {L4} nastavuje, kolik bloků bude naučeno a tedy užito během jedné iterace simulačního cyklu. Při nastavení hodnoty 2 by mělo dojít ke dvěma užitím bloku ve stejný čas. Toto lze zkontrolovat postupem popsaným v **Testu 7**.

Aktivaci relací lze testovat stejným způsobem jako aktivaci bloků. Pro tuto úlohu je však nutné vybrat jiný simulační scénář, který obsahuje bloky se zdroji aktivace a režim kontextového módu v {S3} musí být povolen (ON). Připraveným testovacím scénářem pro validaci relací je scénář `"Test context activation"` v {S2}. Základní hodnota aktivace relace `"s | Activation value "` v {V5} se v principu funkce nijak neliší od hodnoty základní aktivace bloku `"b | Base activation value"` ve {V4}. Lze s ní tedy provádět podobné testy, jako u bloku viz **Test 1** až **Test 8**. Pro zobrazení seznamu užití u relací potom použijeme příkaz `print [ s-u-list ] of relation 0` až `2`.

Při validaci scénáře s relacemi nás ještě bude u bloků zajímat zobrazení hodnoty `"C-W | Context weight"`, která vyjadřuje kontextovou váhu bloku a hodnota `"a | Total activation value"`, vyjadřující celkovou hodnotu aktivace

bloku v {V4}. Pro aktualizaci zobrazení při změně zobrazované hodnoty nebo jiného údaje parametru ve skupině {V} používáme tlačítko {V1} `update-view`. Celková hodnota aktivace bloku `a` je potom součet násobku hodnoty aktivace relace a kontextové váhy bloku, ze kterého relace vychází všech relací, které směřují do bloku. Převod mezi hodnotou a úrovní aktivace je popsán vztahem viz Tabulka 3, s. 43.

### 3.3.2 Numerické odchylky modelu

Z povahy simulačního modelu, kdy stav dalšího kroku je spočítán ze stavu předchozího kroku mohou vznikat při velkém počtu kroků numerické odchylky v iterativním výpočtu od hodnoty, kterou udává analytický vztah pro pokles hodnoty aktivace. Z tohoto důvodu bylo provedeno základní zjištění, jaká je velikost těchto odchylek pro velké počty kroků a pro možné časové rozsahy simulace.

Pro výchozí nastavení byl zvolen testový případ **Test 4**. Časové intervaly dle tabulky na straně 51. Byly testovány dva režimy posunu času simulace a jejich vliv na případné numerické chyby ve výpočtu. Uzamčená, pevně určená velikost kroku v {T6} nastavena na **ON**. A proměnná velikost kroku, kdy je {T6} nastaveno na **OFF**. Pro provedení testu je nutné nejdříve nastavit režim učení v {L1}, potom provést krok o velikosti 1 den pomocí tlačítka `1 day` {T5}. Tím blok získá počáteční hodnotu základní aktivace o velikosti 1,285. Následně přepneme na režim odpočinku v {L1}.

Pro simulaci s pevnou velikostí kroku nastavíme {T6} na **ON** a zvolíme velikost kroku v {T7} nebo příkazem `set tick-delta <velikost>` zadaného pomocí *Command Center*. Následně zadáme do *Command Center* příkaz `go <interval>`, který posune simulaci o zvolený interval. Například `go 7` posune simulační čas o 7 dní na začátek osmého dne. Uvidíme tedy jaký je stav aktivace k danému času zobrazeného v {T4}. Vypnutí kalibračního módu v {C1} neovlivní zobrazované výsledky, ale uvidíme průběžný pokles hodnoty aktivace v zadaném intervalu v {G2}. Při simulaci s proměnnou velikostí kroku v {T6} je velikost kroku počítána také na základě hodnoty `tick-delta` {T7}, pro rychlejší posun simulace tedy nastavíme vyšší hodnotu (například `tick-delta=5`).

**Tabulka 12: Numerické odchylky výpočtu aktivace simulačního a analytického modelu (vlastní výpočet).**

| | $x_u(t_u)$ | 1,285 | krok | 1 den | krok | 1 minuta | krok | proměnný |
|---|---|---|---|---|---|---|---|---|
| | $d$ | 0,437 | $\Delta t$ | 1 | $\Delta t$ | 6,94E-04 | $\Delta t$ | tick-delta=5 |

| posun | $t$ [dny] | $x_u(t)$ | $n$ [iterace] | $\delta$ | $n$ [iterace] | odchylka | $n$ [iterace] | odchylka |
|---|---|---|---|---|---|---|---|---|
| 1 sekunda | 1,15741E-05 | 1,2850E+00 | - | - | - | - | 1 | 5,3745E-10 |
| 1 minuta | 6,94444E-04 | 1,2846E+00 | - | - | 1 | 5,0542E-06 | 1 | 2,0702E-09 |
| 1 hodina | 4,16667E-02 | 1,2623E+00 | - | - | 60 | 4,8565E-06 | 1 | 1,6017E-08 |
| 1 den | 1 | 9,4919E-01 | 1 | 6,1650E-09 | 1 440 | -1,4133E-07 | 10 | -4,3703E-09 |
| 1 týden | 7 | 5,4903E-01 | 6 | -1,2374E-09 | 8 640 | -2,3802E-07 | 61 | -1,2374E-09 |
| 1 měsíc | 30 | 2,9067E-01 | 29 | 5,4771E-09 | 41 761 | 7,4284E-08 | 184 | 5,4771E-09 |
| 3 měsíce | 90 | 1,7985E-01 | 89 | 2,3277E-08 | 128 164 | 2,3277E-08 | 405 | 2,3277E-08 |
| 1 rok | 360 | 9,8129E-02 | 359 | 2,9255E-08 | 516 979 | 2,9255E-08 | 1013 | 2,9255E-08 |
| 5 let | 1800 | 4,8568E-02 | 1 799 | -3,3313E-08 | 2 590 600 | -3,3313E-08 | 2474 | -3,3313E-08 |
| 10 let | 3600 | 3,5876E-02 | 3 599 | 8,7507E-09 | 5 182 562 | 8,7507E-09 | 4174 | 8,7507E-09 |
| 20 let | 7200 | 2,6500E-02 | 7 199 | 6,6206E-09 | 10 366 568 | 6,6206E-09 | 6299 | 6,6206E-09 |
| 50 let | 18000 | 1,7756E-02 | 17 999 | -1,5756E-08 | 25 918 576 | -1,5756E-08 | 10779 | -1,5756E-08 |

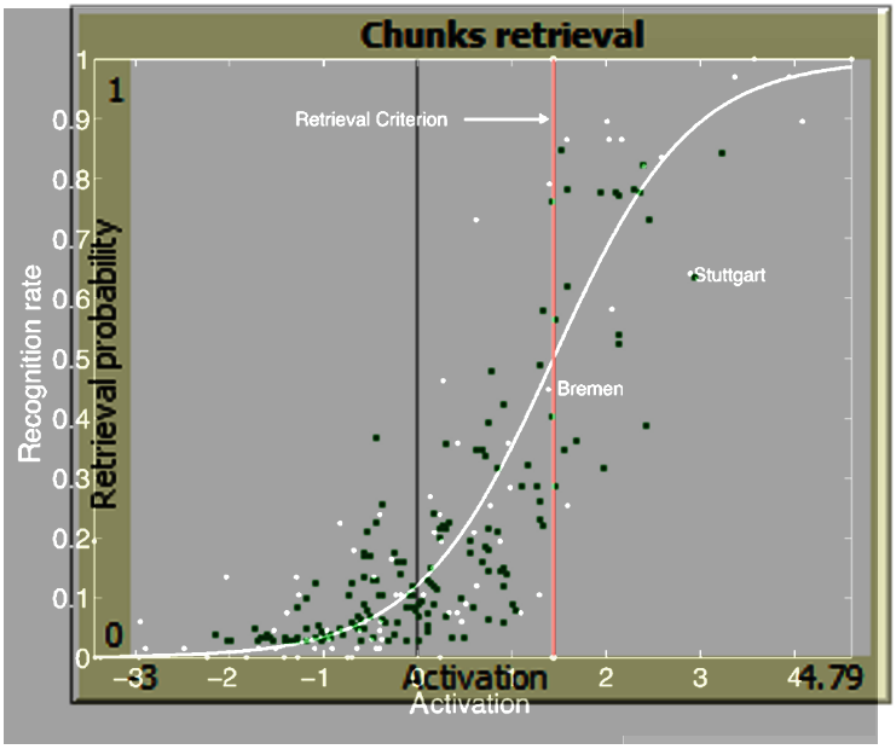
Při validaci podle uvedeného postupu byly zjištěny minimální odchylky v aktivačního hodnotách (viz Tabulka 12) ve všech uvedených režimech od hodnoty získané analytickým vztahem (viz Rovnice 14, s. 20). Změřená a zjištěná absolutní a relativní odchylka od analytického vztahu se pohybuje v řádu $10^{-6}$ až $10^{-10}$. Režim s proměnnou velikostí kroku simulačního času `lock-tick-delta?` {T6} na **ON** se dokonce ukázal být přesnější než režim s pevnou velikostí simulačního kroku. Lze jej tedy doporučit pro využití v experimentech, protože výhodou bude podstatně rychlejší, případně přesnější numerické provedení těchto experimentů. Zjištěné odchylky viz Tabulka 12.

### 3.3.3 Graf vybavení bloků

Provedeno je grafické srovnání výstupu modelu v komponentě {G3} s hodnotami získanými z experimentu popsaného v teoretické části (viz Graf 2, s. 22). Účelem porovnání bylo zatím jen provést optickou shodu rozptylů, zda je shodná s daty z experimentů při nastavení parametrů `chunk-retrieve-treshold` na hodnotu 1,44 v {E3} a `s-activation-noise` na hodnotu 0,73 v {E4}, které byly uvedeny v použitém zdroji.

Srovnánání je provedeno překrytím grafů tak, aby měly nastavený stejný rozsah obou os. Pro aktivační osu je vodítkem nulová hodnota aktivace v grafu NetLoga a limit pro vybavení nastavený na hodnotu 1,44 (viz Obrázek 11, s. 87). Data z původního grafu jsou zobrazena překryvně bílou barvou, data získaná ze simulace zeleně (viz Graf 10). Zjištěná shoda se zdá být dostatečná. V dalším kroku validace by bylo vhodné ji provést kvantitativně statistickým vyhodnocením ze získaných dat.

Pro získání dat byl vybrán simulační scénář `"Set of unrelated chunks"` v {S2}. Hodnoty parametrů `learning-intensity` v {L3} a `chunks-in-one-step` v {L4} nastaveny na hodnotu 10. Počet vytvářených bloků `chunks-count` v {S1} nastaven na hodnotu 200. Strategie učení {L2} byla vybraná `"Random"` a byl zapnut aktivační šum {E2} a nastaveny zmíněné hodnoty {E3} a {E4}. Po inicializaci modelu {T1} bylo provedeno 30 iterací simulačního cyklu {T2} s uvedeným nastavením.



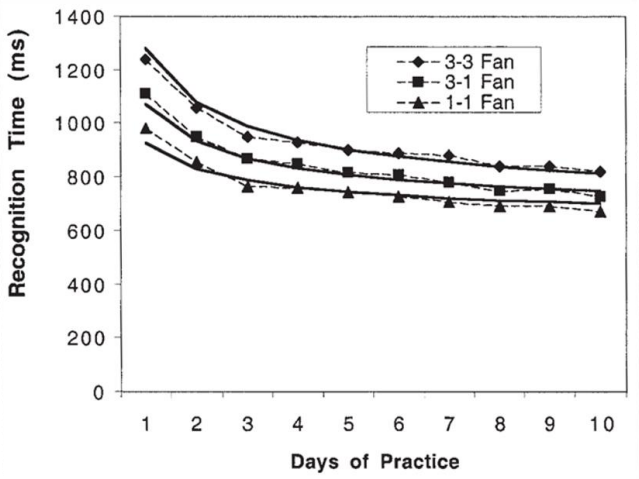

**Graf 10: Grafické srovnání výsledků experimentu ACT-R (bíle) [2] s výstupem získaným v simulačním experimentu modelu (zeleně).**

## 3.4 Kalibrace modelu

Význam a důležitost kalibrace modelu byla popsána v teoretické části (Validace a kalibrace, s. 34) a podle těchto informací byl zpracován návrh kalibračních procedur pro vytvářený model (Validace a kalibrace modelu, s. 57). Procedury byly následně implementovány a popsány (Kalibrace a kalibrační data, s. 76). V této podkapitole jsou uvedeny přesné kroky, jak byla získána kalibrační data z výsledků reálných experimentů Johna Andersona (viz Graf 2, s. 14) a jak byla tato data použita pro kalibraci modelu, aby byla umožněna validita prováděných simulačních experimentů a následná predikce podle získaných výsledků.

### 3.4.1 Příprava dat kalibrační sady

Prvním krokem bylo získat data z fan experimentů. Dostupné byly grafy obsahující naměřené časy $T_r(i)$ pro jednotlivé typy fan bloků a úrovně aktivací $A_i$, které se používají jak pro výpočty časů $T_r(i)$, tak i pro výpočet pravděpodobnosti vypavení bloků. Protože byla požadovaná data dostupná pouze v grafech v podobě rastrových obrázků, bylo je nutné obsažená data pro další zpracování převést z grafické podoby na tabulkové hodnoty. K tomu bylo využito grafického editoru, který umožnil zjistit souřadnice kurzoru v místech datových bodů. Získané souřadnice bylo možné následně přepočítat v tabulkovém procesoru na původní hodnoty, které byly pro tvorbu grafů použity.




**Graf 11: Původní podoba dat získaná měřením ve Fan experimentu dle [15]. Silně je vyznačena predikce podle stávajícího modelu ACT-R.**

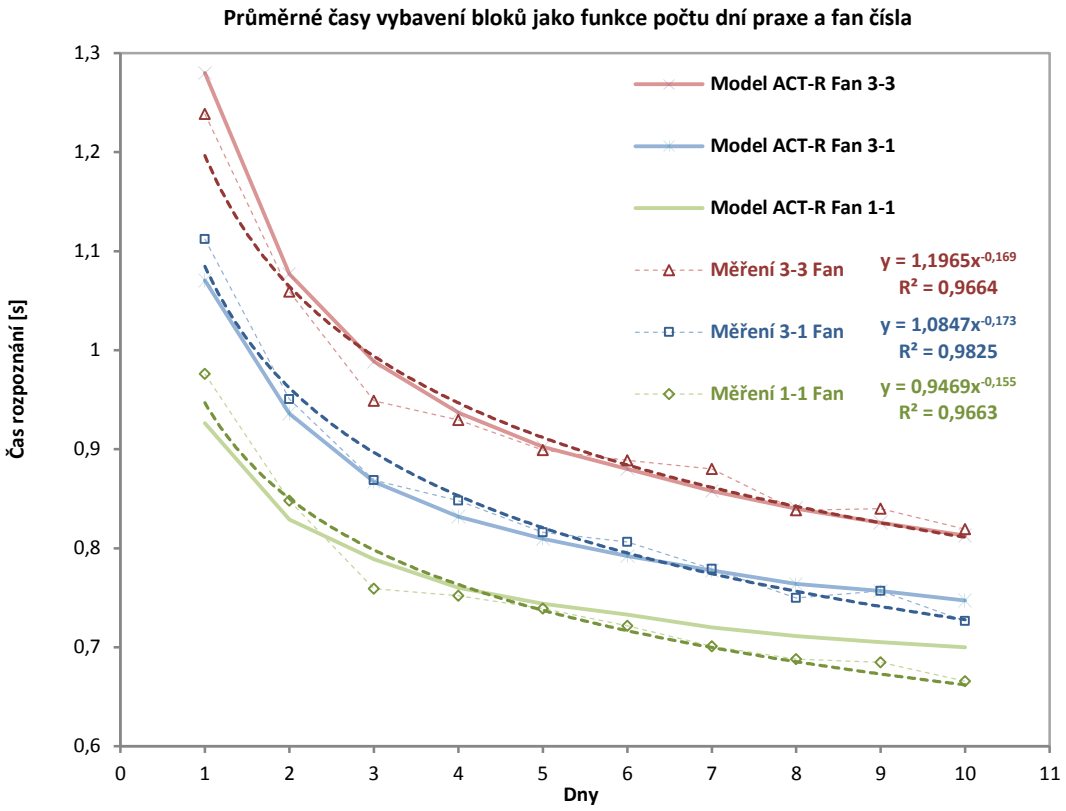
Čas $T_r(i)$ byl pro další zpracování převeden na sekundy, které byly zvoleny jako výchozí časová jednotka pro výpočty. V případě potřeby lze však čas snadno převést zpět na milisekundy jako v původních grafech. Níže je uvedena tabulka s daty získanými z grafů.

**Tabulka 13: Získaná data popisující hodnoty naměřených časů a hodnoty dle modelu ACT-R.[15]**

| Naměřené časy dle typu bloku | | | |
|---|---|---|---|
| den | 1-1 Fan [s] | 3-1 Fan [s] | 3-3 Fan [s] |
| 1 | 0,9760 | 1,1120 | 1,2384 |
| 2 | 0,8480 | 0,9504 | 1,0592 |
| 3 | 0,7592 | 0,8688 | 0,9488 |
| 4 | 0,7520 | 0,8480 | 0,9296 |
| 5 | 0,7392 | 0,8160 | 0,8992 |
| 6 | 0,7216 | 0,8064 | 0,8888 |
| 7 | 0,7008 | 0,7792 | 0,8800 |
| 8 | 0,6880 | 0,7496 | 0,8384 |
| 9 | 0,6848 | 0,7568 | 0,8400 |
| 10 | 0,6656 | 0,7264 | 0,8192 |

| Hodnoty předpovídané modelem ACT-R | | | |
|---|---|---|---|
| den | 1-1 Fan [s] | 3-1 Fan [s] | 3-3 Fan [s] |
| 1 | 1,2800 | 1,0704 | 0,9264 |
| 2 | 1,0768 | 0,9360 | 0,8288 |
| 3 | 0,9888 | 0,8672 | 0,7888 |
| 4 | 0,9368 | 0,8320 | 0,7600 |
| 5 | 0,9024 | 0,8096 | 0,7440 |
| 6 | 0,8800 | 0,7920 | 0,7328 |
| 7 | 0,8576 | 0,7776 | 0,7200 |
| 8 | 0,8400 | 0,7640 | 0,7112 |
| 9 | 0,8256 | 0,7568 | 0,7050 |
| 10 | 0,8128 | 0,7472 | 0,7000 |

Získaná data byla následně použita pro vytvoření grafu pomocí tabulkového procesoru. John Anderson uvádí pro model ACT-R celkovou korelaci 0,986 s naměřenými daty.[15] Pro ověření správnosti převodu dat do tabulkového procesoru byl tedy spočítán koeficient korelace pro data získaných z obrázků grafů a zjištěná hodnota celkové korelace je 0,98637. Převod dat s dostatečnou přesností lze považovat za úspěšný. Pro naměřená data každého typu fan bloku byl v tabulkovém procesoru také vypočítán regresní mocninný model. Důvod této volby byl popsán v teoretické části (viz Čas vybavení bloku, s. 14). Celkový koeficient korelace všech regresních modelů s naměřenými daty má hodnotu 0,9905 a je tedy dosaženo celkově nepatrně lepší shody podle korelačního koeficientu než v případě modelu ACT-R. Všechna data jsou zobrazena v následujícím grafu (viz Graf 12).




**Graf 12: Porovnání výsledků modelu ACT-R s naměřenými daty a jejich regresními mocninnými modely (vlastní zpracování). Parametry modelů jsou v legendě grafu.**

V uvedeném grafu lze vidět znatelnou zápornou odchylku naměřených dat od regresních modelů všech uvedených typů fan bloků ve třetí den měření. Běžně může být odchylka způsobena různými náhodnými vlivy. Jelikož se však projevila ve všech datových sadách fan bloků stejným způsobem a v jeden den, lze předpokládat, že vznikla buď chybou měření, nebo nenáhodným vnějším vlivem, který změnil

parametry testu. Tím bylo dosaženo kratších časů vybavení. Uvedený výkyv se následně stabilizoval.

Pokud data ze třetího dne vyřadíme z datové sady, vzroste koeficient celkové korelace modelu ACT-R s naměřenými daty na hodnotu 0,9883 a celková korelace regresních modelů potom až na hodnotu 0,9955. Navržené regresní modely lze tak považovat za vhodnou aproximaci pro časy získané měřením v experimentu. Ačkoliv mají hodnoty modelu ACT-R poměrně vysokou úroveň korelace, relativní chyba odhadu se s přibývajícími dny a zvyšujícím fan číslem zvětšuje (viz Graf 12).

Z důvodu posouzení kvality obou modelů byly tedy spočítány absolutní odchylky hodnot získaných pomocí regresních modelů a modelem ACT-R a od hodnot naměřených, tj. $\Delta x = x_{\text{model}} - x_{\text{měření}}$. Vypočítané absolutní chyby obou modelů byly poté dány do poměru $q = \left|\Delta x_{\text{ACT-R}}/\Delta x_{\text{regrese}}\right|$, který vyjadřuje o kolik má regresní model absolutně menší odchylku než uvedený model ACT-R. Pro $q = 1$ to potom znamená, že jsou odchylky obou modelů shodné, $q < 1$ vyjadřuje, že uvedený model ACT-R má menší odchylku než regresní model a naopak $q > 1$ poskytuje informaci o kolik má regresní model menší odchylku než model ACT-R. Následně byly vypočítané průměrné hodnoty $\bar{q}$, pro jednotlivé typy fan bloků. Pro vyjádření kvality modelu ACT-R z pohledu chyb v procentech, provedeme výpočet $Q = 1 - q^{-1}$. Zjištěné výsledky jsou uvedeny v následující tabulce.

| **Poměr absolutních chyb modelů $Q = \left\|\Delta x_{\text{ACT-R}}/\Delta x_{\text{regrese}}\right\|$** | | | |
|---|---|---|---|
| **den** | **$q_{\text{1-1 Fan}}$** | **$q_{\text{3-1 Fan}}$** | **$q_{\text{3-3 Fan}}$** |
| 1 | 0,9928 | 1,5238 | 1,7045 |
| 2 | 3,4937 | 1,2282 | 7,8639 |
| 3 | 0,8898 | 0,0568 | 0,7505 |
| 4 | 0,4236 | 2,9623 | 0,6774 |
| 5 | 0,2588 | 1,2587 | 3,5363 |
| 6 | 1,7974 | 1,3321 | 2,5945 |
| 7 | 1,1899 | 0,3519 | 42,4821 |
| 8 | 0,4497 | 1,9555 | 11,6097 |
| 9 | 0,9839 | 0,0000 | 1,8022 |
| 10 | 0,7617 | 10,9538 | 11,7824 |
| Průměr $\bar{q}$ | 1,1241 | 2,1623 | 8,4803 |
| **Chyba $\bar{Q}$** | **11,04 %** | **53,75 %** | **88,21 %** |

**Tabulka 14: Srovnání poměru absolutních chyb modelu ACT-R a chyb regresních modelů.**

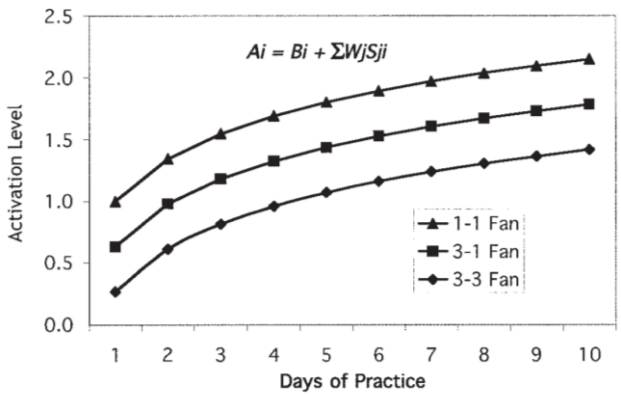
Z porovnání je patrné, že stávající model ACT-R má v případě bloku typu Fan 1-1 v rozsahu 1 až 10 dní průměrně o 11% větší chybu v porovnání s chybou regresního modelu pro naměřené časy. V případě bloku typu Fan 3-1 je chyba ACT-R modelu o 53% větší v porovnání s chybou regresního modelu pro naměřené časy. Pro

blok typu Fan 3-3 je chyba navrženého modelu ACT-R již v průměru o 88% větší v porovnání s chybou regresního modelu pro naměřené časy bloku Fan 3-3.

Ačkoliv se tyto odchylky v chybách zdají být velké, jedná se o relativní srovnání dvou typů modelů. Pro srovnání modelů platí, že pokud je vzájemná odchylka modelu ACR-R od regresního modelu 90%, potom je poměr absolutních chyb 10/1 a tedy model ACT-R má 10krát větší absolutní chybu oproti regresnímu modelu, který má nejmenší odchylku od naměřených dat. Hodnota 0% vyjadřuje, že jsou oba modely z pohledu absolutních chyb srovnatelné. Stejná metodika bude použita i pro porovnání výsledků navrženého simulačního modelu.

### 3.4.2 Výpočet parametrů modelu

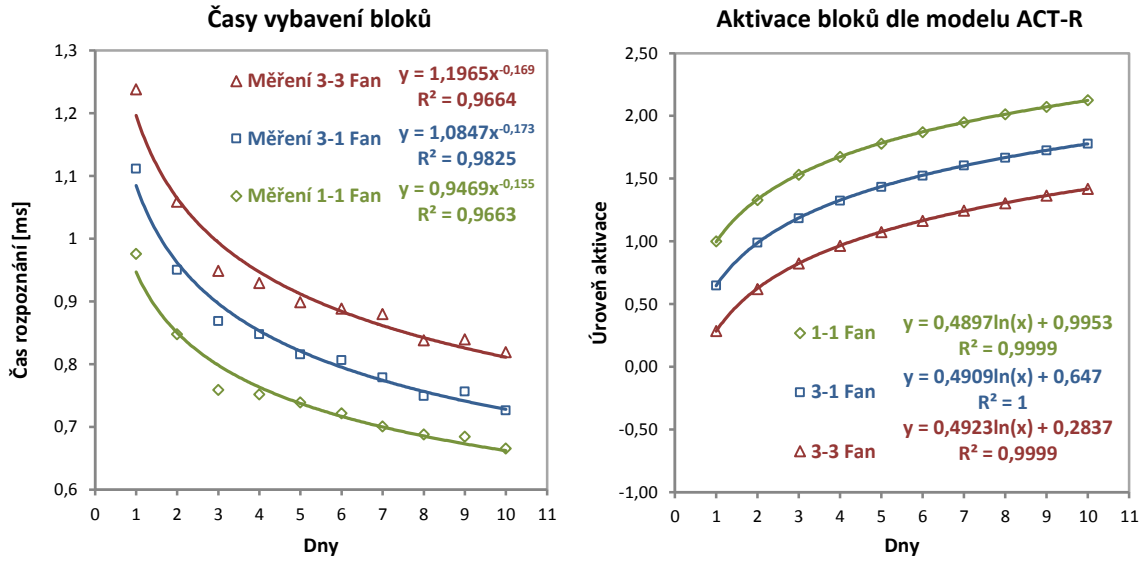
Podle dříve uvedeného vztahu (viz Rovnice 4, s. 16) lze naměřený čas $T_r(i)$ v experimentu pro daný typ bloku převést na příslušnou celkovou aktivaci $A_i$ a naopak aktivaci následně převést na čas vybavení. John Anderson uvádí ve svém článku pro navržený ACT-R model uvedené aktivace různých typů bloků. [15] Data bylo nutné opět získat podobně jako v případě časů (viz Příprava dat kalibrační sady, s. 95).




| Aktivační hodnoty dle modelu ACT-R | | | |
|---|---|---|---|
| den | 1-1 Fan [s] | 3-1 Fan [s] | 3-3 Fan [s] |
| 1 | 1,0000 | 0,6475 | 0,2875 |
| 2 | 1,3300 | 0,9900 | 0,6200 |
| 3 | 1,5325 | 1,1850 | 0,8250 |
| 4 | 1,6750 | 1,3250 | 0,9650 |
| 5 | 1,7800 | 1,4350 | 1,0750 |
| 6 | 1,8700 | 1,5250 | 1,1650 |
| 7 | 1,9500 | 1,6050 | 1,2450 |
| 8 | 2,0150 | 1,6675 | 1,3050 |
| 9 | 2,0725 | 1,7250 | 1,3650 |
| 10 | 2,1250 | 1,7800 | 1,4200 |

**Graf 13: Aktivace různých typů bloků podle návrhu ACT-R modelu a získaná data z grafu dle [15]**

Aby bylo možné převádět aktivaci $A_i$ na čas vybavení $T_r(i)$ je nutné zjistit velikost koeficientů časového měřítka $F$ a aktivačního měřítka $f$. Tyto koeficienty je možné odhadnout, ale pro přesnou shodu je vhodné provést jejich výpočet. Pro tento výpočet budeme potřebovat regresní modely získané jak z dat pro časy vybavení $T_r(i)$ (viz Graf 12, s. 96), tak pro aktivace bloků $A_i$ podle navrženého ACT-R modelu. Následně potom dva libovolně zvolené dny uvedené v tabulce hodnot. Regresní model data aproximuje s nejmenší možnou odchylkou, avšak vlastní data mohou být také zatížena různými chybami.




**Graf 14: Regresní modely pro data časů vybavení bloků a navržených hodnot aktivací dle ACT-R.**

Pro časy vybavení bloku byl zvolen mocninný regresní model. Důvod této volby byl popsán v teoretické části (viz Čas vybavení bloku, s. 14). Průběh aktivace podle návrhu teorie ACT-R nejpřesněji aproximuje logaritmický regresní model. Uvedené funkce regresních modelů byly spočítány pomocí tabulkového procesoru. V grafu aktivací navrženého modelu ACT-R je vidět, že jsou jednotlivé funkce od sebe rovnoměrně vzdáleny. Tento předpoklad potvrzují i funkce regresních modelů: $a \cdot \ln(x) + c$, kde koeficient $a$ je u všech modelů téměř identický (průměrně 0,4909). Jediný parametr, který se tak mění je konstanta $c$ přičítaná k výsledku funkce. V grafu časů vybavení mají potom všechny regresní mocninné modely $b \cdot x^{-d}$ velmi podobnou hodnotu parametru $d$ (průměrně 0,165).

Vztah $T_r(i) = F \cdot e^{-f \cdot A_i}$ potom umožní propojit uvedené dva typy modelů a převádět hodnotu času vybavení bloku $T_r(i)$ na úroveň aktivace $A_i$ a naopak (viz Rovnice 4 a Rovnice 5, s. 16). K tomu je však nutné spočítat velikost koeficientů $F$ a $f$, které oba modely uvádějí do vzájemné relace. Pro každou relaci mezi korespondujícími modely (1-1 Fan, 3-1 Fan a 3-3 Fan) by potom byly užity mírně odlišné koeficienty $F$ a $f$. V simulačním modelu je však nutné zvolit pouze jednu hodnotu pro hodnoty koeficientů. Z tohoto důvodu byla vybrána funkce aktivace a času vybavení pro datovou sadu Fan 1-1, protože ostatní funkce aktivací Fan 3-1 a Fan 3-3 jsou v ACT-R modelu pouze hrubě zvolenou aproximací (viz Graf 12, s. 96).

$$T_r(i) = 0{,}9469 \cdot x^{-0{,}155}$$

$$A_i = 0{,}4897 \cdot \ln(x) + 0{,}9953$$

$$T_r(i) = F \cdot e^{-f \cdot A_i}$$

**Rovnice 28: Vztahy použité pro výpočet koeficientů časového *F* a aktivačního *f* měřítka pro kalibraci modelu, kde *x* je zadaný počet dní.**

Uvedené rovnice řešíme jako soustavu, kde si za $x$, představující jednotlivé dny, dosadíme dvě vybrané hodnoty. Následně získáme soustavu dvou rovnic o dvou neznámých. Pro výpočet jsem si vybral hodnoty $x = 1$ a $x = 10$. Získáváme tedy:

$A_i = 0{,}4897 \cdot \ln(1) + 0{,}9953 = \mathbf{0,9953}$ $\quad$ $A_i = 0{,}4897 \cdot \ln(10) + 0{,}9953 = \mathbf{2,1229}$

$T_r(i) = 0{,}9469 \cdot 1^{-0{,}155} = \mathbf{0,9469}$ $\quad$ $T_r(i) = 1{,}0751 \cdot 10^{-0{,}091} = \mathbf{0,6627}$

Soustava rovnic pro výpočet parametrů $F$ a $f$ bude mít potom následující podobu:

$$0{,}9469 = F \cdot e^{-f \cdot 0{,}9953}$$

$$0{,}6627 = F \cdot e^{-f \cdot 2{,}1229}$$

Řešením získáváme hodnoty $F = 1{,}297498$ a $f = 0{,}316487$. Správnost spočítaných parametrů lze zjistit dosazením jedné z vybraných hodnot aktivace $A_i$ použitých pro výpočet do vztahu:

$$T_r(i) = 1{,}2975 \cdot e^{-0{,}3165 \cdot A_i}$$

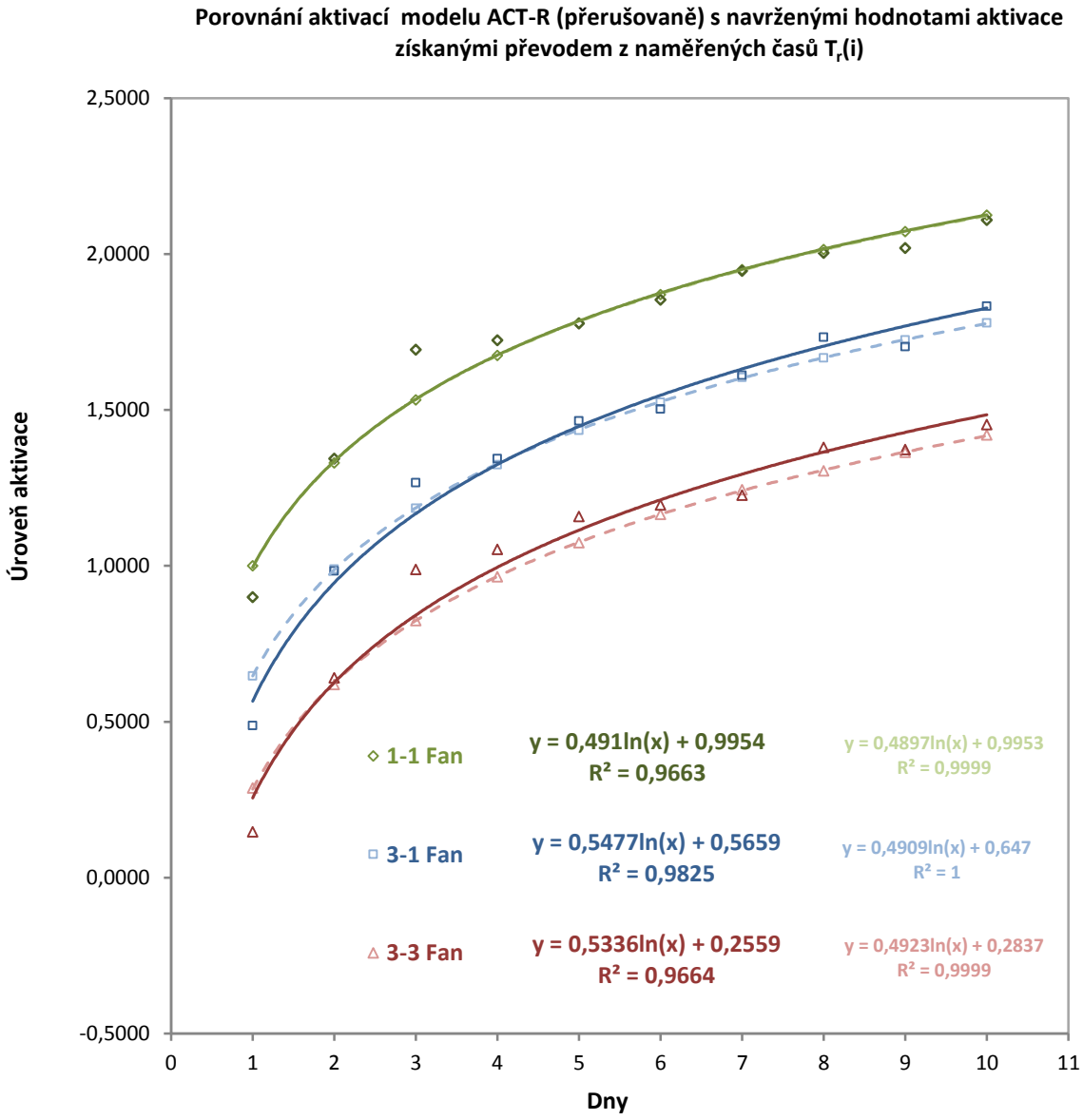
Výpočtem s hodnotou například $A_i = 2{,}1229$ získáme tedy čas vybavení $T_r(i)$ pro danou úroveň aktivace podle regresního modelu, v tomto případě $T_r(i) = 0{,}6627$. Je možné také použít inverzní vztah (viz Rovnice 5, s. 16) pro výpočet hodnoty aktivace z času vybavení:

$$A_i = \ln\left(1{,}2975 \cdot \left(T_r(i)\right)^{-1}\right) \cdot 0{,}3165^{-1}$$

Ověřit jeho správnost lze například dosazením hodnoty $T_r(i) = 0{,}6627$, čímž získáme odpovídající hodnotu aktivace podle regresního modelu, a tedy úroveň aktivace $A_i = 2{,}1228$.

Zjištěním velikosti hodnot koeficientů F a f jsme parametrizovali matematický model, který umožní vygenerovat data kalibrace pro aktivace bloků z naměřených časů $T_r(i)$ tak, aby hodnoty regresního modelu času vybavení pro data Fan 1-1 odpovídaly daným úrovním aktivací podle modelu ACT-R. Stejný vztah potom použijeme i pro převod všech ostatních časů měření u datových sad Fan 3-1 a Fan 3-3. Tím získáme hodnoty aktivací pro kalibraci modelu. V následujícím grafu jsou porovnány hodnoty aktivací pro fan bloky různého typu dle ACT-R modelu a hodnoty

aktivací získané převodem podle uvedeného vztahu z časů $T_r(i)$ naměřených v experimentu.




**Graf 15: Porovnání modelu aktivace dle ACT-R s modelem aktivace získaným z naměřených dat.**

Převedené hodnoty aktivací z časů měření $T_r(i)$ byly aproximovány logaritmickým regresním modelem, který umožní vygenerovat kalibrační sadu obsahující takové hodnoty, aby měly co nejmenší odchylku od naměřených dat (viz Graf 15). Z grafu je patrná shoda modelu s hodnotami Fan 1-1 podle očekávání, avšak navržený model podle ACT-R se v případě bloků typu Fan 3-1 a Fan 1-1 odchyluje od hodnot naměřených v experimentech. Odchylka je relativně malá, avšak může docházet k jejímu zvětšování se zvyšujícím se počtem dnů.

| $A_i = a \cdot \ln(t) + c$ | | $A_i = \ln(F/T_r(i)) \cdot f^{-1}$ | | pro $F = 1,2975$ a $f = 0,3165$ | | | |
|---|---|---|---|---|---|---|---|
| $a$ | 0,4910 | 0,5477 | 0,5336 | $a$ | 0,4910 | 0,5477 | 0,5336 |
| $c$ | 0,9954 | 0,5659 | 0,2559 | $c$ | 0,9954 | 0,5659 | 0,2559 |

| $t$ [dny] | $A_{1-1\ \mathrm{Fan}}$ | $A_{3-1\ \mathrm{Fan}}$ | $A_{3-3\ \mathrm{Fan}}$ |
|---|---|---|---|
| 1 | 0,9954 | 0,5659 | 0,2559 |
| 2 | 1,3357 | 0,9455 | 0,6258 |
| 3 | 1,5348 | 1,1676 | 0,8421 |
| 4 | 1,6761 | 1,3252 | 0,9956 |
| 5 | 1,7856 | 1,4474 | 1,1147 |
| 6 | 1,8752 | 1,5472 | 1,2120 |
| 7 | 1,9508 | 1,6317 | 1,2942 |
| 8 | 2,0164 | 1,7048 | 1,3655 |
| 9 | 2,0742 | 1,7693 | 1,4283 |
| 10 | 2,1260 | 1,8270 | 1,4846 |

| $t$ [dny] | $A_{1-1\ \mathrm{Fan}}$ | $A_{3-1\ \mathrm{Fan}}$ | $A_{3-3\ \mathrm{Fan}}$ |
|---|---|---|---|
| 11 | 2,1728 | 1,8792 | 1,5354 |
| 12 | 2,2155 | 1,9269 | 1,5818 |
| 13 | 2,2548 | 1,9707 | 1,6246 |
| 14 | 2,2912 | 2,0113 | 1,6641 |
| 15 | 2,3251 | 2,0491 | 1,7009 |
| 16 | 2,3567 | 2,0844 | 1,7354 |
| 17 | 2,3865 | 2,1177 | 1,7677 |
| 18 | 2,4146 | 2,1490 | 1,7982 |
| 19 | 2,4411 | 2,1786 | 1,8271 |
| 20 | 2,4663 | 2,2067 | 1,8544 |

**Tabulka 15: Hodnoty pro kalibraci aktivace bloku podle časů naměřených ve fan experimentu.**

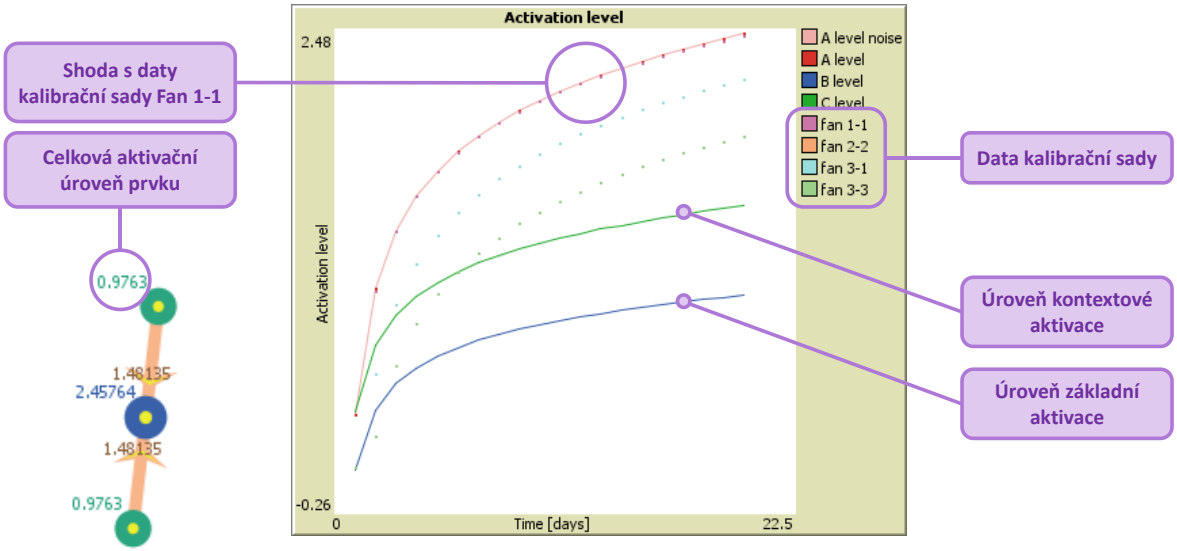
Uvedený postup lze použít pro jakékoliv naměřené údaje $T_r(i)$, aby z nich bylo možné vytvořit kalibrační sadu obsahující hodnoty aktivací. V implementovaném modelu je uvedená sada (viz Tabulka 15) v komponentě `calibration-set` {C3} označena jako `Activation by time data (1974)`. Pro více informací o implementaci kalibračních sad viz Ukázka kódu 32, s. 76.

### 3.4.3 Význam a vliv parametrů modelu na výsledky

Před vlastním postupem kalibrace bude popsán význam jednotlivých parametrů modelu na pozorované výsledky aktivací. Vybereme si tedy v modelu kalibrační sadu `Activation by time data (1974)` v {C3} a nastavíme model do výchozí kalibrace pomocí tlačítka `default-calibration` {C5}. Následně vybereme v `scenario` {S2} simulační scénář `"ACT-R fan 1-1 calibration"` a stiskneme tlačítko `test-calibration-run` {C4}. Model bude automaticky nastaven do výchozího stavu a proběhne 19 iterací simulačního cyklu, které odpovídají devatenácti po sobě jdoucím dnům experimentu s blokem typu Fan 1-1.

V grafu aktivací vidíme celkovou aktivaci *A* bloku (červeně), která je tvořena základní aktivací *B* bloku (modře) a kontextovou aktivací *C* bloku (zeleně). Pokud kontextovou aktivaci vypneme nastavením `use-context-cue?` {X2} na **OFF** a opět stiskneme tlačítko `test-calibration-run` {C4}, tak je nyní celková základní aktivace *A* bloku rovna základní aktivaci *B* bloku. Důvodem je, že kontext, a tedy relace se zdroji aktivace nyní nezvyšují celkovou aktivaci bloku. Toto je vidět jak v náhledu modelu {G1}, tak v grafu aktivací {G2}.

Model můžeme také krokovat ručně pomocí jeho nastavení do počátečního stavu tlačítkem `setup` {T1} a následně opakovaným stiskem tlačítka `go-once` {T2}, kterým posuneme čas vždy o jeden den. Modelovaný učící se agent je stále v režimu učení, takže s každým dnem je provedeno užití bloku, a tedy bloku připsán zisk aktivace díky tomuto užití. Při přepnutí zobrazovaných hodnot bloku `chunk-value-display` {V4} na `"Retrieve time"` je zobrazován čas vybavení bloku v sekundách podle hodnoty celkové aktivace *A* bloku a se zvyšující se aktivací tento čas klesá. Je možné také vyzkoušet scénáře `"ACT-R fan 3-1 calibration"` a `"ACT-R fan 3-3 calibration"` v `scenario` {S2} a zkontrolovat shodu s kalibračními daty (viz Tabulka 13, s. 95) a odchylky od regresního modelu (viz Graf 12. s. 96).




**Obrázek 13: Zobrazení bloku Fan 1-1 s grafem úrovně aktivace obsahujícím body kalibrační sady.**

Aktuální hodnota aktivace bloku v simulaci modelu je výsledkem dvou protichůdných efektů, které působí v modelu zároveň a jejichž velikost lze řídit nastavením jeho parametrů. Jedná se o zisk hodnoty aktivace nastavovaný pomocí parametrů `chunk-gain` {B1}, `relation-gain` {R1}, `fan-gain-coef` {R3} a `diff-gain-coef` {R5} a ve stený okamžik také ztráty hodnoty aktivace díky její degradaci, která je řízena prostřednictvím parametrů `decay-coef` {D2}, `chunk-dec-rate` {B2}, `relation-dec-rate` {R2}, `fan-decay-coef` {R4} a `diff-dec-coef` {R6}. Všechny uvedené parametry modelu tak dávají kontrolu nad celým základním nastavením modelu a umožňují jej zkalibrovat pro jakákoliv získaná

data z experimentů. Kalibrovat model tedy znamená nastavit přesnou rovnováhu mezi ziskem hodnoty aktivace a její degradací.

V modelu při simulaci zároveň vzniká několik zpětných vazeb, kdy velikost aktuální úrovně aktivace bloku je závislá na předchozí úrovni aktivace a stavu modelu. Ta vznikla na základě různě časově posunutých užití bloku, jejichž hodnoty aktivace byly vytvořeny a následně degradovány podle nastavení uvedených parametrů. Hodnotami těchto parametrů je pak řízen vliv těchto zpětných vazeb na výslednou aktivaci. Nastavení modelu je díky zpětné vazbě tak velice citlivé, kdy pro kalibraci s co největší mírou shody s kalibračními daty jsou hodnoty parametrů nastavovány v rozlišení tisícin. Pro více informací o zpětné vazbě viz kapitola Zpětná vazba na straně 25.

### 3.4.4 Postup kalibrace modelu

Závislosti mezi změnami parametrů modelu jsou často nelineární a změna jednoho parametru komplexním, avšak do jisté míry odhadnutelným způsobem ovlivní celkový průběh úrovně aktivace bloku. Při kalibraci modelu je vhodné vyjít z existujícího nastavení a upravit jej experimentálně pro jinou nebo vlastní kalibrační sadu (viz Příprava dat kalibrační sady, s. 95).

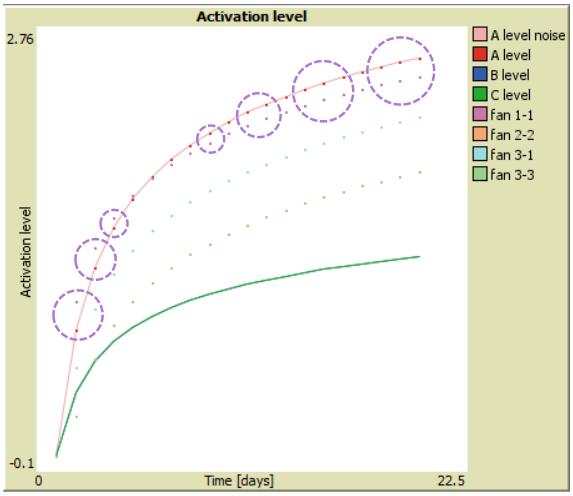
V případě algoritmického přístupu k nastavení modelu si nejdříve musíme uvědomit, že cílem je vyvážit efekt zisku aktivace (pozitivní zpětnou vazbu) její ztrátou v podobě degradace aktivační hodnoty (negativní zpětná vazba). Kombinovaný efekt obou těchto typů vazeb musí dosáhnout rovnováhy v bodech určených kalibrační sadou pro daný typ fan bloku.

Pro úplně výchozí nastavení zvolíme kalibrační sadu `Validation` v `calibration-set` **{C3}** a pomocí tlačítka `default-calibration` **{C5}** nastavíme parametry modelu. Model přepneme na sadu `Activation by time data (1974)` v `calibration-set` **{C3}**, zvolíme typ simulačního scénáře `ACT-R fan 1-1 calibration` v `scenario` **{S2}** a provedeme kalibrační běh simulace pomocí `test-calibration-run` **{C4}**. Pro přesnější nastavení je vhodné graf aktivace v NetLogo modelu dočasně zvětšit.

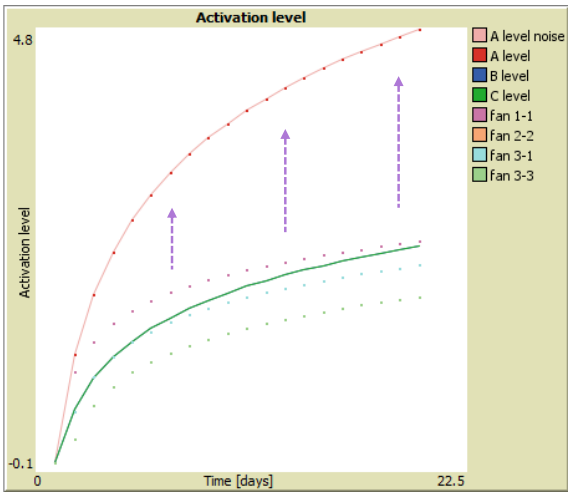
V grafu (viz Graf 16) aktivací vidíme, že základní i kontextová aktivace mají stejnou velikost. Celková aktivace bloku nejdříve ztrácí oproti kalibračním hodnotám a po jistém čase naopak hodnotu kalibračních dat překoná. Podle ACT-R se používá

pro degradaci paměti základní hodnota 0,5 a v případě dlouhodobé paměti hodnota nižší. Nejlepších výsledků heuristického rozhodování lze dosáhnout při hodnotě $d$ přibližně v rozsahu 0,25 až 0,34 (viz Graf 3, s. 23).

Zvolíme proto výchozí hodnotu například $d = 0{,}3$ jak pro degradaci bloků `chunk-dec-rate` **{B2}**, tak pro degradaci relací `relation-dec-rate` **{R2}** a opět provedeme kalibrační běh pomocí **{C4}**. Z uvedeného grafu (viz Graf 17) je vidět, že při snížení hodnoty $d$ na hodnotu 0,3 úroveň aktivace bloku prudce vzroste. Důvodem je, že každé množství aktivace užití bloku i relací nyní degraduje pomaleji. Následkem je díky převládající pozitivní zpětné vazbě v zisku aktivace kumulativní efekt projevující se jako růst aktivace. Zisk i pokles aktivace je stále v rovnováze, avšak zisk nyní výrazně převládá nad degradací.




**Graf 16: Průběh aktivace Fan 1-1 s $d = 1$ pro bloky i relace s vyznačenou odchylkou od dat kalibrace.**

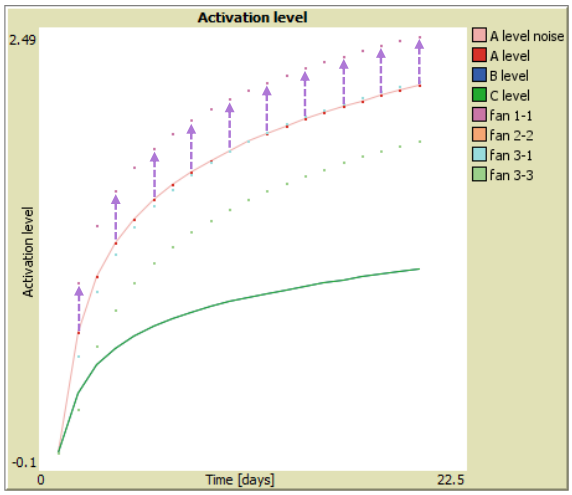



**Graf 17: Průběh aktivace Fan 1-1 se snížením míry degradace $d$ na hodnotu 0,3 pro bloky i relace.**

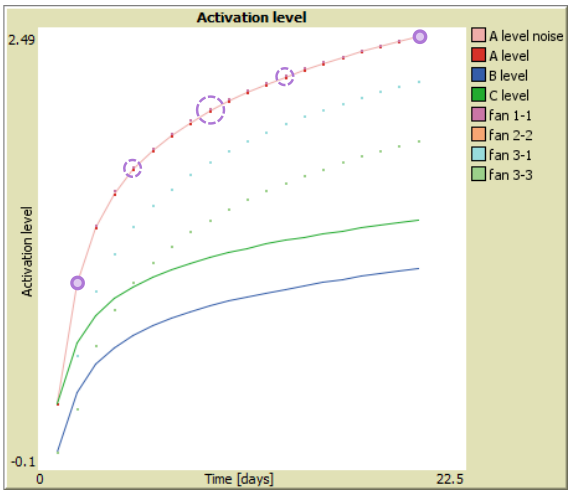
Následně přistoupíme k nastavení nelineárního poklesu aktivace pomocí koeficientu `decay-coef` **{D2}**. Pokles hodnot je řízen uvedeným vztahem (viz Graf 7, s. 48), a proto bude prvních několik dní pro každou aktivaci degradace rychlejší. Maxima dosáhne při $t = e = 2{,}718$ a poté bude asymptoticky klesat, a tedy zpomalovat k nastavené hodnotě $d = 0{,}3$.

Zvyšujeme tedy hodnotu `decay-coef` **{D2}** a sledujeme pokles aktivace bloku (viz Graf 18). Při ideálním nastavení hodnoty `decay-coef` musí modelovaná aktivace bloku přesně kopírovat kalibrační data. Experimentováním s nastavením bylo zjištěno, že k nejlepší shodě modelované aktivace s kalibračními daty dochází při volbě `decay-coef` překvapivě kolem hodnoty `3.14`. Z tohoto důvodu bylo zvoleno

`decay-coef` $\cong \pi \cong 3{,}1416$ a maximum dosáhne degradace $d = 1$ jak již bylo uvedeno při hodnotě $t \cong 2{,}718$ a následně asymptoticky klesá k nastavené hodnotě.



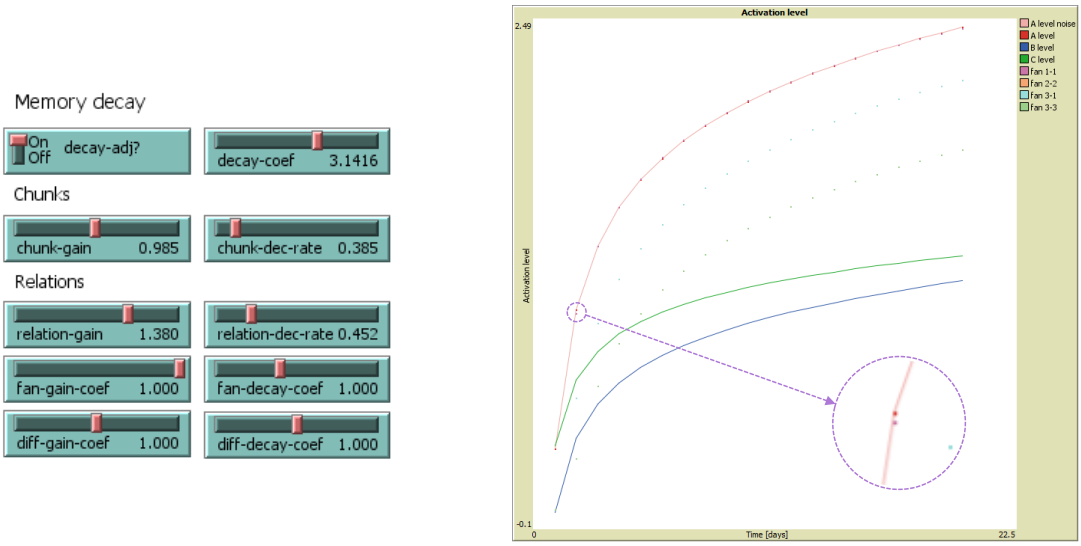

**Graf 18: Nastavení hodnoty `decay-coef` na 3,1416 a degradace *d* bloku i relace na hodnotu 0,410.**




**Graf 19: Zvýšení `relation-gain` na 1,330. V grafu jsou vyznačena místa přesné shody i odchylky.**

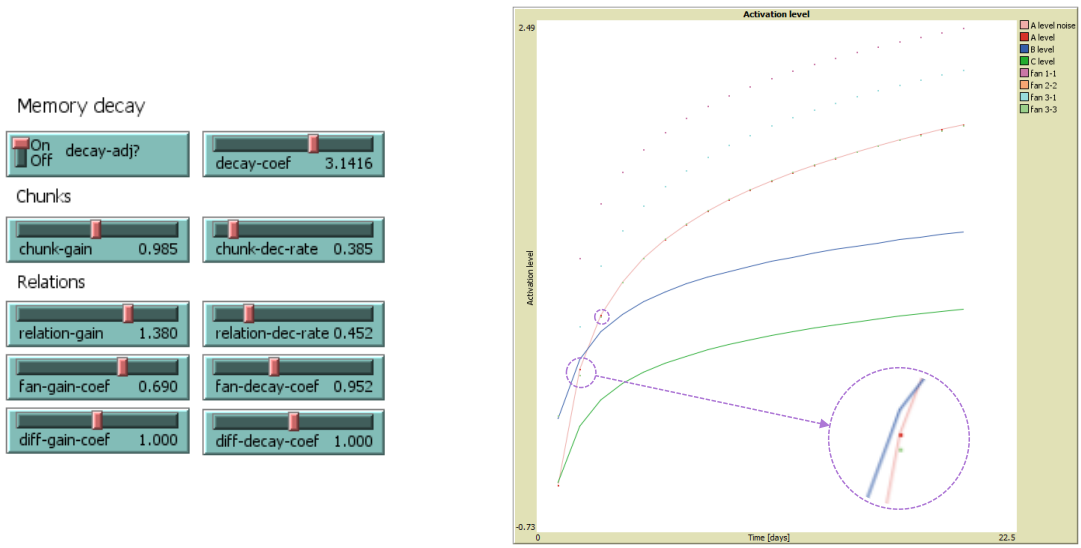
Při této volbě je ale aktivace stále příliš vysoká a je nutné zvýšit degradaci relace `chunk-dec-rate` {B2} i bloku `relation-dec-rate` {R2}, dokud není aktivace vertikálně posunutá o jistou konstatní hodnotu oproti kalibračním datům bloku Fan 1-1 (viz Graf 18), například na hodnotu $d = 0{,}415$. Následně provedeme zvýšení zisku relace `relation-gain` {R1} na hodnotu 1,330, který posune úroveň aktivace o konstatní hodnotu vzhůru (viz Graf 19 oproti Graf 18). Nyní je již dosaženo poměrně dobré shody s kalibračními daty. Při nastavování parametrů lze postupovat iterativně. Nejdříve nastavit {B2} a {R2}, následně {R1}. Po každém nastavení zkontrolovat výsledek kalibračním během {C4} a celý postup opakovat dokud není dosažena přesná shoda pro první a poslední den kalibrační sady (viz Graf 19).

Modelovaná aktivace bloku již poměrně dobře aproximuje kalibrační data. V grafu je nicméně vidět malá odchylka u hodnot ve středu kalibrační sady. Tuto odchylku lze eliminovat nastavením odlišné rychlosti degradace bloku a relace. Předpokladem je, že základní aktivace má menší rychlost degradace než kontextová aktivace, nicméně zde je jistý prostor pro experimentování s nastavením. Pokud jsme ochotni akceptovat jistou odchylku od kalibračních hodnot pro první den, můžeme použít například nastavení podle viz Graf 20.



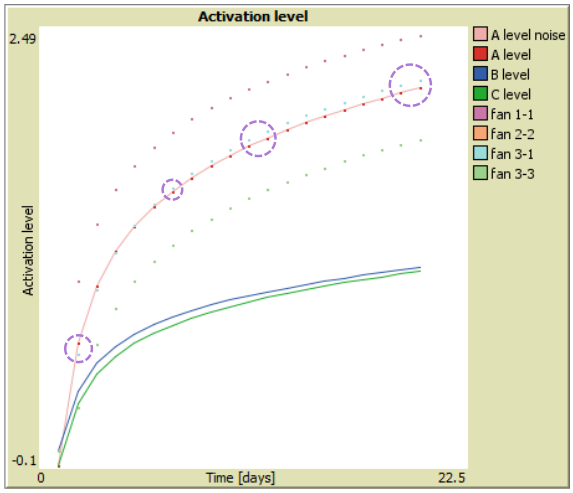

**Graf 20: Míra shody aktivace bloku typu Fan 1-1 v modelu s kalibračními daty pro uvedené nastavení.**

Tímto je model kalibrován pro blok typu Fan 1-1. Následuje kalibrace modelu pro blok vyšším symetrickým fan číslem. Jedná se o bloky typu Fan 2-2 nebo Fan 3-3 podle dostupných kalibračních dat. Pro uvedenou kalibrační sadu `Activation by time data (1974)` vybereme simulační scénář `ACT-R fan 3-3 calibration` v `scenario` **{S2}**. Cílem je snížit hodnotu koeficientů `fan-gain-coef` **{R3}** a `fan-decay-coef` **{R4}** tak, aby bylo dosaženo co největší shody s příslušnými kalibračními daty. Toto nastavení potom vychází z nastavení Fan 1-1, které však neovlivňuje. Je tedy nejdříve nutné dosáhnout co nejlepší kalibrace pro Fan 1-1. Jako vhodným nastavením se jeví hodnoty 0,690 pro `fan-gain-coef` a 0,952 pro `fan-decay-coef`. V této fázi je již model velice citlivý na nastavení parametrů a jejich absolutní změny hodnot se pohybují v řádu $10^{-3}$. V grafu lze vidět malé odchylky u dní na začátku kalibrační sady (viz Graf 21).

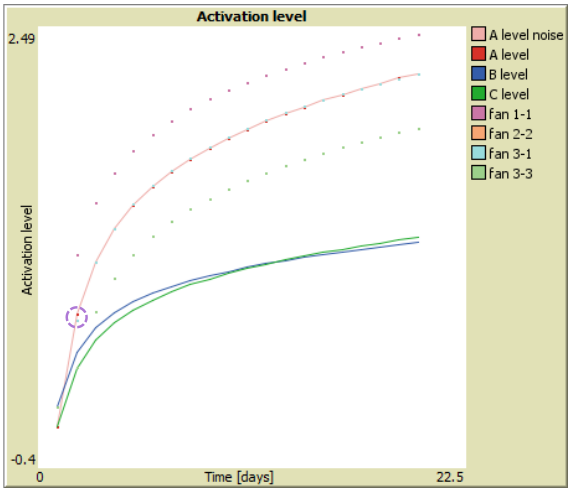


**Graf 21: Míra shody aktivace bloku typu Fan 3-3 v modelu s kalibračními daty pro uvedené nastavení.**

Poslením krokem je kalibrace modelu pro bloky s asymetrickými fan čísly. V tomto případě se jedná o blok s číslem Fan 3-1. Vybereme simulační scénář `ACT-R fan 3-1 calibration` v `scenario` {S2} a následně pomocí nastavení parametrů `diff-gain-coef` {R5} na hodnotu 0,976 a `diff-decay-coef` {R6} na hodnotu 0,962 provedeme korekci modelovaná aktivace bloku.(viz Graf 23).




**Graf 22: Odchylka aktivace od kalibračních hodnot pro blok typu Fan 3-1 po kalibraci Fan 3-3.**




**Graf 23: Kalibrovaná modelovaná aktivace bloku typu Fan 3-1 podle kalibračních dat.**

Díky tomu, že existuje mnoho způsobů, jak lze nastavit model tak, aby byl ve scénáři `ACT-R fan 1-1 calibration` ve shodě pro kalibrační sadu Fan 1-1 nabízí uvedená možnost experimentování s různým nastavením modelu. Reálný poměr mezi základní a kontextovou aktivací je neznámý a může se u jednotlivých modelovaných agentů lišit. Tímto lze simulovat větší nebo menší důraz na kontext nebo základní aktivaci. Příklad výsledného nastavení podle uvedeného postupu zobrazuje následující tabulka:

| Sada nastavení parametrů pro kalibrační sadu: *Activation by time data (1974)* | | | | | |
|---|---|---|---|---|---|
| {D1} | `decay-adj` | **ON** | {D2} | `decay-coef` | **3,1416** |
| {B1} | `chunk-gain` | **0,985** | {B2} | `chunk-dec-rate` | **0,385** |
| {R1} | `relation-gain` | **1,380** | {R2} | `relation-deg-rate` | **0,452** |
| {R3} | `fan-gain-coef` | **0,690** | {R4} | `fan-decay-coef` | **0,952** |
| {R5} | `diff-gain-coef` | **0,972** | {R6} | `diff-decay-coef` | **0,962** |
| {A1} | `time-scaling` | **1,2975** | {A2} | `activation-scaling` | **0,3165** |

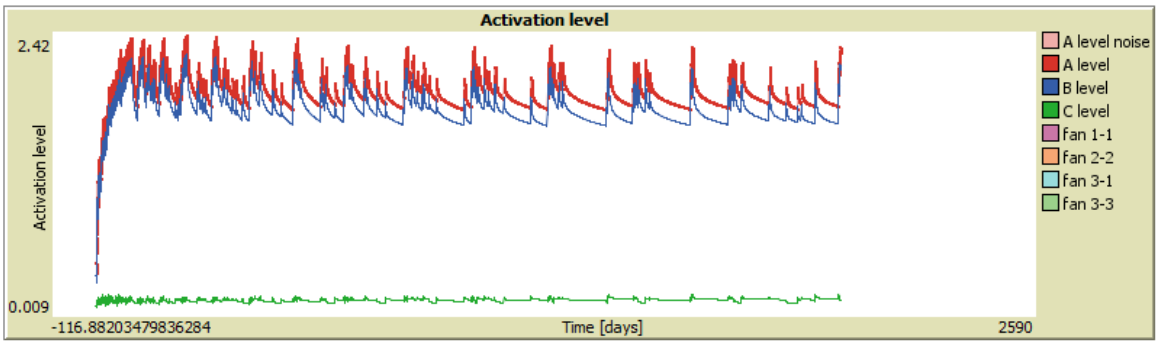
**Tabulka 16: Nastavení parametrů modelu podle hodnot kalibrační sady *Activation by time data (1974)* s větším vlivem základní aktivace z důvodu její pomalejší degradace (data zjištěny experimentálně).**

## 3.5 Experimenty

Experimenty s modelem se zaměřují na zjištěné emergentní chování modelu v různých scénářích v `scenario` {S2} v kombinaci s nastavením strategie učení v `learner-strategy` {L2} a podle hodnot kalibrace modelu viz Tabulka 16. Nejdůležitější praktickou strategií se jeví strategie `Bellow Threshold` (viz Tabulka 7, s. 56), kdy je blok naučen, a tedy použit, pokud dojde k poklesu jeho úrovně aktivace pod limit vybavení. Limit je nastavován pomocí `chunk-retrieve-treshold` {E3}. Úroveň aktivace bloku může být ovlivněna aktivačním šumem povolovaným pomocí `activation-noise?` {E2}. Velikost tohoto šumu je řízena parametrem `s-activation-noise` {E4} (viz Ukázka kódu 14, s. 59). Při experimentech nastavíme model, následuje inicializace počátečního stavu pomocí `setup` {T1} a potom spuštění simulace pomocí `go` {T3}. Pro záznam potřebných dat je vhodné doplnit kód modelu vhodnými příkazy.

### 3.5.1 Experiment 1 - Efekt opakovaného učení

Při opakovaném učení bloku dochází k jeho dlouhodobějšímu uchování v paměti díky pomalejšímu poklesu jeho aktivace. Toto se projeví prodloužením nutných intervalů mezi jednotlivými vybaveními bloku. V tomto experimentu byl zvolen scénář `Numbers 0 to 9 next/prev` v `scenario` {S2} (viz Tabulka 6, s. 56). Tím je vytvořeno 20 bloků a 11 zdrojů aktivace. Vybraná strategie v `learner-strategy` {L2} je `Bellow Threshold` (viz Tabulka 7, s. 56). Limit vybavení $\tau$ je nastaven na hodnotu 1.0 v `chunk-retrieve-treshold` {E3} a šum v aktivačních úrovních $s$ nastaven v `s-activation-noise` {E4} na hodnotu 0,4 (viz Rovnice 6, s. 16). Počet bloků učených v jeden den `chunks-in-one-step` {L4} je nastaven na hodnotu 20 z důvodu, že průměrně by měl být blok naučen jednou za den. Režim kalibrace {C1} je vypnutý.

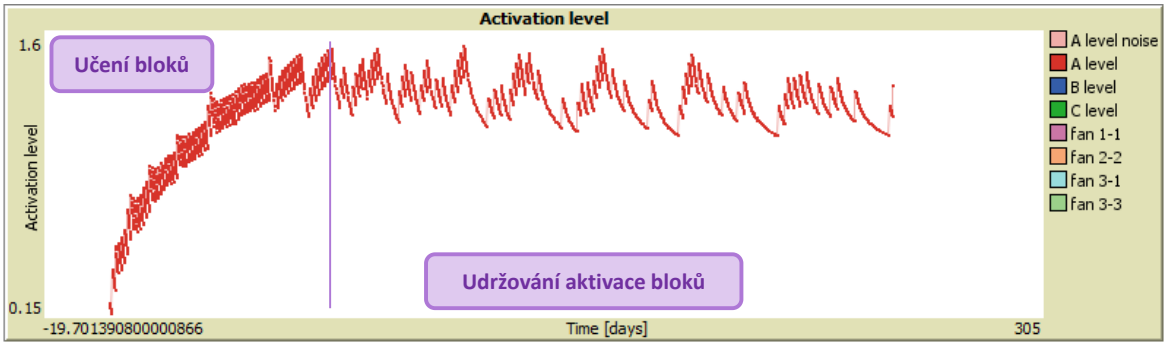



**Graf 24: Aktivace 20 bloků učených v průměru jednou za den při strategii *Bellow Threshold.* (simulace)**

Z výsledného grafu (viz Graf 24) je patrné, že se intervaly mezi nutným vybavováním bloku postupně prodlužují. Průběh aktivace tvoří po jistém čase vzor, který se opakuje, avšak díky malé asymetrii v degradacích aktivací podle fan čísel a rozdílným počátečním podmínkám není zcela periodický, i když je šum v úrovních aktivacích bloků vypnutý.

### 3.5.2 Experiment 2 - Vliv kontextu na učení a zapomínání

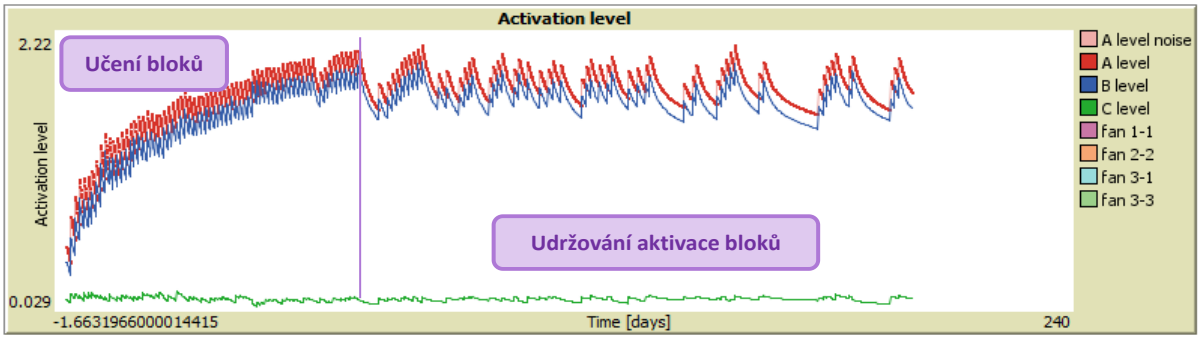
Cílem experimentu bylo zjistit, jaký vliv má učení v kontextu s využitím relací a učení bez kontextu na rychlost naučení bloků a jejich následné zapomínání. Pro tuto úlohu byl zvolen scénář `Numbers 0 to 9 next` v `scenario` {S2}. Vytvořeno je tím 10 bloků a 11 zdrojů aktivace. Vybraná strategie v `learner-strategy` {L2} je `Bellow Threshold`. Limit vybavení a šum je nastaven jako v Experimentu 1. Počet bloků učených v jeden den `chunks-in-one-step` {L4} je nastaven na hodnotu 10 z důvodu, že průměrně by měl být blok naučen jednou za den, aby byly splněny podmínky, za kterých probíhala kalibrace modelu. Režim kalibrace {C1} je vypnutý. Vytváření relací, a tedy kontext, je zapínán a vypínán pomocí {S3}.




**Graf 25: Průměrná úroveň aktivace 10 nesouvisených bloků v bezkontextovém režimu. (simulace)**

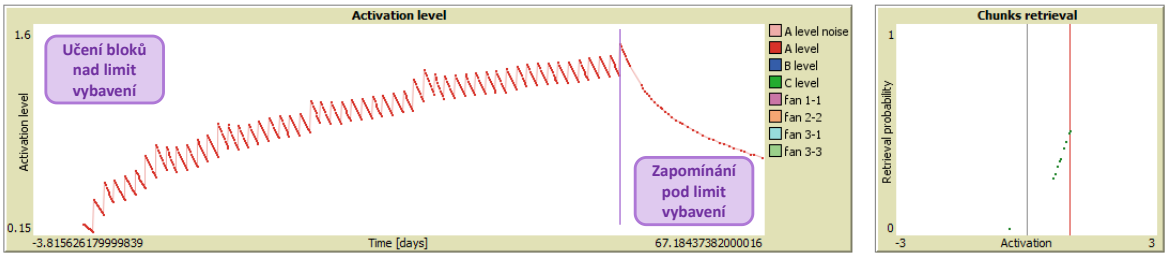
Při experimentu byly zjištěny dvě fáze. V první fázi dochází k učení nových bloků a naučené bloky jsou udržovány opakováním nad úrovní nutnou podle teorie

ACT-R pro vybavení bloku. V druhé fázi jsou všechny bloky již naučeny a je tedy prováděno pouze udržování jejich aktivace nad limitem vybavení. Díky opakování naučených bloků dochází k pomalejší degradaci jejich aktivace a interval mezi nutným opakováním se začíná prodlužovat. Tento jev byl pozorován jak v bezkontextovém režimu, kdy není užíváno relací (viz Graf 25), tak při režimu kontextovém (viz Graf 26).




**Graf 26: Průměrná úroveň aktivace 10 bloků, které jsou propojeny relacemi v kontextovém režimu.**

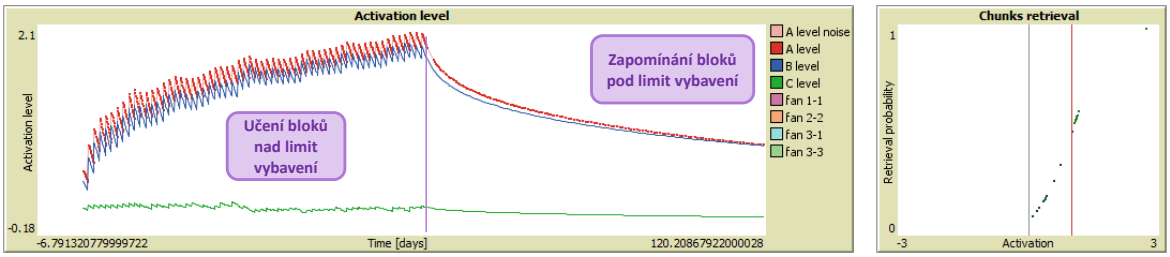
Zajímavým výsledkem je fakt, že v bezkontextovém režimu, kdy nejsou využívány relace mezi bloky, trvalo naučení všech 10 bloků přibližně 54 simulovaných dní. Při použití relací mezi koncepty trvalo učení všech 10 bloků v průměru přibližně 59 simulovaných dní, a tedy téměř o 10 % déle, než u bezkontextového učení. V některých případech potom však až 70 dní, tedy o 30 % déle. Zjištěné výsledky by podpořily argument, že učení úplně nových a neznámých informací bez souvislostí přináší výsledky rychleji, a je tedy efektivnější strategií za předpokladu, že je vyžadováno pouze udržení daného bloku v paměti pro vybavení.



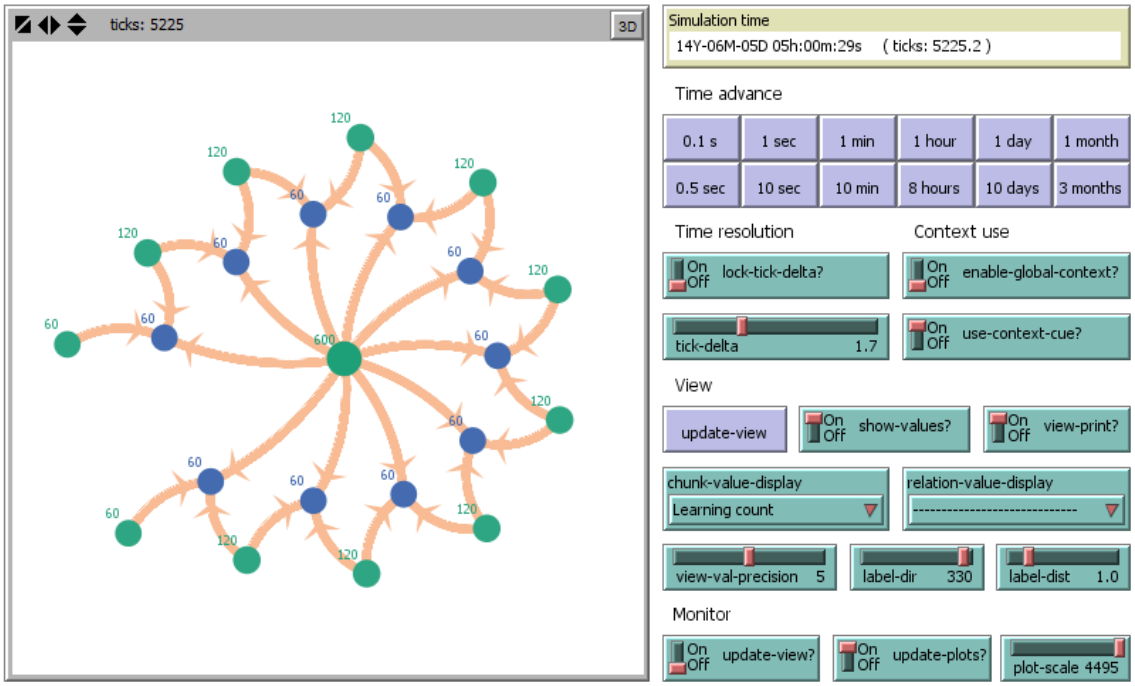

**Graf 27: Zapomínání bloků při učení bez kontextu. (simulace)**

Pokud však uvážíme následné zapomínání, je výsledek zcela opačný. Při učení bez souvislostí dojde k poklesu všech bloků pod limit vybavení již po 13 dnech (viz Graf 27). Pokud jsou do výpočtu zahrnuty relace, a tedy i kontext, potom při učení v délce 61 dní dochází k následnému poklesu pod limit vybavení pro poslední z bloků po 59 dnech (viz Graf 28). Blok, který je zdrojem aktivace pro všechny související

bloky, poklesne pod úroveň vybavení až po 14 letech, protože byl použit celkem 600 krát oproti blokům informací, které byly použity pouze 60 krát (viz Obrázek 14).




**Graf 28: Zapomínání bloků při učení v kontextovém režimu za použití relací. (simulace)**




**Obrázek 14: V náhledu modelu je zobrazen počet použití bloků v kontextovém režimu. (model)**

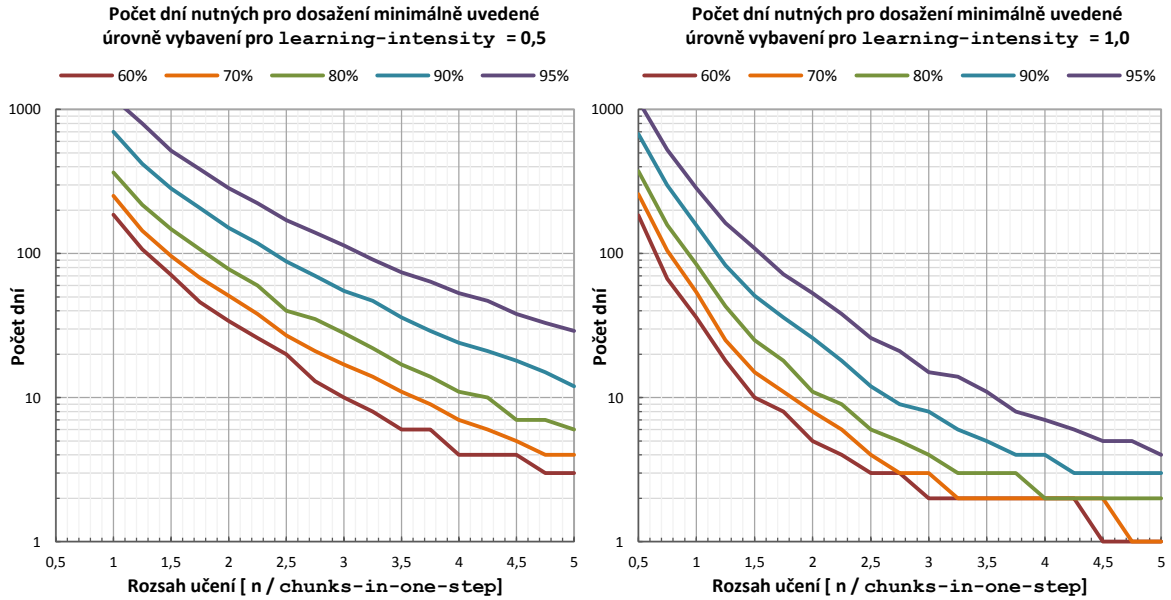
### 3.5.3 Experiment 3 - Vliv rozsahu a intenzity učení

Cílem tohoto experimentu bylo zjistit, jaký vliv mají rozsah a intenzita učení na čas potřebný pro dosažení jisté úrovně spolehlivosti vybavení informace pamětí. Byl použitý scénář `Numbers 0 to 9 next/prev` v `scenario` **{S2}**, kde je vygenerováno 20 bloků, které jsou v kontextu s jinými bloky prostřednictvím sdílených zdrojů aktivace. Experiment je zastaven v okamžiku, kdy jsou všechny bloky naučeny minimálně na zadanou pravděpodobnost vybavení 60, 70, 80, 90 nebo 95 %. Nejhůře naučený blok tedy bude mít uvedenou pravděpodobnost vybavení a ostatní bloky na tom budou s pravděpodobností vybavení lépe.

Byl měřen počet dní, které jsou potřeba pro dosažení uvedeného cíle. Předpokladem je, že každý den ve stejný čas proběhne lekce nebo sezení, během

kterého je naučeno `chunks-in-one-step` {L4} bloků. Získaná úroveň aktivace vzhledem ke kalibračním hodnotám je nastavena v `learning-intensity` {L3}. Parametr {L4} vyjadřuje rozsah učení (například kolik příkladů je spočítáno) a parametr {L3} umožní vyjádřit relativní zisk, jenž se může měnit s koncentrací, kterou věnujeme dané úloze. Použitá strategie `Bellow Retrieve Limit` nejdříve opakuje bloky, kterým klesla pravděpodobnost vybavení pod zadanou úroveň a pokud mají všechny bloky pravděpodobnost vybavení nad zadanou úrovní je naučen nový blok. Všechna ostatní nastavení jsou stejná jako v předchozích experimentech.

Pro snadnější orientaci ve výsledcích je na vodorovné ose grafů uveden poměr mezi počtem bloků v úloze vzhledem k rozsahu učení nastaveném v `chunks-in-one-step`. V případě hodnoty 1 je každý den učeno či opakováno tolik bloků, kolik je bloků v úloze, což znamená pro tento scénář 20 bloků. Pro hodnotu 5 je každý den učen či opakován pětinásobek rozsahu úlohy, tedy 100 bloků. Na svislé ose je potom počet dní, jak dlouho trvá dosažení požadované úrovně při zadaném rozsahu učení.

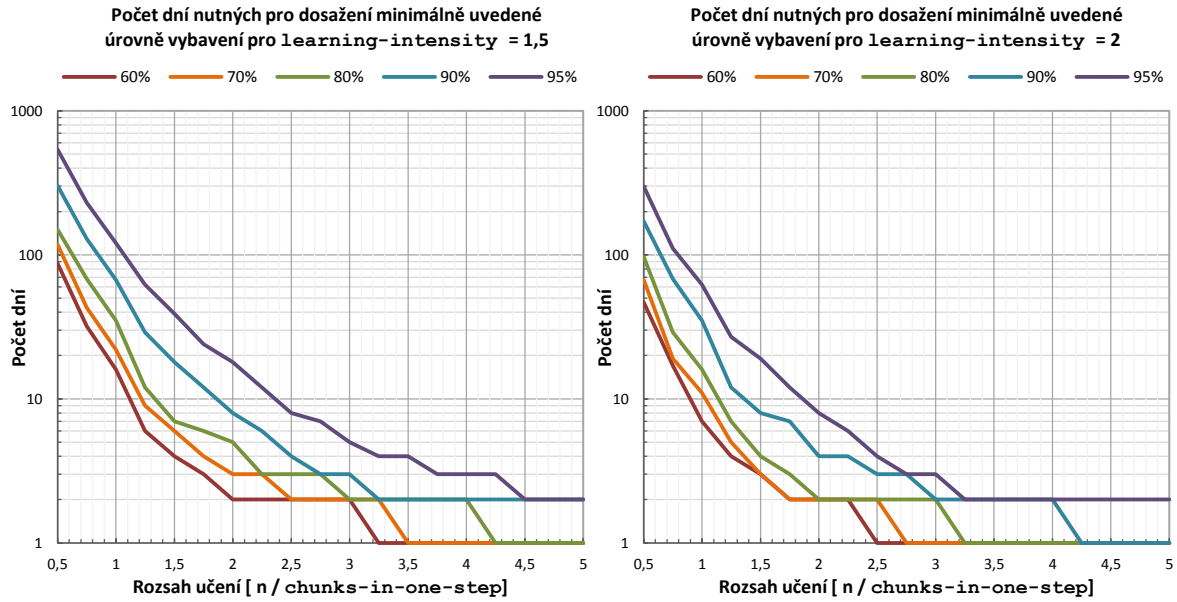



**Graf 29: Náročnost dosažení minimální úrovně vybavení bloku pro intenzitu učení 0,5 a 1. (simulace)**

Každý ze čtyř uvedených grafů vyjadřuje různou intenzitu učení nastavenou v `learning-intensity`. Hodnota 0,5 znamená, že je realizován pouze poloviční zisk aktivace z každé úlohy oproti kalibračním hodnotám v modelu. Toto nastavení simuluje sníženou pozornost, kterou učící se agent věnuje blokům, které jsou učeny nebo opakovány.

Hodnota 2 znamená, že je realizován dvojnásobný zisk aktivace oproti kalibračním hodnotě, což vyjadřuje vysokou schopnost koncentrace agenta při procesu učení nebo opakování.

Ze zjištěných výsledků je patrné, že jak rozsah učení, tak intenzita má velký vliv na celkový počet dní nutných pro naučení daného množství bloků na požadovanou úroveň pro vybavení. Pro přehlednost grafu je na svislé ose použito logaritmické měřítko. Rozdíl v počtu dní mezi úrovní 60 % a 90 % je přibližně čtyřnásobný. Mezi úrovní 60 % a 95 % je potom více než desetinásobný. Lze si tak vytvořit jistou představu, jak dlouho bude trvat dosažení minimální požadované úrovně pro pravděpodobnost vybavení bloku.




**Graf 30: Náročnost dosažení minimální úrovně vybavení bloku pro intenzitu učení 1,5 a 2. (simulace)**

Z grafů je také patrné, že i malé zvětšení rozsahu učení v daný den má dramatický vliv na pokles času nutného pro dosažení požadované úrovně vybavení bloku. Zdvojnásobení rozsahu učení má za následek přibližně pětinásobné zkrácení doby učení. A ztrojnásobení rozsahu učení tuto dobu zkrátí téměř o dvacetinásobek. Větší rozsah praxe tedy přináší podstatně rychlejší dosažení požadované úrovně pravděpodobnosti vybavení bloku. Proti tomuto efektu však působí fakt, že s větším rozsahem učení zpravidla klesá pozornost a tím i zisk aktivace, který tímto učením nebo opakováním získáváme. Model však zatím tento vztah nedokáže zohlednit. Při interpretaci výsledků je tedy nutné vyjít z toho, že oba efekty pracují zároveň a jejich vliv je protichůdný.

Větší praxe tedy může přinést rychleji požadovanou úroveň vybavení bloku, ale celková efektivita učení bude podstatně ovlivněna tím, jak dobře jsme schopni udržet pozornost. Díky vyšší úrovni pozornosti dosahujeme požadovaného zisku

aktivace, a tedy přínosu daného učení nebo opakování. Pokud nejsme schopni udržet pozornost, více času stráveného učením nemusí automaticky znamenat podstatně lepší výsledky a rychlejší dosažení cíle.

Důležitá je tedy schopnost koncentrace po delší časový úsek, která umožní realizovat dlouhodobě větší zisk aktivace, a tím zkrátit celkovou dobu potřebnou pro učení a zvýšit jeho efektivitu. Při kombinaci většího rozsahu učení a opakování za předpokladu udržení jeho efektivity lze dosáhnout překvapivě krátké doby nutné pro dosažení požadované úrovně.

### 3.5.4 Experiment 4 - Srovnání strategií učení

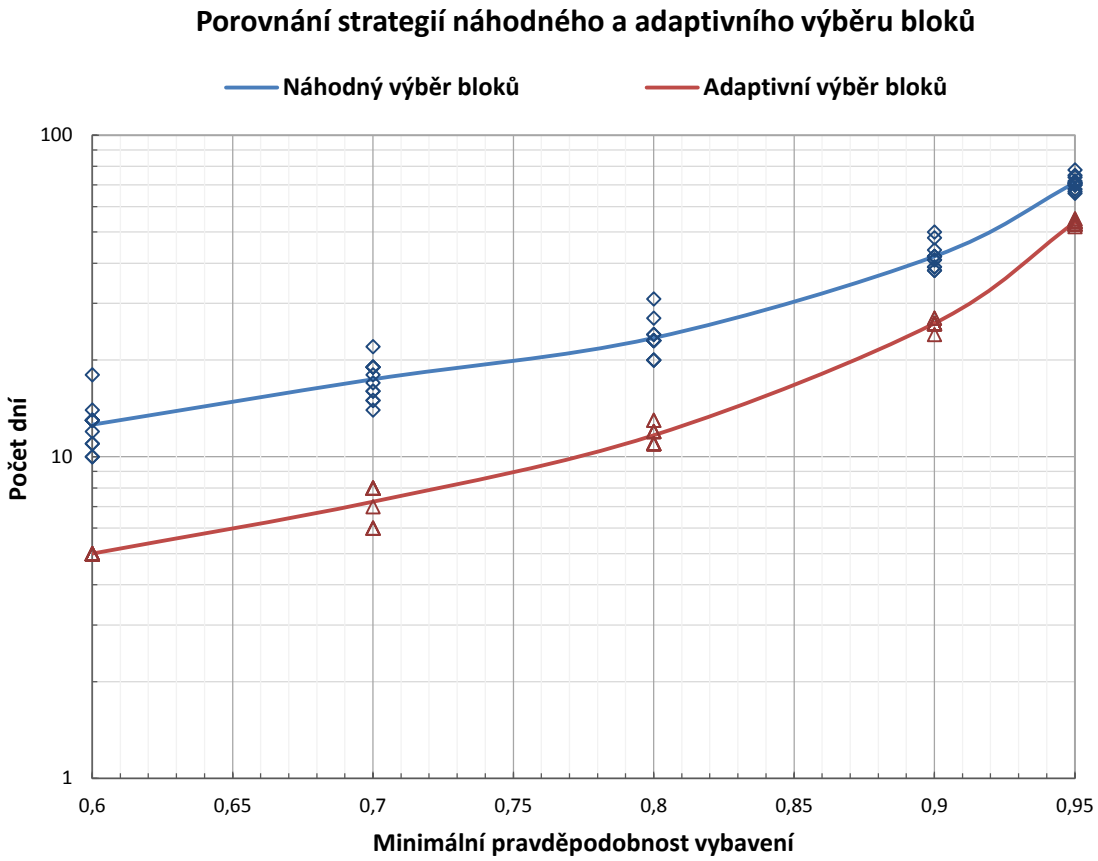
Cílem posledního experimentu bylo porovnat, jak velký je rozdíl mezi strategií náhodného výběru bloků `Random` a strategií adaptivního výběru bloků `Bellow Retrieve Limit`, která nejdříve opakuje bloky, které jsou pod limitem, a následně poté přidává nové bloky. Nastavení modelu bylo shodné jako u předchozích experimentů.

Z důvodu omezení rozsahu experimentu byla vybrána hodnota intenzity učení `learning-intensity` **{L3}** odpovídající kalibrační hodnotě 1. Zvolený počet opakování bloků za jeden den potom jako dvojnásobek celkového počtu učených bloků, což znamená nastavení `chunks-in-one-step` **{L4}** na hodnotu 40. Pro zvýšení objektivity měření bylo pro každou ze strategií provedeno 12 jednotlivých pokusů a měřen byl počet dní, kolik je potřeba pro dosažení minimální zvolené pravděpodobnosti pro vybavení bloku. Naměřené hodnoty jsou prezentovány v následující tabulce.

| Náhodný výběr bloků | | | | | |
|---|---|---|---|---|---|
| **min(P(i))** | **60%** | **70%** | **80%** | **90%** | **95%** |
| pokus | t[d] | t[d] | t[d] | t[d] | t[d] |
| 1 | 13 | 19 | 20 | 50 | 71 |
| 2 | 10 | 15 | 23 | 42 | 72 |
| 3 | 13 | 16 | 20 | 48 | 70 |
| 4 | 12 | 22 | 23 | 41 | 78 |
| 5 | 10 | 19 | 20 | 44 | 74 |
| 6 | 11 | 19 | 24 | 39 | 75 |
| 7 | 13 | 15 | 23 | 38 | 66 |
| 8 | 14 | 14 | 24 | 38 | 67 |
| 9 | 13 | 18 | 23 | 42 | 71 |
| 10 | 11 | 19 | 31 | 41 | 68 |
| 11 | 13 | 17 | 23 | 42 | 70 |
| 12 | 18 | 16 | 27 | 39 | 71 |
| **průměr** | **12,58** | **17,42** | **23,42** | **42,00** | **71,08** |
| porovnání | 40 % | 42 % | 50 % | 62 % | 75 % |

| Adaptivní výběr bloků | | | | | |
|---|---|---|---|---|---|
| **min(P(i))** | **60%** | **70%** | **80%** | **90%** | **95%** |
| pokus | t[d] | t[d] | t[d] | t[d] | t[d] |
| 1 | 5 | 8 | 11 | 26 | 52 |
| 2 | 5 | 8 | 12 | 26 | 54 |
| 3 | 5 | 6 | 11 | 26 | 55 |
| 4 | 5 | 8 | 11 | 26 | 54 |
| 5 | 5 | 7 | 12 | 26 | 53 |
| 6 | 5 | 8 | 13 | 26 | 54 |
| 7 | 5 | 8 | 12 | 26 | 53 |
| 8 | 5 | 8 | 12 | 26 | 53 |
| 9 | 5 | 6 | 11 | 24 | 54 |
| 10 | 5 | 6 | 13 | 27 | 55 |
| 11 | 5 | 8 | 11 | 27 | 53 |
| 12 | 5 | 6 | 11 | 26 | 53 |
| **průměr** | **5,00** | **7,25** | **11,67** | **26,00** | **53,58** |
| porovnání | 252 % | 240 % | 201 % | 162 % | 133 % |

**Tabulka 17: Porovnání strategií náhodného a adaptivního výběru bloků. (simulace)**




**Graf 31: Porovnání strategií výběru bloku pro parametr learning-intensity=1 a chunks-in-one-step = 40 pro 20 bloků ve scénáři Numbers 0 to 9 next/prev. (simulace)**

Ze srovnání strategií výběru bloku je jasně zřetelná výhoda strategie výběru bloku, která se přizpůsobuje aktuální úrovni jejich aktivace, a tím i pravděpodobnosti jejich vybavení. Bloky jsou tak nejdříve systematicky opakovány a následně, když je jejich pravděpodobnost nad stanovenou úrovní, jsou přidávány nové nenaučené bloky.

V náhodné strategii jsou bloky vybírány bez jakéhokoliv klíče, což vede k nadbytečnému opakování jistých bloků a nedostatečnému opakování jiných. Tato strategie má potom za následek snížení efektivity učení a prodloužení času nutného pro dosažení stanovaného minimálního limitu pro vybavení. Pro 60 % limit dosahuje náhodná strategie výběru pouze 40% efektivity oproti strategii adaptivního výběru, která je tak 2,5 krát rychlejší oproti této strategii. Rozdíl ve strategiích se snižuje se zvyšujícími se nároky na minimální limit pravděpodobnosti vybavení bloku, avšak zůstává stále velmi podstatný.

# 4 Dosažené výsledky

Při porovnání výstupů navrženého a implementovaného simulačního modelu ACT-R v NetLogu (podle nastavení viz Tabulka 16) s aktuálním analytickým modelem ACT-R[15] a regresními modely (viz Graf 12. s. 96) sestrojenými podle naměřených dat byly zjištěny následující výsledky (viz Tabulka 18).

| Model | Celková korelace s daty měření | Chyba modelu vzhledem k chybě regresního modelu maximální, průměrná a medián chyby pro dny 1 až 10 (viz s. 97) | | |
|---|---|---|---|---|
| | | Fan 1-1 | Fan 3-1 | Fan 3-3 |
| **Měřená data** | 1,0000 | - | - | - |
| **Regesní** | (-0,95%) 0,9905 | 0% | 0% | 0% |
| **Analytický ACT-R** | (-1,36%) 0,9864 | +71% \| +11% \| -7,0% | +91% \| +54% \| +23% | +98% \| +88% \| +67% |
| **Simulační ACT-R** | (-1,15%) 0,9885 | +35% \| -2,7% \| -5,7% | +39% \| +2,6% \| -4,5% | +15% \| -16% \| -9,2% |

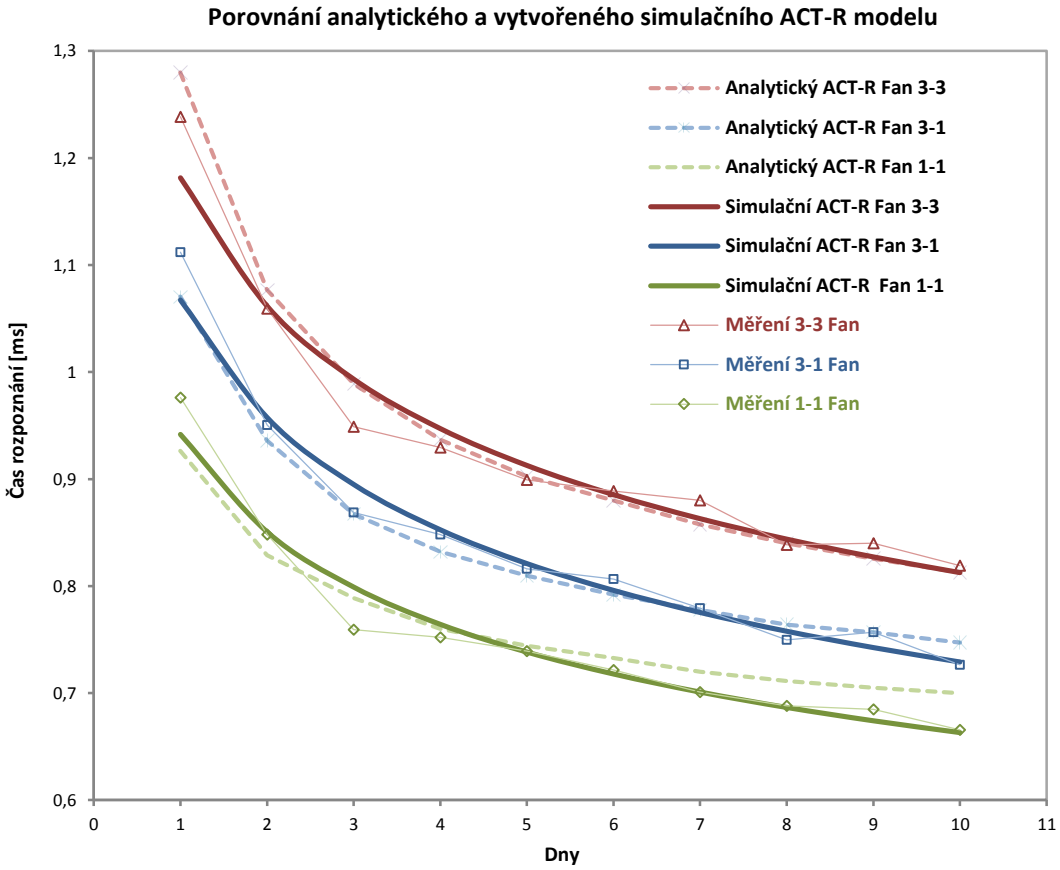
**Tabulka 18: Porovnání relativní velikosti chyb analytického a simulačního modelu. (vlastní výpočty)**

Z porovnání je patrné, že navržený simulační model kalibrovaný podle uvedených dat viz Tabulka 16 má v některých případech téměř o řád menší absolutní chybu v porovnání s chybou analytického modelu (viz s. 97). Celkový koeficient korelace 0,9885 je také nepatrně vyšší než u analytického modelu (0,9864) a přibližuje se modelu regresnímu (0,9905).

Pro porovnání jsou uvedeny odchylky od chyb regresního modelu v procentech. Kladné číslo vyjadřuje větší chybu a záporné číslo menší chybu než uvedený regresní model (metodika viz Tabulka 14). Pro celkový přehled o chybách modelů je uvedena odchylka maximální, průměrná i mediánová ze souboru odchylek pro všechny dny. Simulační model se dokonce jeví jako přesnější prediktor než regresní model s mocninnou funkcí, pokud uvážíme průměrnou chybu a chybu, která je mediánem. Maximální odchylka v porovnání s analytickým modelem je podstatně menší. S více daty a přesnější kalibrací je zde potenciál dosáhnout lepších výsledků a případně zajistit celkovou validitu modelu pro různý typ fan bloků.

Agentový a simulační přístup umožnil vytvořit model, jenž popisuje chování na úrovni jednotlivých bloků a relací.[33] Umožňuje tím parametrizovat zisk a ztrátu aktivace bloku i relace v závislosti na typu bloku dle jeho fan čísla. Bylo zjištěno, že pro úplné parametrizování modelu je nutné u zisku a poklesu aktivací relací uvažovat jak s velikostí fan čísla bloku, tak případnou nesymetrií ve fan číslech. Nedostatek

dalších dat z reálných experimentů však zatím neumožňuje potvrdit či vyvrátit celkovou obecnou validitu navrženého modelu i pro bloky jiných typů fan čísel.




**Graf 32: Srovnání analytického a simulačního ACT-R modelu pro měřená data. (vlastní zpracování)**

Při vytváření simulačního modelu byl objeven vztah, kterým je korigována velikost zisku a degradace aktivace relace. Korekce se provádí na základě velikosti fan čísla zdroje aktivace pro relaci, a také na základě asymetrie ve fan číslech pro blok, ke kterému relace směřuje (viz Ukázka kódu 24 a Ukázka kódu 25, s. 72).

$$diff_{ji} = fan_j - \sum_{k=i_{sources}, k \neq j} fan_k$$

**Rovnice 29: Výpočet asymetrie ve fan číslech pro relaci ze zdroje aktivace $j$ pro blok $i$ (viz Ukázka kódu 23, s. 72). (vlastní návrh)**

Absolutní velikost zisku nebo degradace aktivace lze určit podle následujícího vztahu.

$$x_{ji} = c \cdot {c_f}^{(fan_j - 1)} \cdot {c_d}^{(diff_{ji})}$$

**Rovnice 30: Výpočet korekce zisku nebo ztráty aktivace $x_{ji}$ relace. (vlastní návrh)**

V uvedeném vztahu je $c$ základní koeficient zisku nebo ztráty aktivace relace, $c_f$ je koeficient vlivu fan čísla zdroje aktivace na tento zisk nebo ztrátu a $diff_{ji}$ je velikost asymetrie mezi fan číslem bloku $j$ a celkovým fan číslem všech zdrojů aktivace bloku $i$ mimo blok $j$.

# 5 Závěr

Cílem práce bylo navrhnout, implementovat a provést experimenty se simulačním modelem učícího se systému v prostředí NetLogo. Jako učící se systém byl zvolen model lidského agenta a jeho deklarativní paměť uchovávající informace jejich vzájemné vazby. Teoretickým podkladem pro tvorbu modelu byla vybrána kognitivní teorie ACT-R a její modul deklarativní paměti (viz s. 10), která používá model, jehož parametrizováním se snaží popsat a předpovědět výstupy měřené v experimentech (viz s. 12). Některé z komponent tohoto modelu vycházejí ze vztahů použitelných pro simulační účely (viz s. 20). Pro popis relací se potom používá převážně analytický přístup, který se snaží parametrizovat statistický model tak, aby jeho predikce co nejvíce odpovídala měřeným datům. Standartní ACT-R model se tedy nesnaží popsat využívání relací ani procesy, které ovlivňují jejich tvorbu a stav.

Navržený simulační model proto pro aktivaci relací používá stejnou metodu jako standardní ACT-R teorie používá pro základní aktivaci bloků (viz s. 24). Díky tomuto přístupu je umožněno simulovat jak vývoj stavu bloků nesoucí informace, tak i vývoj stavu relací, které tyto informace propojují. Před implementací modelu se jednalo pouze o hypotézu, že lze tímto způsobem aktivaci relací modelovat.

Po analýze teorie ACT-R byl proveden návrh simulačního modelu (viz s. 40), který byl následně rozšířen o parametry, které umožňují model a jeho části nastavit a kalibrovat. Byly odvozeny také nové vztahy, které ACT-R nepopisuje (viz s. 45, s. 47). Ty se ukázaly jako klíčové pro popis procesů, které v modelu probíhají a mají podstatný vliv na pozorované výstupy a jejich shodu s daty z reálných experimentů.

Pomocí vývojového cyklu (viz s. 40) a postupně se rozšiřujícího návrhu byly části modelu a jeho jednotlivé verze implementovány, validovány a testovány. Během tohoto procesu byly objeveny další vztahy popisující fungování modelu na lokální úrovni jednotlivých bloků a relací. Z tohoto pohledu lze vytvořený model považovat za agentově orientovaný.[32] Celkem proběhlo přibližně 120 iterací tohoto cyklu.

U pozdějších verzí modelu probíhaly pokusy model kalibrovat tak, aby jeho výstupy byly ve shodě s daty získanými z reálných experimentů. Tento cíl se podařilo dostatečně splnit a v dalších verzích byl model ještě rozšířen o parametry umožňující jej přesněji nastavit. Základní nastavení jádra modelu podílející se přímo na stavu aktivací bloků i relací má nyní sedm parametrů (viz s. 102), které řídí zpětné vazby procesů zisku a ztráty těchto aktivací. To má vliv na pozorované výstupy modelu při

simulaci. Konečná verze modelu byla optimalizována pro co nejrychlejší běh a následně strukturována do funkcí, procedur a detailně popsána (viz s. 58). Pro usnadnění orientace v rozhraní modelu byl vytvořen jeho komplexní přehled (viz s. 83) a uveden význam jednotlivých komponent (viz s. 84).

Formální správnost modelu byla zajištěna důkladnou validací funkcí základních vztahů v různých testovacích scénářích (viz s. 89). Model byl také validován z pohledu numerických odchylek, které mohou při výpočtu vznikat. Zjištěná odchylka simulačního modelu od analytického vztahu se pohybuje v řádech $10^{-6}$ až $10^{-10}$ a pro praktické účely je tedy naprosto zanedbatelná (viz s. 92).

Kalibrace modelu nastaví model tak, aby jeho výstupy byly ve shodě s daty získanými v experimentech. Pro tento krok bylo nutné připravit kalibrační sadu (viz s. 95), jež představuje hodnoty, podle kterých je model kalibrován. V rámci kalibrace potom proběhl také výpočet potřebných parametrů časového a aktivačního měřítka (viz s. 98), které určují převod aktivace na čas potřebný pro vybavení informace měřený v experimentech. Byl také popsán vliv a význam parametrů modelu na pozorované výsledky (viz s. 102) a detailní postup jak model kalibrovat (viz s. 104).

Implementovaný a kalibrovaný simulační model má v některých případech až o jeden řád (10krát) menší absolutní chybu, menší než aktuální analytický model používaný v teorii ACT-R. V průměru je tento model srovnatelný s regresním mocninným modelem pro naměřená data a dokonce jej mírně překonává menší průměrnou absolutní chybou. Díky simulačnímu přístupu byly také objeveny nové vztahy popisující zisk i degradaci aktivace relací podle velikosti fan čísla (viz s. 117). Omezené množství dostupných experimentálních dat zatím neumožňuje potvrdit nebo vyvrátit celkovou obecnou validitu uvedeného simulačního modelu.

V experimentech prováděných s modelem bylo zjištěno, že opakované učení stejné informace má za následek její dlouhodobější udržení v paměti (viz s. 109). Při učení nových informací v kontextu jsou tyto informace učeny pomaleji, ale o to déle jsou udrženy v paměti oproti informacím, které jsou učeny jako izolovaná fakta (viz s. 110). Dále byl prokázán velký vliv rozsahu učení a předněji potom soustředění na čas potřebný pro dosažení požadované úrovně vybavení informací (viz s. 112). Na závěr bylo provedeno srovnání dvou strategií výběru bloků při učení i opakování. Náhodný výběr dosahoval podstatně horšího výsledku než strategie zaměřená na systematické opakování bloků a postupné rozšiřování o nové bloky (viz s. 115).

Známým nedostatkem modelu je neschopnost zachytit pokles zisku aktivace v důsledku intenzivního opakování stejného bloku. Rychlost degradace aktivace daného užití bloku by měla být ovlivněna tím, kdy naposled došlo k jeho užití. Je tedy rozdíl, zda je provedeno 10 užití během 10 minut nebo stejný počet užití během 10 sekund. Toto by se mělo projevit na počáteční úrovni aktivace daného užití případně na rychlosti degradace této aktivace. Uvedená vlastnost nebyla do modelu zahrnuta z důvodu absence experimentálních dat, podle kterých by pak bylo možné provést kalibraci parametrů.

Implementovaný model také neobsahuje aktivní vybavování bloků a slouží nyní pouze pro předpovědní účely úrovně aktivace a tím času a pravděpodobnosti vybavení bloku informace. Dalším krokem by bylo jej doplnit o procedurální modul, tak jak jej definuje teorie ACT-R, obsahující pravidla (*productions*) pro výběr bloků. Princip výběru pravidel by potom mohl být implementován podobně jako u bloků a relací s využitím aktivačních úrovní.

Šum je v modelu zatím reprezentován výpočtem podle analytického vztahu. Pro věrnější výsledky simulace by jej bylo možné generovat jako vhodně parametrizovanou náhodnou procházku (*random walk*). Případně se zdá být nyní možné navrhnout model kompletně s využitím popsaných parametrů na principu náhodné procházky a popsat jej tím tak velmi nízko úrovňově. Pozorované jevy na makroúrovni by byly potom emergentními jevy procesů na mikroúrovni. Tento přístup však nebyl zvolen, protože by byl příliš velkým krokem do neznáma s nejistým výsledkem. S většími získanými znalostmi se však jeví jako jeden z možných směrů dalšího zkoumání.

Model by bylo možné rozšířit o možnost vytváření zcela nových relací mezi bloky, se kterými současná teorie ACT-R příliš nezabývá a simulovat jejich dynamický vývoj v čase. Zajímavých výsledků by mohlo být dosaženo, pokud by byl model parametrizován daty v podobě textu a následně by jej bylo možné použít jako kontextový generátor textových řetězců. Další zajímavou aplikací by mohlo být řízení chování inteligentního učícího se agenta podle principu aktivací, na kterém model funguje.

# 6 Seznam literatury


[1] DOMINGOS, Pedro. Master algorithm: how the quest for the ultimate learning machine will remake our world. New York: Basic Books, 2015. ISBN 978-0-465-06570-7.

[2] ANDERSON, John R. *How can the human mind occur in the physical universe?* 1. New York: Oxford University Press, 2007. ISBN 978-0-19-532425-9.

[3] BACH, Joscha. *Principles of synthetic intelligence: PSI: an architecture of motivated cognition*. New York: Oxford University Press, 2009. Oxford series on cognitive models and architectures. ISBN 978-0-19-537067-6.

[4] BUZAN, Tony, ABBOTT, Susanna (ed.). *The Ultimate Book of Mind Maps®: Unlock Your Creativity, Boost Your Memory, Change Your Life*. FL, USA: Buzan Centres Ltd, 2012. ISBN 978-0-007-49956-4.

[5] WILENSKY, U. (1999). *NetLogo.* http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[6] RAILSBACK, Steven F. a Volker GRIMM. *Agent-based and individual-based modeling: a practical introduction*. Princeton: Princeton University Press, c2012. ISBN 978-0-691-13674-5.

[7] SKALSKÁ, Hana. *Stochastické modely a modelování*. 1. Hradec Králové: GAUDEAMUS, 2006. ISBN 80-7041-488-X.

[8] WILENSKY, U. (1999). *The NetLogo 5.3.1 User Manual* [online]. Northwestern University, Evanston, IL (USA): Center for Connected Learning and Computer-Based Modeling, 2016 [cit. 2016-05-18]. Dostupné z: http://ccl.northwestern.edu/netlogo/docs/NetLogo%20User%20Manual.pdf

[9] ROY, Peter Van. *Programming Paradigms for Dummies: What Every Programmer Should Know* [online]. 2009, s. 39 [cit. 2016-03-15]. Dostupné z: https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf

[10] Northestern University: Profile of Dr. Uri Wilensky. *Northestern University* [online]. Evanston, IL (USA): Northwestern University, 2016 [cit. 2016-06-10]. Dostupné z: http://www.sesp.northwestern.edu/profile/?p=89

[11] The Center for Connected Learning and Computer-Based Modeling: About. *Northwestern University: The Center for Connected Learning and Computer-Based Modeling* [online]. Evanston, IL (USA): Northwestern University, 2016 [cit. 2016-06-10]. Dostupné z: http://ccl.northwestern.edu/

[12] SONDAHL, F., S. TISUE a U. WILENSKY. *BREEDING FASTER TURTLES: PROGRESS TOWARDS A NETLOGO COMPILER* [online]. Evanston, IL (USA), 2004 [cit. 2012]. Dostupné z: http://ccl.northwestern.edu/papers/sond_tis_wil_breeding.pdf . článek

[13] WILENSKY, U. (1999). *The NetLogo 5.3.1 User Manual* [online]. Northwestern University, Evanston, IL (USA): Center for Connected Learning and Computer-Based Modeling, 2016 [cit. 2016-05-18]. Dostupné z: http://ccl.northwestern.edu/netlogo/docs/NetLogo%20User%20Manual.pdf

[14] Robotomie.cz: NetLogo. *Robotomie.cz* [online]. Praha: projekt MFF UK, Akademie věd ČR a Generation Europe, 2010 [cit. 2016-07-02]. Dostupné z: http://www.robotomie.cz/netlogo.php

[15] ANDERSON, John R., Michael D. BRYNE, Scott DOUGLASS, Christian LEBIERE a Yulin QIN. *An Integrated Theory of the Mind* [online]. Pittsburgh, PA (USA), 2004 [cit. 2016-07-20]. Dostupné z: http://act-r.psy.cmu.edu /wordpress/wp-content/uploads/2012/12/526FSQUERY.pdf.
Carnegie Mellon University.

[16] LEBIERE, Christian. *The dynamics of cognition: An ACT-R model of cognitive arithmetic* [online]. Pittsburgh, PA (USA), 1999 [cit. 2016-05-24]. Dostupné z: http://act-r.psy.cmu.edu/wordpress/wp-content/uploads/2012/12/459459 .pdf. Psychology Department, Carnegie Mellon University.

[17] ANDERSON, John R. a Christian D. SCHUNN. *Implications of the ACT-R Learning Theory: No Magic Bullets*. Pittsburgh, PA (USA), 2000. Carnegie Mellon University.

[18] FRANKLIN, Gene F., J. David POWELL a Abbas. EMAMI-NAEINI. *Feedback control of dynamic systems*. 6th ed. Upper Saddle River [N.J.]: Pearson, c2010. ISBN 978-0-13-601969-5.

[19] SHOHAM, Yoav a Kevin LEYTON-BROWN. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. New York: Cambridge University Press, 2009. ISBN 0521899435.

[20] NEWELL, Allen a Paul S. ROSENBLOOM. Mechanisms of skill acquisition and the law of practice. *Research Showcase @ CMU* [online]. Carnegie Mellon University, 1982, 60 [cit. 2016-07-29]. Dostupné z: http://repository.cmu.edu/cgi/viewcontent.cgi?article=2615&context=compsci

[21] PIROLLI, Peter L. a John R. ANDERSON. The Role of Practice in Fact Retrieval. *Journal of Experimental Psychology: Learning, Memory and Cognition* [online]. 1985, **11**(1), 136-153 [cit. 2016-06-17]. Dostupné z: https://www.gwern.net/docs/spacedrepetition/1985-pirolli.pdf

[22] MACKAY, David J. C. *Information theory, inference, and learning algorithms*. 4. Cambridge: Cambridge University Press, 2003. ISBN 978-0-521-64298-9.

[23] MEIER, P. S. *Mind-mapping: a tool for eliciting and representing knowledge held by diverse informants* [online]. UK, 2007 [cit. 2016-03-21]. Dostupné z: http://sru.soc.surrey.ac.uk/SRU52.pdf. Department of Sociology, University of Surrey.

[24] EPPLER, Martin J. *A comparison between concept maps, mind maps, conceptual diagrams, and visual metaphors as complementary tools for knowledge construction and sharing* [online]. Lugano, Switzerland, 2006 [cit. 2016-07-20]. Dostupné z: http://www.liquidbriefing.com/twiki/pub/Dev/RefEppler2006/comparison_between_concept_maps_and_other_visualizations.pdf. Faculty of Communication Sciences, University of Lugano (USI).

[25] *Duolingo* [online]. 2016 [cit. 2016-07-21]. Dostupné z: https://www.duolingo.com/

[26] STREETER, Matthew. Mixture Modeling of Individual Learning Curves. In: *Duolingo* [online]. Pittsburgh, PA: Duolingo, Inc., 2015 [cit. 2016-07-07]. Dostupné z: https://s3.amazonaws.com/duolingo-papers/publications/streeter.edm15.pdf

[27] VESSELINOV, Roumen a John GREGO. Duolingo Effectiveness Study: FINAL REPORT. In: *Http://static.duolingo.com/* [online]. 2012 [cit. 2016-07-07]. Dostupné z: http://static.duolingo.com/s3/DuolingoReport_Final.pdf

[28] *KhanAcademy: Math* [online]. 2016 [cit. 2016-07-07]. Dostupné z: https://www.khanacademy.org/math

[29] LINK, Daniela a Julian N. MAREWSKI. *Populating ACT-R's Declarative Memory with Internet Statistics* [online]. Switzerland, 2015 [cit. 2016-06-07]. Dostupné z: http://www.iccm2015.org/proceedings/papers/0015/paper0015.pdf. UNIL-Internef.

[30] LEIBOWITZ, Nathaniel, Barak BAUM, Giora ENDEN a Amir KARNIEL. The exponential learning equation as a function of successful trials results in sigmoid performance. *Journal of Mathematical Psychology* [online]. Ben-Gurion University of the Negev, Beer-Sheva, Israel, 2009, **2010**(54), 3 [cit. 2016-03-05]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0022249610000179

[31] KASAIE, Parastu a W. David KELTON. GUIDELINES FOR DESIGN AND ANALYSIS IN AGENT-BASED SIMULATION STUDIES. In: *Proceedings of the 2015 Winter Simulation Conference* [online]. 2015, s. 11 [cit. 2016-06-18]. Dostupné z: http://dl.acm.org/citation.cfm?id=2888638

[32] MACAL, Charles M. a Michael J. NORTH. TUTORIAL ON AGENT-BASED MODELING AND SIMULATION PART 2: HOW TO MODEL WITH AGENTS. *Proceedings of the 2006 Winter Simulation Conference* [online]. Center for Complex Adaptive Agent Systems Simulation: Argonne National Laboratory, 2016, 11 [cit. 2016-07-20]. Dostupné z: http://www2.hawaii.edu/~nreed/ics606/papers/Macal06model.pdf

[33] C. M. SIEBERS, J. GARET, D. BUXTON a M. PIDD. Discrete-Event Simulation is Dead, Long Live Agent-Based Simulation!. *Journal of Simulation* [online]. 2010, **4**(3), 204-210 [cit. 2016-06-03]. Dostupné z: http://www.cs.nott.ac.uk/~pszps/docs/pos-JOS-2010-2.pdf

# 7 Přílohy

## 7.1 Seznam obrázků



## 7.2 Seznam tabulek

## 7.3 Seznam grafů

## 7.4 Seznam rovnic



## 7.5 Seznam ukázek kódu

## 7.1 Datový nosič CD

Přiložené CD obsahuje následující soubory:

- Elektronický text práce
  - **diplomova-prace.pdf** verze souboru pro *Acrobat Reader*
  - **diplomova-prace.docx** verze douboru pro *Microsoft Word*
- Model deklarativní paměti pro NetLogo
  - **declarative-memory1.nlogo** model pro spuštění v prostředí NetLoga
  - **declarative-memory2.nlogo** model pro spuštění v prostředí NetLoga
  - **model-verze.zip** složka obsahující pracovní verze modelu

Univerzita Hradec Králové
Faculty of Informatics and Management
Akademický rok: 2015/2016

Studijní program: Applied Informatics
Forma: Combined
Obor/komb.: Aplikovaná informatika (ai2-k)

**Podklad pro zadání DIPLOMOVÉ práce studenta**

| PŘEDKLÁDÁ: | ADRESA | OSOBNÍ ČÍSLO |
|---|---|---|
| Bc. Michna Michal | Pražská 411, Děčín - Děčín XXXII-Boletice nad Labem | I14297 |

**TÉMA ČESKY:**

Zpětná vazba v učícím se systému - model v NetLogu

**TÉMA ANGLICKY:**

Feedback in learning system - NetLogo model

**VEDOUCÍ PRÁCE:**

doc. RNDr. Kamila Štekerová, Ph.D. - KIT

**ZÁSADY PRO VYPRACOVÁNÍ:**

1. Úvod
 - Cíl
 - Metoda
2. Teoretická část
 - Vymezení pojmů (učící se systém, zpětná vazba, efektivita učení, myšlenková mapa)
 - Přehled dostupných aplikací (výuka jazyků, výuka matematiky)
 - Simulace jako metoda zkoumání učících se systémů
3. Praktická část
 - Návrh modelu učícího se systému
 - Implementace v NetLogu
 - Validace a kalibrace modelu s využitím reálných dat
 - Experimenty (měření efektivity výukového procesu pomocí modelu, srovnání strategií učení)
 - Vyhodnocení experimentů
4. Závěr

**SEZNAM DOPORUČENÉ LITERATURY:**

Mařík, Vladimír. Umělá inteligence. Praha, 2007. ISBN 978-80-200-1470-2
BUZAN, Tony a Barry BUZAN. Myšlenkové mapy: probuďte svou kreativitu, zlepšete svou paměť, změňte svůj život. 2. vyd. Brno: BizBooks, 2012, 210 s. ISBN 978-80-265-0030-8.
FRANKLIN, Gene F, J POWELL a Abbas EMAMI-NAEINI. Feedback control of dynamic systems. 6th ed. Upper Saddle River [N.J.]: Pearson, c2010, xviii, 819 p. ISBN 0136019692.
MACKAY, David J. Information theory, inference, and learning algorithms. New York: Cambridge University Press, 2003, xii, 628 p. ISBN 9780521642989.

Podpis studenta: ..... Datum: 8.8.2016

Podpis vedoucího práce: ..... Datum: 8.8.2016