Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Programovatelný celulární automat v Javě

Plzeň 2013 Tomáš Rojík

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2013

Tomáš Rojík

Abstract

The aim of this bachelor thesis is to create a programmable cellular automaton in Java programming language. The application should be easy to run using the Java Web Start technology. The main target is to suggest an appropriate interface for the design of the cellular automata. The other targets are to export a run of a cellular automaton into a sequence if images and to export the development of a cellular automaton to a video sequence. The thesis also includes an overview of the existing solutions for simulation of cellular automata.

Cílem této bakalářské práce je vytvoření programovatelného celulárního automatu v programovacím jazyce Java. Aplikace má být snadno spustitelná jako Java Web Start. Hlavním úkolem je navrhnout vhodné rozhraní pro návrh celulárních automatů. Mezi další cíle patří export celulárního automatu do obrázkového formátu a export vývoje celulárního automatu do videozáznamu. Práce dále obsahuje popis některých existujících řešení pro simulaci celuárních automatů.

Obsah

1 Úvod

Celulární automat je matematický model určený k simulaci fyzikálního systému, prostor i čas modelu je diskrétní. Nejvíce užívaný je v podobě dvojrozměrné mřížky, přičemž každá buňka může nabývat určité hodnoty. Při kroku simulace se počítá další generace. Pro výpočet se používá přechodová funkce – tato funkce pracuje pouze s hodnotou dané buňky a jejím okolí. Detailní popis viz kapitola 2.

Hlavním cílem této práce byl návrh a implementace programovatelného prostředí pro celulární automaty v programovacím jazyce Java tak, aby bylo možné tento software zprovoznit jako Java Web Start aplikaci. Při návrhu měl být kladen důraz zejména na možnost vytváření vlastních celulárních automatů a k tomuto účelu navrhnout vhodné generické třídy nebo rozhraní. Dalším cílem této práce bylo umožnění exportu vizualizace celulárního automatu do některého ze standardních obrázkových formátů (JPG, PNG, apod.) a ideálně i export videosekvence vývoje celulárního automatu. Mezi teoretické cíle práce patřilo seznámení se s problematikou celulárních automatů, jejich implementací a existujících řešení.

Kromě výše popsaných požadavků vyplývajících ze zadání práce bylo do aplikace doplněno i několik funkcí nad rámec zadání. Pro vyšší komfort při definici nového pravidla byl implementován editor pravidla, ve kterém je možno definovat přechodovou funkci, počet a barvy stavů, okrajové podmínky a počáteční populaci. Počáteční populaci v editoru je možné načíst ze souboru nebo vytvořit kreslením. Další funkčností nad rámec zadání je možnost načtení libovolného obrázku na pozadí automatu, což umožňuje sledování chování automatu vzhledem ke skutečnosti. Lze tak sledovat například lesní požár nad skutečnou kartografickou mapu.

Celá práce je členěna do několika kapitol. Kapitola 2 obsahuje teoretický základ problematiky celulárních automatů. Kapitola 3 je přehledem existujících řešení simulátorů celulárních automatů s popisem vlastností relevantních k této práci. Kapitola 4 obsahuje stručný popis použitých technologií. Kapitola 5 popisuje samotnou implementaci a části samotného simulátoru automatu. Kapitola 6 popisuje implementovaná vzorová pravidla.

2 Celulární automat

Celulární nebo také buněčný automat je matematický model fyzikálního systému, jehož prostor a čas jsou diskrétní. Skládá se z pravidelné mřížky buněk a pravidla pro vytvoření nové generace. Každá buňka může nabývat konečného počtu stavů. Počáteční stav všech buněk se volí přiřazením určitého stavu. Pravidlo pro určení nového stavu pracuje pouze se stavem buňky a stavy buněk v jejím okolí.

2.1 Historie celulárních automatů

Jako první se problematikou celulárních automatů začal zabývat John von Neumann ve 40. letech 20. století. Snažil se navrhnout stroj, který by se dokázal reprodukovat, tedy vytvořit svoji vlastní kopii bez nutnosti vztahu k biologickým procesům. Von Neumann spolu se Stanislawem Ulamem rozdělili celý prostor na jednotlivé buňky, přičemž každá buňka byla na začátku charakterizována počátečním stavem.

V 50. letech 20. století se celulární automaty začaly používat k automatickému zpracování obrazů. Byla vyvinuta speciální pravidla pro úpravu šumu, odhad velikosti a počtů objektů na obraze pořízeného mikroskopem.

V roce 1970 navázal na von Neumannovu práci britský matematik John Horton Conway, který našel přechodovou funkci platnou pro všechny buňky, a toto pravidlo nazval Game of Life (Hra života).

V 80. letech se především rozvíjely jednodimenzionální celulární automaty zásluhou Stephena Wolframa, který dokázal, že při aplikaci omezujících podmínek se celulární automat chová v souladu s rovnicemi o proudění nestlačitelné kapaliny, a je ho tedy možné využít k modelování fyzikálních problémů. [5, 13, 14]

2.2 Vlastnosti celulárního automatu

Celulární automaty mají tři základní vlastnosti:

- Paralelismus – výpočty ve všech buňkách probíhají zároveň,
- lokalita – nový stav závisí na aktuálním stavu buňky samotné a jejích sousedů,
- homogenita – všechny buňky používají stejné pravidlo k přechodu do další generace.

Za další vlastnosti celulárních automatů lze považovat okrajové podmínky, dimenze mřížky, počet stavů buňky a způsob určení okolí buňky. Tyto vlastnosti budou popsány níže.

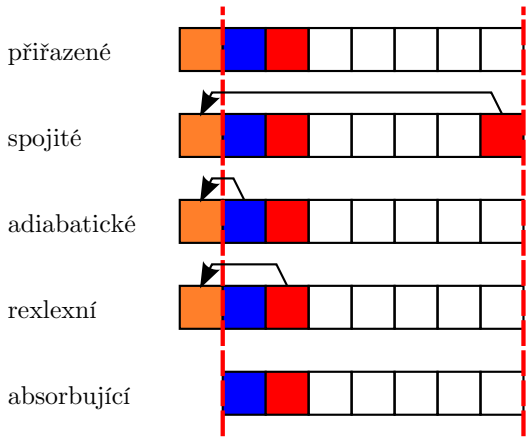
Dále se budeme zabývat jednodimenzionálními a dvojdimenzionálními automaty. Existují také třídimenzionální automaty sloužící k modelování prostorových závislostí. [5, 13]

2.2.1 Okrajové podmínky celulárních automatů

V obecném pojetí struktura vyplňuje celý nekonečný prostor, pro jehož omezení musíme předpokládat určité okrajové podmínky. Buňkám, které se nachází na hranici automatu, mohou chybět buňky nutné k vytvoření předepsaného okolí. Proto je potřeba stanovit stav buněk nacházejících se za hranicí automatu.

Možné okrajové podmínky(obrázek 2.1):

- Určené hodnotou – buňkám je nastavena určitá hodnota (například nulová),
- spojité – v případě jednodomenzionálních automatů se jedná o smyčku a případně u dvojdimenzionálních automatů anuloid (obrázek 2.2),
- adiabatické – buňce za hranicí automatu je přiřazen stav, který má hraniční buňka,
- reflexní – buňce za hranicí automatu je přiřazen stav, který má buňka sousedící z hraniční buňkou z druhé strany,
- absorbující – předpokládá se, že za hranicí automatu už nejsou žádné buňky.[2]



Obrázek 2.1: Okrajové podmínky celulárního automatu [2]

2.2.2 Formální definice celulárních automatů

Definice jednorozměrného celulárního automatu

Jednorozměrný celulární automat je jednoznačně určen sedmicí:

$$A = (Q, N, R, z, b_1, b_2, c_0), \tag{2.1}$$

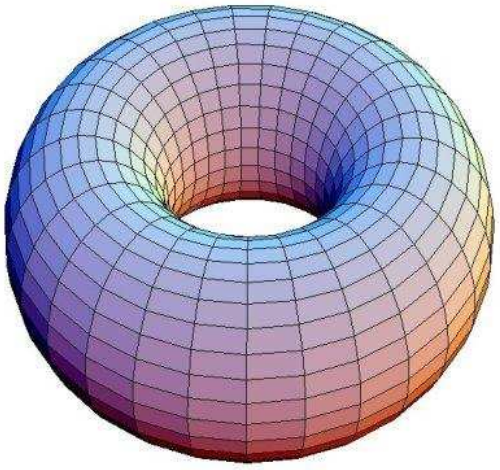
kde Q je množina stavů, N okolí buňky, z počet buněk, b_1 a b_2 hraniční hodnoty a c_0 počáteční konfigurace. Množina R je zobrazení množiny S do množiny δ, $R : S \to \delta$. Množina S obsahuje jednotlivé buňky v mřížce $S = 1, 2, ..., z$. Množina δ je množina lokálních přechodových funkcí $\delta_1, \delta_2, ..., \delta_z$. Takto popsané přechodové funkce jsou předpisem, které množině stavů Q přiřazují právě jeden stav z množiny $Q \cdot \delta_i = Q^N \to Q$. [13, 18]

Definice dvojrozměrného celulárního automatu

Dvojrozměrný celulární automat je jednoznačně určen šesticí:

$$(L, \Sigma, B, c_0, N, \varphi), \tag{2.2}$$

kde L představuje typ mřížky automatu (čtvercová, šestiúhelníková, trojúhelníková), Σ je množina stavů, B popisuje okrajovou podmínku, c_0 počáteční konfiguraci, N okolí buňky (Moorovo, Neumannovo) a ϕ je přechodová funkce, kterou lze formulovat jako:



Obrázek 2.2: Anuloid [31]

$$s_i(t+1) = \phi(s_j(t)), \tag{2.3}$$

kde stav s je aktualizován podle přechodové funkce. Přechodová funkce je *pravidlo celulárního automatu*. [2, 11]

2.2.3 Klasifikace automatů

Stephen Wolfram v roce 1980 definoval čtyři základní třídy, do nichž rozdělil celulární automaty v závislosti na jejich chování:

- Třída 1 – počáteční populace se vyvíjí velmi rychle a přechází do konečného stabilního stavu, tedy dále se nemění,
- třída 2 – počáteční populace se rychle vyvíjí do stabilních oscilujících struktur, tzv. oscilátorů,
- třída 3 – počáteční populace se vyvíjí pseudonáhodným nebo chaotickým způsobem, jakékoliv stabilní struktury jsou rychle zničeny chaotickým chováním okolních buněk,
- třída 4 – počáteční populace se vyvíjí do složitých, zajímavých struktur, které jsou schopny přežít po dlouhou dobu. Může se také vyvíjet

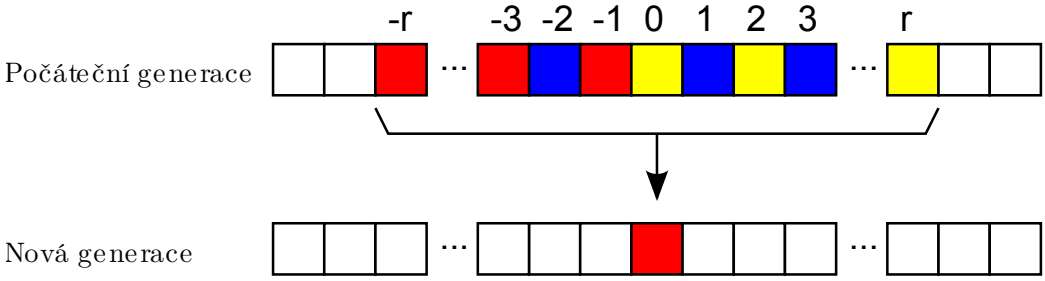
do stabilních nebo oscilujících struktur, ale počet kroku potřebných k dosažení tohoto stavu může být velmi vysoký.

Podle Stephena Wolframa není možné jednotlivá pravidla přiřadit do jedné specifické třídy, protože podle jedné definice může pravidlo ukazovat na některé vlastnosti jedné třídy a podle jiné definice na vlastnosti jiné třídy.

Později se uskutečnilo několik dalších pokusů o třídění celulárních automatů do formálně přesných tříd. Členství jednotlivých automatů v těchto třídách se však ukázalo jako nerozhodnutelné. [13, 10, 12]

2.2.4 Jednodimenzionální celulární automat

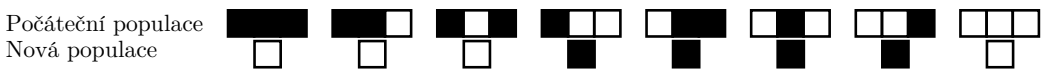
Jednodimenzionální automaty se rozdělují podle rozsahu okolí a podle počtu stavů buněk. Příklad vzniku nové generace jednodimenziálního celulárního automatu je na obrázku 2.3.



Obrázek 2.3: Vznik nové generace jednodimenzionálního automatu [2]

Dvojstavový jednodimenzionální celulární automat

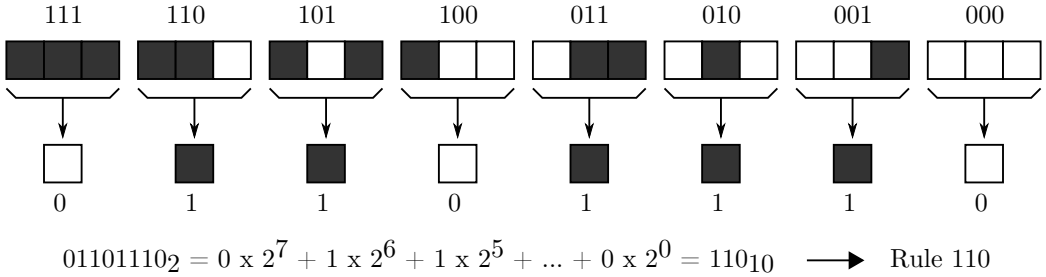
Nejjednodušším typem automatu je jednodimenzionální dvoustavový celulární automat podle vzoru Stephena Wolframa, který je také nazýván jako elementární (obrázek 2.4). U elementárního celulárního automatu může každá buňka nabývat dvou stavů (0 nebo 1).



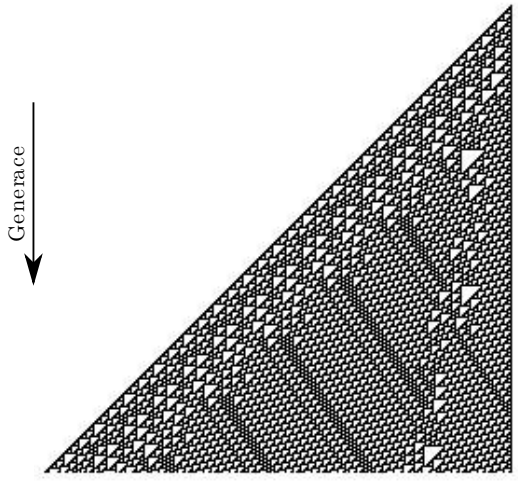
Obrázek 2.4: Příklad definice pravidla pro elementární celulární automat [2]

Pravidla pro jednodimenzionální celulární automat, jehož okolí je definováno jako buňka samotná a buňky bezprostředně sousedící, jsou popsána osmiciferným číslem v binární podobě. Každá cifra čísla reprezentuje výsledný stav buňky v další generaci pro jednotlivé přechodové funkce pravidla, kterých je $2^3 = 8$. Při stanovení pravidla celulárních automatů používáme binární číslo v desítkové soustavě. Každé osmiciferné číslo v binární podobě reprezentuje pravidlo pro celulární automat, z toho vyplývá, že je $2^8 = 256$ různých pravidel, která jsou známá jako Wolframův kód. [17].

Jako příklad je uvedeno pravidlo 110, které je v binární formě 01101110_2. Jeho přechodové funkce a odvození označení jsou na obrázku 2.5. Vývoj pravidla 110 je zobrazen na obrázku 2.6, kde počáteční populace je na horním (prvním) řádku a řádky pod ním zobrazují vývoj automatu v dalších generacích, který se řídí podle přechodových funkcí z obrázku 2.5.



Obrázek 2.5: Odvození pravidla 110 Wolframova kódu [2]



Obrázek 2.6: Pravidlo 110 [34]

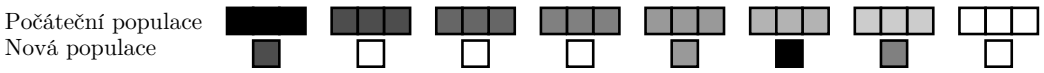
Na obrázku 2.4 je vidět příklad stavů, kterých může jednodimenzionální celulární automat nabýt. Na horním řádku jsou zobrazeny možné hodnoty

tří sousedních buněk a na spodním řádku je zobrazena výsledná hodnota centrální buňky v další generaci. Nastavení pravidel pro přechod do další generace na obrázku 2.4 je pouze ilustrativní. [13, 14, 2]

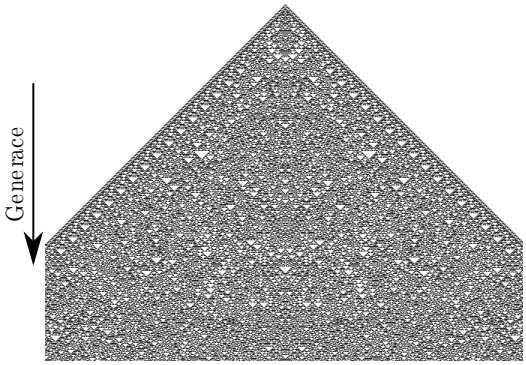
K-stavový jednodimenzionální celulární automat

Dalším typem jednodimenzionálního automatu je k-stavový celulární automat označován také jako souhrnný automat. Souhrnný automat byl vytvořen Stephenem Wolframem. Funguje na podobném principu jako elementární celulární automat, s tím rozdílem, že každá buňka může nabývat k hodnot. Jako u elementárního automatu je klíčové, v jakém stavu se nachází daná buňka, její levý a pravý soused. U souhrnného automatu je však hodnota buňky v další generaci určena na základě průměrné hodnoty dané trojice, jak je zobrazeno na obrázku 2.7. [14]

Jako příklad je uveden vývoj pravidla 777, který patří do kategorie k-stavových jednodimenzionálních celulárních automatů a je zobrazen na obrázku 2.8. Počáteční populace automatu je na horním (prvním) řádku a řádky pod ním zobrazují vývoj automatu v dalších generacích.



Obrázek 2.7: Souhrnný automat [2]



Obrázek 2.8: Pravidlo 777 [20]

2.2.5 Dvojdimenzionální celulární automat

Dvojdimenzionální celulární automaty se rozdělují podle počtu stavů, kterých může buňka nabýt a podle typu a velikosti okolí, ze kterého je určen stav buňky v nové generaci.

Okolí rozdělujeme do čtyř základních skupin [5, 9, 2]:

- *Neumanovské okolí* (obrázek 2.9a) je tvořené čtyřmi přilehlými buňkami,
- *Moorovo okolí* (obrázek 2.9c) je tvořeno přilehlými buňkami a buňkami dotýkajícími se pouze rohy,
- *šestiúhelníkové okolí* (obrázek 2.9b), buňka je znázorněna jako šestiúhelník a okolí je tvořeno všemi šesti přilehlými sousedy,
- *trojúhelníkové okolí* (obrázek 2.9d), buňka je znázorněna jako trojúhelník a okolí je tvořeno třemi přilehlými buňkami.

Všechny výše zmíněné skupiny je možné rozšířit, například Moorovo rozšířené okolí je zobrazeno na obrázku 2.9e.

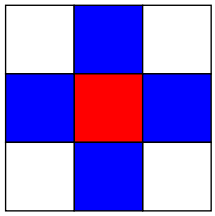
Hra života

Nejznámějším zástupcem kategorie dvojdimenzionálních automatů je Hra života (Game of Life) [5, 11] od britského matematika Johna Conwaye. Důvodem pro název tohoto automatu je jeho podobnost k vývoji živých organismů.

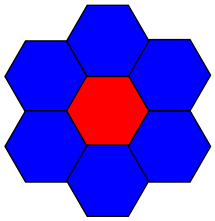
Hra života používá pro určení dalšího stavu buňky Moorova okolí, přičemž každá z nich může nabýt stavu 0 nebo 1.

Pravidlo je dáno následovně:

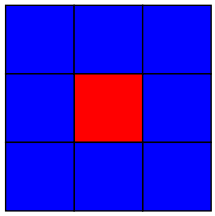
- Je-li buňka živá (stav 1) a má počet sousedů roven 2 nebo 3, buňka přežívá (stav 1),
- je-li buňka mrtvá (stav 0) a počet sousedů je roven 3, buňka ožívá (stav 1),



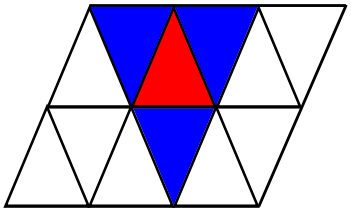
(a) Neumannovské okolí



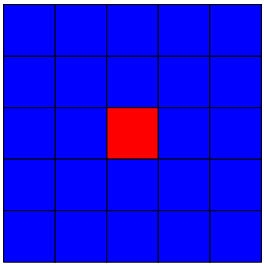
(b) Šestiúhelníkové okolí



(c) Moorovo okolí



(d) Trojúhelníkové okolí



(e) Moorovo rozšířené

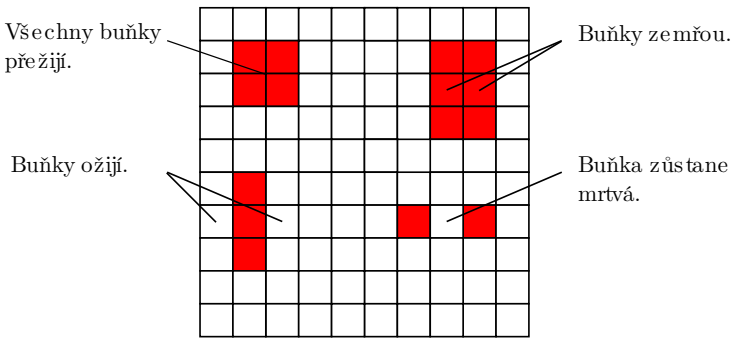
Obrázek 2.9: Základní typy okolí pro dvojdimenzionální celulární automaty

- je-li buňka živá (stav 1) a počet sousedů je menší než 2 nebo větší než 3, buňka umírá (stav 0).

Pravidlo Hry života (obrázek 2.10) se označuje jako pravidlo 23/3, kde číslo před lomítkem označuje, s jakým počtem sousedů buňka přežije a číslo za lomítkem označuje s jakým počtem sousedů buňka ožije. Ve všech ostatních případech buňka umírá, nebo zůstává mrtvá.

Zavedení různých pravidel vzniklo mnoho verzí této hry. Například hra zvaná HighLife má pravidlo 23/36 nebo hra LongLife má pravidlo 5/345.

Hra života je automat, který demonstruje tzv. emergentní jevy. Emergentní jevy jsou jevy vzniku velmi složitého chování systému, které přímo nevychází z vlastností daného systému. V průběhu vývoje byly objeveny



Obrázek 2.10: Rule 23/3

struktury, které jsou schopny pohybu ve dvourozměrné mřížce, shluky, které jsou schopné tyto pohybující struktury vytvářet a také shluky, které tyto pohybující struktury začleňují do sebe. Tyto shluky, tzv. vzory se rozdělují do čtyř základních skupin a to „Zátiší“ (Still lives), „Oscilátory“ (Oscillators), „Děla“ (Guns) a „Vesmírné lodě“ (Spaceships). [13]

Zátiší

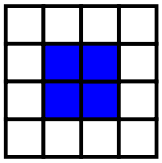
Vzory „Zátiší“ [5, 14] jsou stabilní uspořádání buněk, jejichž stav se v čase nemění. Pokud se do blízkosti této struktury dostane nestabilní struktura, pravděpodobně nastane narušení rovnováhy a zánik stabilní struktury. Příklady stabilních struktur jsou zobrazeny na obrázcích 2.11a, 2.11b, 2.11c a 2.11d.

Oscilátory

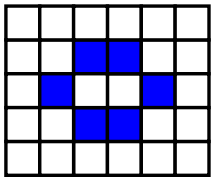
Vzory „Oscilátory“ [5, 14] jsou nestabilní vzory přecházející mezi konečným počtem konfigurací. Například oscilátory s periodou 2 postupně nabývají jedné ze dvou konfigurací a nazývají se alternátory. Příklady oscilátorů jsou zobrazeny na obrázcích 2.12a, 2.12b a 2.12c.

Děla

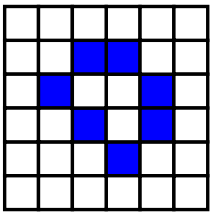
Jako „Děla“ [5, 14] (obrázek 2.13) se označují stacionární vzory, které neustále vytvářejí vzory „Hvězdné lodě“ (Spaceships) nebo „Vidle“ (Rakes).



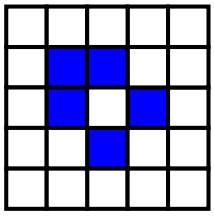
(a) Blok (Block)



(b) Včelín (Beehive)



(c) Bochník (Loaf)



(d) Loď (Boat)

Obrázek 2.11: Vzory „Zátiší“

Vesmírné lodě

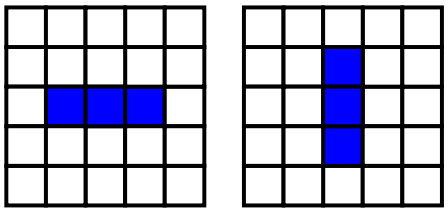
„Vesmírné lodě“ [5, 14] jsou pohybující se vzory, které se v nezměněné podobě objevují po určitém počtu generací. Mezi nejznámější hvězdné lodě patří „Kluzák“ (Glider) (obrázek 2.14), který se nachází v jedné ze čtyř fází složených z pěti živých buněk.

2.3 Speciální celulární automaty

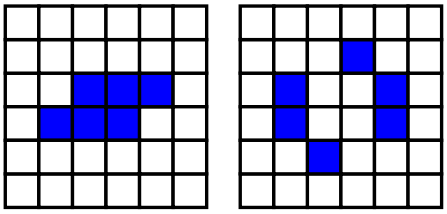
2.3.1 Coddův celulární automat

Coddův celulární automat je speciálním případem dvojdimenzionálního celulárního automatu vynalezeného britským vědcem Edgarem F. Coddem v roce 1960.

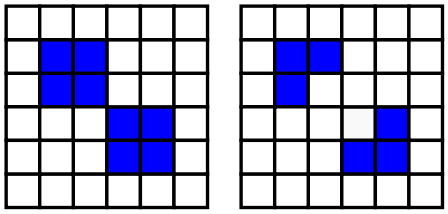
Coddův automat používá von Neumannovo okolí a 8 stavů buněk. Codd se snažil dokázat, že je možné vytvořit sebereprodukující automat s pouhými 8 stavy buněk narozdíl od von Neumannova univerzálního konstruktoru s 29 stavy. Von Neumannův univerzální konstruktor je sebereplikující stroj v prostředí celulárních automatů. [6]



(a) Blikač (Blinker)

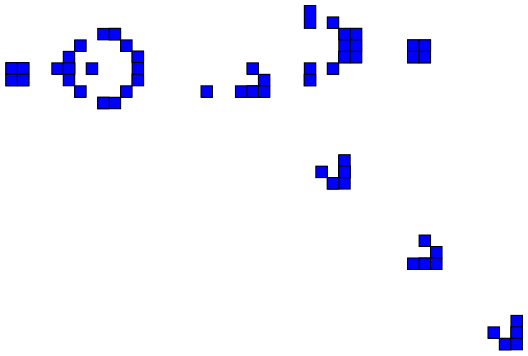


(b) Ropucha (Toad)



(c) Maják (Beacon)

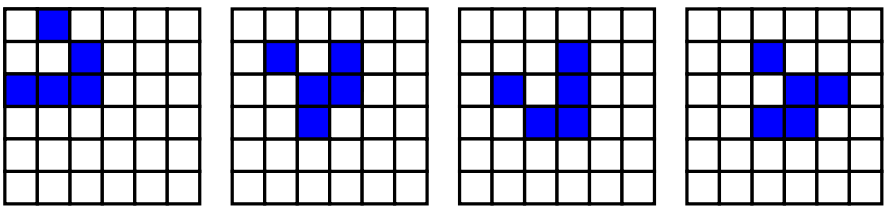
Obrázek 2.12: Vzory „Oscilátory“



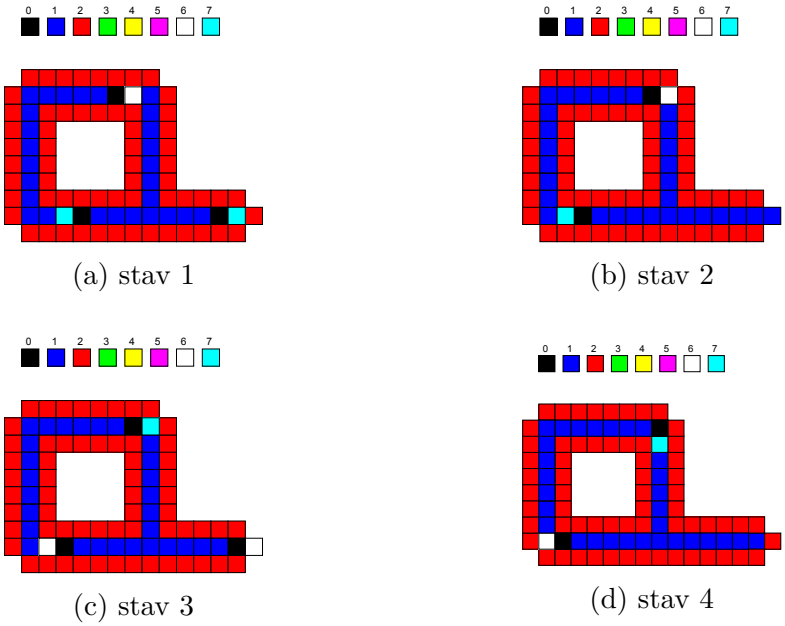
Obrázek 2.13: Děla (Guns)

Následující popis Coddova automatu včetně obrázku 2.15 byl inspirován souborem *golly-2.4-win\Patterns\Codd\repeater-emitter-demo.rle* v aplikaci Golly [15].

Na obrázku 2.15a až 2.15d je zobrazena jednoduchá konfigurace Coddova automatu. Buňky ve stavu 2 (modrá) předávají signál, buňky ve stavu 1 (červená) jsou zobrazeny jako plášť drátu. Smyčkou obíhají dva typy signálu a jsou duplikovány na T-křižovatce. První signál, tedy buňky ve stavu 7-0, způsobí obnažení konce drátu (obrázek 2.15a a 2.15b) a druhý signál, tedy buňky ve stavu 6-0, drát opět uzavře a drát je tedy o jednu buňku delší (obrázek 2.15c a 2.15d).



Obrázek 2.14: Kluzák (Glider)



(a) stav 1 (b) stav 2

(c) stav 3 (d) stav 4

Obrázek 2.15: Coddův automat

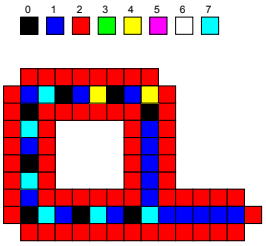
2.3.2 Langtonovy smyčky

Langtonovy smyčky [1] (viz obrázek 2.16) jsou speciálním případem umělého života v celulárním automatu vytvořeném v roce 1984 Christopherem Langtonem. Langtonovy smyčky jsou založeny na jednoduché konfiguraci Coddova automatu.

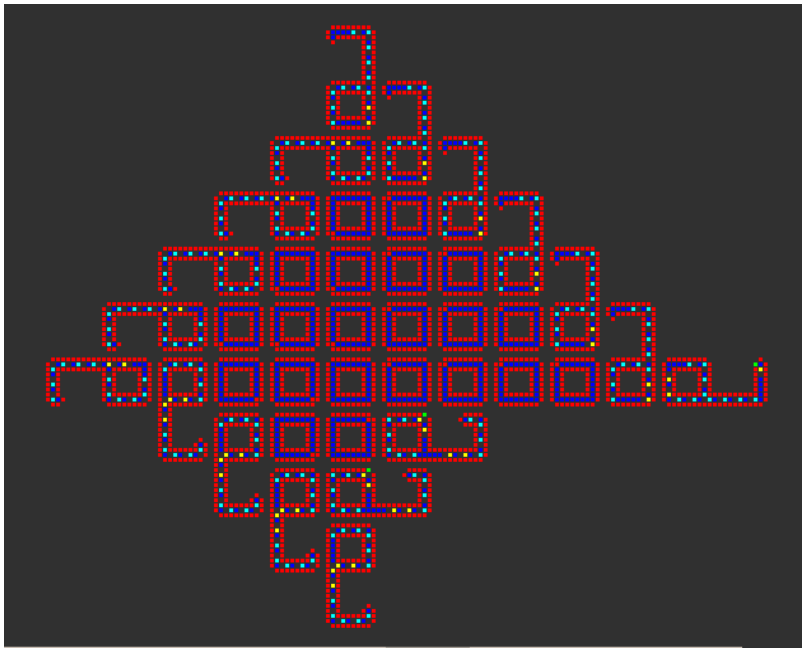
Celulární automat pro Langtonovy smyčky má 8 stavů a používá von Neumannovo prostředí, stejně jako Coddův automat a také se skládají z drátů v plášti. Buňky v drátu nesou informaci dokud nedosáhnou konce drátu a poté vykonají příslušnou akci (danou informací). Při vývoji Langtonových smyček vzniká tzv. kolonie (obrázek 2.17).

Obrázky 2.16 a 2.17 byly převzaty ze souboru *golly-2.4-win\Patterns*

Loops\Langtons-Loops.rle v aplikaci Golly [15].



Obrázek 2.16: Startovní konfigurace Langtonovo smyčky

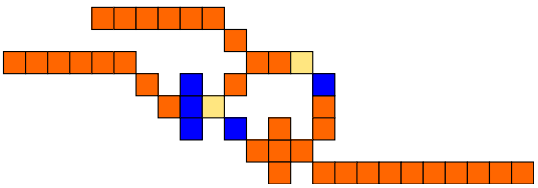


Obrázek 2.17: Kolonie Langtonových smyček

2.3.3 Wireworld

Wireworld [33] je celulární automat navržený Brianem Silvermanem v roce 1984. Wireworld je vhodný pro simulaci elektrických logických prvků nebo bran (viz obrázek 2.18) pomocí jednoduchých pravidel. Buňka ve Wireworldu se může nacházet v jednom ze čtyř stavů a používá se Mooreovo okolí.

Obrázek 2.18 byl převzaty ze souboru *golly-2.4\Patterns\WireWorld\ gate-AND.mcl* v aplikaci Golly [15].

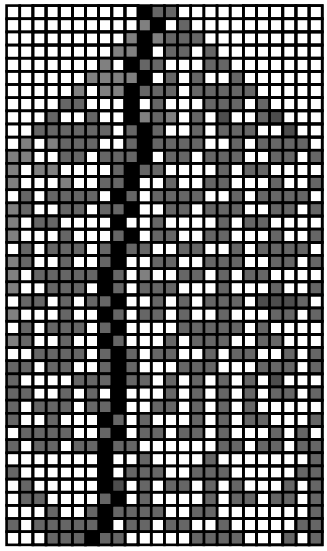


Obrázek 2.18: Wireworld – AND brána [15]

2.4 Využití celulárních automatů

Celulární automaty slouží pro modelování a simulaci diskrétních dynamických systémů. Úspěšně se využívají u případů, kde se vzájemné působení daných prvků omezuje na své nejbližší okolí. Celulárními automaty je možné řešit nejrůznější ekologické, biologické, ale také fyzikální, chemické či technické problémy, jako například šíření lesních požárů, degradace materiálu, modelování barevného pigmentu mušlí nebo růst krystalu. [14]

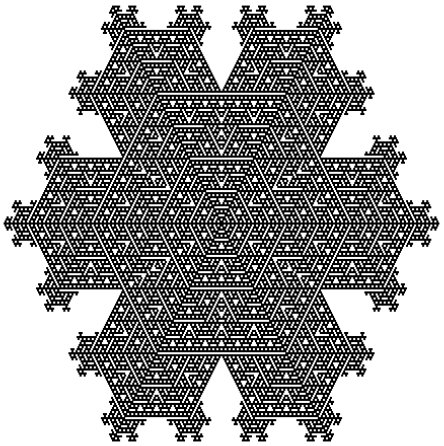
Na obrázku (obrázek 2.19) je vyjádřen průběh lomu materiálu. Vzniklé dislokace jsou znázorněny černými buňkami. Jednotlivé materiály jsou odlišeny pouze hustotou zaplněných buněk. Dislokace jsou v tomto případě, tedy podle Stephena Wolframa, voleny náhodně. [16]



Obrázek 2.19: Degradace materiálu podle Stephena Wolframa [16]

Na obrázku (obrázek 2.20) je znázorněn růst krystalu, v tomto případě vznik sněhové vločky. Zvolené pravidlo buď ovlivní růst krystalu tak, že do-

sahuje stromového tvaru nebo tvaru s hladkou stranou. Pravidlo má speciální vlastnost, že pokud určitá buňka ožije, již neumírá. Tím při různých inicializačních stavech vznikají různé typy vloček, což je shodné se vznikem vloček v přírodě. [14]



Obrázek 2.20: Sněhová vločka podle Stephena Wolframa (generace 107) [19]

Celulární automaty se také používají pro generování tzv. SEC-DEC[1] kódů, což jsou Hammingovy kódy s přidanou paritou, které slouží pro detekci dvojité chyby a opravování jednoduché chyby při přenosu dat. Tento problém je blíže popsán v literatuře [8, 7].

[1]SEC-DEC – „single error correction, double error detection“

3 Existující řešení

3.1 Mirek's Cellebration

Mirek's Cellebration [32] nebo také MCell je volně dostupný program určený pro simulaci jednodimenzionálních a dvojdimenzionálních celulárních automatů pro platformu Windows. MCell je naprogramován v jazyce Object Pascal, ale existuje také verze v jazyce Java.

Aplikace MCell obsahuje databázi předdefinovaných pravidel. Aplikace MCell dokáže načíst pravidlo ze sedmi formátů souborů, které jsou určeny pro popis celulárních automatů.

3.1.1 Vlastnosti

Vlastnosti aplikace Mirek's Cellebration:

- Obsahuje předdefinovaná pravidla,
- umí načíst pravidlo ze sedmi typů souborů, které jsou určeny pro popis celulárních automatů, formáty souborů jsou popsány v kapitole 3.1.2,
- umožňuje definovat vlastní pravidla následujícími způsoby:
  - vytvořením jednoho ze souborů popsaných v kapitole 3.1.2,
  - vytvořením DLL souboru, který lze dynamicky nahrát za běhu programu,
  - editací existujícího pravidla přímo v aplikaci.

3.1.2 Formáty souborů pro popis celulárních automatů

Formát MCLife (*.MCL)

MCLife [22] je prostý textový soubor ve formátu ASCII, který kombinuje formáty Life 1.05 a RLE. Dále zavádí rozšíření nezbytná pro uložení všech MCell funkcí. V souboru MCLife ve verzi 4.0 lze nastavit tyto vlastnosti:

- typ hry, například „Life“, „Generations“, „Weighted Life“, atd.,
- pravidlo ve formě textu (například „23/3“), kde čísla před lomítkem značí při jakém počtu sousedů buňka přežívá a čísla za lomítkem při jakém počtu sousedů buňka zemře, v ostatních případech buňka umírá nebo zůstává mrtvá,
- rychlost simulace pravidla (0 až 5000), udává zpoždění v milisekundách mezi jednotlivými generacemi,
- velikost automatu (například 300×300),
- počet stavů buňky (2 až 256),
- okrajové podmínky,
- import barevné palety pro stavy buněk,
- nastavení počáteční populace.

Příklad souboru ve formátu MCLife je uveden v příloze A.

Formát Life 1.05 (*.LIF, *.LIFE)

Soubor ve formátu Life 1.05 [22] je prostý textový soubor ve formátu ASCII. Jednotlivé řádky souboru by neměly přesáhnout délku 80 znaků.

Základní vlastnosti formátu Life 1.05:

- Nastavení počáteční populace,
- pravidlo automatu (například „23/3“), kde čísla před lomítkem značí při jakém počtu sousedů buňka přežívá a čísla za lomítkem při jakém počtu sousedů buňka zemře, v ostatních případech buňka umírá nebo zůstává mrtvá.

Příklad souboru ve formátu Life 1.05 je uveden v příloze B.

Novějším formátem je formát Life 1.06. Je to prostý textový soubor ve formátu ASCII, který je pouze seznam živých buněk v podobě jejich souřadnic „x“ a „y“.

Formát RLE (*.RLE)

Tento formát používá pro zápis počáteční populace kompresi RLE. RLE („Run Length Encoded“) je bezeztrátová komprese, která kóduje data tak, že posloupnost stejných znaků kóduje do dvojic (délka posloupnosti, hodnota), čímž se formát RLE [22] stává vhodným pro velké vzory. Jednotlivé řádky RLE souboru nesmí přesáhnout délku 70 znaků.

Základní vlastnosti formátu RLE:

- velikost automatu (například $x = 300$, $y = 300$),
- pravidlo automatu (například 23/3),
- nastavení počáteční populace ve formě ⟨`run count`⟩ ⟨`tag`⟩, kde ⟨`run count`⟩ je počet výskytů znaku ⟨`tag`⟩, konec řádku je označen znakem „!“.

Příklad reprezentace vzoru „kluzák“ v RLE formátu:

```
x = 3, y = 3
3o$o$bo!
```

Další formáty souborů

Další formáty souborů [22] podporované aplikací MCell jsou dbLife (*.L) a XLife 2.0 (*.LIF, *.LIFE), které jsou kombinací některých z výše popsaných formátů. Posledním formátem je ProLife (*.PLF), což je jediný formát v binární podobě.

3.2 Cafun

Cafun [3] (Cellular Automata Fun) slouží k simulaci dvojdimenzionálních celulárních automatů. Cafun je napsán v programovacím jazyce Java.

Cafun umožňuje vytvářet simulace komplexních systémů, jako například sociálních skupin, živých organismů, fyzikálních procesů nebo chemických

struktur. Cafun zachycuje základní rysy komplexních systémů, zejména jejich sklon k sebeorganizaci.

3.2.1 Vlastnosti

Vlastnosti aplikace Cafun:

- obsahuje jedno předdefinované pravidlo,
- dodefinování vlastních pravidel formou XML souboru.

3.2.2 Formát XML souboru

Základní vlastnosti celulárního automatu nastavitelné v souboru XML [4]:

- počet stavů buněk,
- barva jednotlivých stavů,
- nastavení tzv. „aktivních“ a „pasivních“ buněk, „pasivní“ buňka je ohodnocena, pouze je-li v jejím okolí buňka „aktivní“ (tento parametr byl zaveden z důvodu zrychlení simulace),
- pravidlo automatu (v tomto případě zvané „mutace“),
- pravděpodobnost, se kterou bude aplikováno pravidlo na buňku (určuje se vždy při přechodu do další generace),
- typ buňky, lze nadefinovat několik typů buněk, přičemž každý typ buňky má definované jiné pravidlo.

3.3 Cellular Automaton Explorer

Cellular Automaton Explorer [21] (CA Explorer) je volně dostupný nástroj pro výuku a vývoj jednodimenzionálních a dvojdimenzionálních celulárních automatů v jazyce Java. Cellular Automaton Explorer je dostupný pro platformy Windows, Linux a Mac OS.

Hlavní myšlenkou vzniku aplikace Cellular Automaton Explorer byla snaha o umožnění simulace libovolného pravidla pro celulární automaty.

3.3.1 Vlastnosti

Vlastnosti aplikace Cellular Automaton Explorer [21]:

- obsahuje předdefinovaná jednodimenzionální a dvojdimenzionální pravidla,
- možnost dodefinování vlastních pravidel formou zdrojového kódu v jazyce Java (viz kapitola 3.3.2) nebo editací existujícího pravidla přímo v aplikaci (viz kapitola 3.3.2),
- různé typy okolí (čtvercové, trojúhelníkové, šestiúhelníkové),
- možnost uložení textových dat (statistiky populace), obrázků nebo dokonce video záznamu.

3.3.2 Způsoby dodefinování pravidla

Pravidlo je definováno v podobě zdrojového souboru programovacího jazyka Java. Do aplikace lze načíst pouze přeložený soubor. Po načtení pravidla je možné v aplikaci dále nastavit:

- typ okolí buňky,
- velikost automatu,
- okrajové podmínky,
- počet stavů,
- počáteční populaci (náhodná, načtená z obrázku, načtená z jiného pravidla),

Po úpravě pravidla v aplikaci lze pravidlo uložit jako „*.ca“ soubor (formát souboru aplikace CA Explorer), který je poté možné načíst do aplikace. [21]

3.4 Fast Cellular Automata Simulator

Fast Cellular Automata Simulator [23] (FCAS) je program pro simulaci dvojdimenzionálních celulárních automatů. FCAS je napsán v programovacím jazyce Java.

3.4.1 Vlastnosti

Vlastnosti aplikace Fast Cellular Automata Simulator [23]:

- obsahuje předdefinovaná pravidla,
- možnost vytvoření vlastního pravidla (viz kapitola 3.4.2),
- podpora načtení externích souborů formátu MCLife popsaného v kapitole 3.1.2,
- simulace dvojdimenzionálních celulárních automatů.

3.4.2 Vytvoření pravidla

Pravidlo v aplikaci FCAS lze vytvořit nastavením vlastností v běžící aplikaci. Základní vlastnosti pravidla:

- volba jednoho ze vzorů (Hra života, Větší než život, Generace, Cyklický celulární automat, Vlastní pravidlo),
- podle volby vzoru jsou umožněny některé následující vlastnosti:
  - počet buněk pro přežití,
  - počet buněk pro zánik,
  - počet stavů,
  - velikost okolí,
  - typ okolí,
  - volba přechodových podmínek,
- volba počáteční populace nakreslením,
- volba barvy stavů z připravené škály barev.

3.5 Golly

Golly [26] je volně dostupná multi-platformní aplikace pro simulaci celulárních automatů. Aplikace Golly je napsána v programovacím jazyce C++.

3.5.1 Vlastnosti

Vlastnosti aplikace Golly [26]:

- simulace jednodimenzionálních a dvojdimenzionálních celulárních automatů,
- obsahuje předdefinovaná pravidla,
- načtení externích pravidel ze souborů formátu Life1.05/1.06, dbLife, RLE a MCLife popsaných v kapitole 3.1.2 a formátu Macrocell popsaném v kapitole 3.5.2,
- volba okrajových podmínek,
- různé typy okolí,
- 256 stavů buněk.

3.5.2 Formát Macrocell

Formát Macrocell [25] se používá pro obrovské vzory, které by v jiném formátu byly nepřiměřeně velké.

Formát Macrocell se skládá ze dvou hlavních částí hlavičky a stromu. V hlavičce může být definováno pravidlo a číslo generace. Strom slouží pro popis populace pravidla. Strom je reprezentován tzv. *čtyřstromem* (quadtree).

Příklad formátu Macrocell je uveden v příloze C.

4 Použité technologie

4.1 Technologie Java Web Start

Java Web Start [28] (JWS) je technologie, která umožňuje nasazení aplikace bez instalace odkazem ve webovém prohlížeči. Plnohodnotná aplikace je automaticky stažena a spuštěna. Libovolnou aplikaci lze umístit na webový server a vytvořit soubor JNLP (Java Network Launching Protocol), který umožní její spuštění. Aplikace musí být vždy připravena do jednoho nebo více souborů JAR. Všechna data, která aplikace potřebuje pro svůj běh, musí být také v souboru JAR.

Výhodou tohoto způsobu distribuce aplikace je jednoduchá aktualizace pouhým umístěním aktualizovaného souboru JAR na web. Nevýhodou je nutnost stažení aplikace z internetu před jejím spuštěním.

Nasazení Java Web Start aplikace zahrnuje následující kroky:

- Nastavení webového serveru (definice MIME typu pro JNLP: `application/x-java-jnlp-file`),
- vytvoření souboru JNLP,
- umístění aplikace na webový server,
- umístění odkazu na webovou stránku.

Nastavení webového serveru

Chceme-li nasadit aplikaci s Java Web Start na webový server, musíme zajistit, aby webový server uměl pracovat s JNLP soubory. Webový server musí být nastaven, aby vracel správný MIME typ k souborům JNLP. Nastavení MIME typu pro JNLP závisí na webovém serveru, který používáme. Pro webový server Apache stačí přidat do souboru `mime.types` řádek `application/x-java-jnlp-file`.

Vytvoření souboru JNLP

Ke spuštění Java Web Start aplikace je potřeba vytvořit soubor JNLP (Java Network Launching Protocol). Soubor JNLP je soubor XML, který obsahuje elementy a atributy, které sdělují jak aplikaci Java Web Start spustit.

Příklad JNLP souboru:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0"
        codebase="URL of application on your Web server"
        href="Notepad.jnlp">
    <information>
        <title>Notepad Demo</title>
        <vendor>Sun Microsystems, Inc.</vendor>
        <offline-allowed/>
    </information>
    <resources>
        <jar href="Notepad.jar"/>
        <j2se version="1.3+"
            href="http://java.sun.com/products/autodl/j2se"/>
    </resources>
    <application-desc main-class="Notepad"/>
</jnlp>
```

Umístění odkazu na webovou stránku

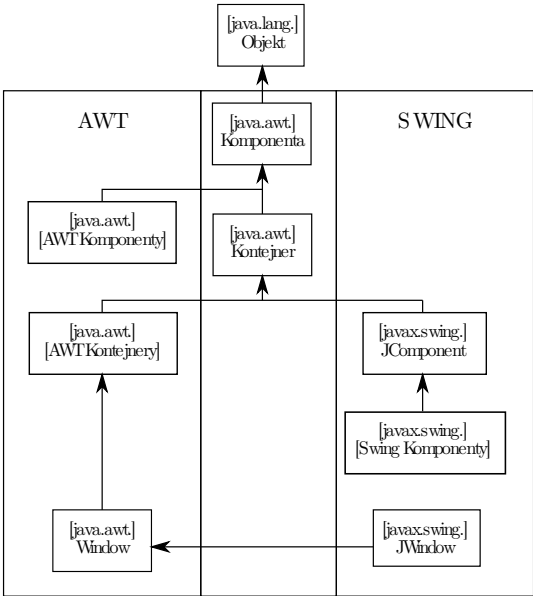
Pro spuštění Java Web Start aplikace z webové stránky je nutné na webovou stránku umístit odkaz na JNLP soubor aplikace. Jako odkaz se používá standardní HTML zápis s atributem *href* specifikujícím umístění JNLP souboru:

```
<a href="Notepad.jnlp">Launch Notepad Application</a>
```

4.2 Java Swing

Swing [27] je knihovna pro tvorbu grafického uživatelského rozhraní (GUI) programů v jazyce Java. Swing byl vyvinut s cílem poskytnout propracovanější sadu komponent než poskytuje knihovna AWT, což je starší knihovna na tvorbu GUI v jazyce Java, na níž je Swing postaven (obrázek 4.1). Komponenty knihovny AWT jsou implementovány platformně závislým kódem. Na rozdíl od toho jsou komponenty v knihovně Swing implementovány v jazyce Java, a proto je GUI Swing platformně nezávislé.

Knihovna Swing používá architekturu Model-View-Controller (MVC), což je softwarová architektura, která odděluje datový model, uživatelské rozhraní



Obrázek 4.1: Hierarchie tříd knihovny Swing a propojení s AWT [24]

a řídící logiku aplikace. Hlavní výhodou této technologie je, že úprava jedné části má minimální vliv na části ostatní.

4.3 Další použité technologie

RSyntaxTextArea

RSyntaxTextArea [30] je komponenta Swing, určená k použití v GUI, zvýrazňující syntaxi zdrojových kódů různých programovacích jazyků

RSyntaxTextArea obsahuje:

- Zvýraznění syntaxe pro více než 30 programovacích jazyků,
- dokončování rozepsaného kódu,
- funkce najít/nahradit,
- neomezené undo/redo,
- funkce drag and drop,

- funkce na kontrolu závorek,
- zvýraznění aktuálního řádku.

MediaTool API

MediaTool API [29] je jednoduché API, které používá knihovnu Xuggler. Xuggler je obsáhlá knihovna, která umožňuje dekompresi, úpravu a následnou kompresi jakéhokoliv video formátu v programovacím jazyce Java.

5 Implementace

5.1 Analýza

Navrhovaná aplikace obsahuje několik základních hledisek, které je nutné zohlednit. V první řadě se jedná o obecné jádro vykonávající činnost celulárního automatu. Dalším aspektem je podpora uživatelem definovaných pravidel. Následující část tvoří výkonná vizualizace aktuálního stavu automatu. A v závěru je nutné navrhnout uživatelské rozhraní, které bude dobře sloužit jak k ovládání běhu automatu tak k snadné definici vlastního pravidla.

5.1.1 Jádro celulárního automatu

Nutností při návrhu aplikací je oddělení jednotlivých částí aplikace do samostatných celků. Jádro automatu je samostatnou částí, která by měla být zároveň nezávislá na ostatních částech jako např. vizualizaci. Základními požadavky na funkčnost jsou:

- nastavení pravidla,
- přechod do další generace,
- reset do počátečního stavu,
- nastavení počáteční populace
- a nastavení velikosti života.

5.1.2 Uživatelem definovaná pravidla

Definice vlastních pravidel je jedním z důležitých aspektů použití softwaru pro celulární automaty. Existují dva důležité avšak protichůdné požadavky. Prvním je snadnost definice vlastního pravidla a druhým je možnost definice libovolného pravidla.

Řada programů uvedených v kapitole 3 má možnost snadné definice vlastního pravidla. Jedná se např. o pouhý zápis dvou čísel, která udávají při jakém

počtu živých sousedů buňka přežije a při jakém počtu sousedů buňka ožije. Ve všech ostatních případech buňka umírá, nebo zůstává mrtvá. Zmíněný způsob je velice jednoduchý avšak zároveň velmi omezující.

Oproti tomu program Mirek's Cellebration popsaný v kapitole 3.1 umožňuje načtení vlastního pravidla z dynamické knihovny dll. Tímto způsobem lze definovat prakticky libovolné pravidlo. Nevýhodou tohoto způsobu je náročnost kladená na uživatele.

5.1.3 Vizualizace stavu automatu

Na první pohled nevýznamná věc jako je vizualizace stavu automatu, nabývá na významu zejména tehdy, pokud potřebujeme simulovat automat velkých rozměrů po mnoho generací. Již automat 1000×1000 obsahuje milión buněk, které je třeba při každém přechodu do následující generace vykreslit. Pokud bychom počítali s RGB barvou a buňkou o velikosti 5×5 pixelů, jedná se o 75 MB dat na jedno vykreslení stavu. Z tohoto je patrné, že výkon vizualizační části bude mít značný vliv na celkový výkon simulace automatu.

5.1.4 Uživatelské rozhraní

Intuitivní uživatelské rozhraní je základem každé úspěšné aplikace. Pro návrh uživatelského rozhraní neexistuje žádný exaktní postup, kvalitní rozhraní je často výsledkem řady iterací, které jsou doprovázeny uživatelskými testy.

5.2 Celulární automat

Jedná se o dvojdimenzionální celulární automat s možností definování pravidel uživatelem. Definice pravidla bude v samostatném souboru ve formě zdrojového kódu programovacího jazyka Java. Samotná implementace pravidla by měla být oddělena od GUI, typu hranic, typu mřížky a od stavu života.

Požadovaného oddělení lze dosáhnout vhodným návrhem rozhraní pro třídu pravidla. Samotný celulární automat pak bude toto pravidlo využívat pro určení další generace.

V rámci pravidla by mělo být možné definovat:

- jméno pravidla,
- počet stavů, kterých může automat nabýt,
- barvy stavů, každému stavu bude přiřazena barva,
- přechodová funkce, samotné pravidlo pro přechod stavu do další generace,
- počáteční populace, uživateli bude umožněno vytvoření libovolné počáteční populace,
- typ hranice, bude určovat okrajové podmínky automatu,
- uživatelské akce, uživateli bude umožněno definovat libovolné akce (pro každou akci bude vytvořeno tlačítko spouštící tuto akci).

Rozhraní pravidla vypadá následovně:

```
1   /* Návratová hodnota spojité hranice. */
2   public static final int BOUND_TYPE_CONTINUOUS = Integer.MIN_VALUE;
3
4   /* Inicializace pravidla. */
5   public void initialization(Action stdAction);
6
7   /* Zjištění typu hranice. */
8   public int getBoundsType();
9
10  /* Zjištění akcí dostupných uživateli. */
11  public String[] getUserActions();
12
13  /* Spustí akci definovanou uživatelem. */
14  public void runAction(String action);
15
16  /* Vrací jméno pravidla. */
17  public String getName();
18
19  /* Vrací počet stavů. */
20  public int getNumberStates();
21
22  /* Vrací barvu stavu. */
23  public java.awt.Color getColor(int state);
24
25  /* Vytvoří počáteční populaci. */
26  public boolean initialPopulation(int[][] matrix);
27
28  /* Hlavní metoda, kde je definováno pravidlo pro celulární automat. */
29  public int applyRule(Cell cell);
```

RuleInterface.java

Přechodová funkce převádí současný stav buňky na nový. Pro určení nového stavu potřebuje současný stav buňky a stav sousedních buněk. Pro určení dalšího stavu buňky je nutné tyto informace předat pravidlu. Pro tento účel byla navržena třída `Cell`.

Metody třídy `Cell` jsou vztaženy k buňce, pro níž je počítán nový stav, přičemž jsou respektovány okrajové podmínky. Metody poskytované pravidlu třídou `Cell`:

- `getCellState()` – vrací současný stav buňky,
- `getNeighbor(x, y)` – vrací stav souseda, souřadnice jsou relativní,
- `aroundCells(roundSize)` – počet buněk s nenulovým stavem v okolí, velikost okolí je dána parametrem,
- `aroundCells(roundSize, state)` – počet buněk se specifikovaným stavem a okolím.

5.3 Vizualizace stavu automatu

Vizualizace stavu automatu je provedena následovně:

1. Pokud je načten obrázek pozadí, je vykreslen na pozadí a stav automatu se kreslí částečně průhledně,
2. provádí se pouze při prvním vykreslování nebo po změně rozměru automatu:
   (a) Vytvoření barevného modelu pro indexovaný[1] obrázek,
   (b) vytvoření indexovaného obrázku pro vykreslení stavu automatu,
   (c) inicializace jednorozměrného pole dat pro uložení stavu automatu.
3. naplnění pole dat aktuálním stavem automatu,
4. nastavení jednotlivých pixelů obrázku z pole dat,
5. vykreslení obrázku se zvětšením tak, aby velikost buňky odpovídala zvolené velikosti,
6. vykreslení mřížky automatu přes obrázek (pouze pokud je mřížka zapnuta).

[1] V indexovaném obrázku je použitá omezená paleta barev, nejčastěji 256 nebo méně. Výhodou indexovaných obrázků je, že jsou méně náročné na spotřebu paměti než obrázky typu RGB.

Při vytváření barevného modelu pro indexovaný obrázek je vytvořena barva pro každý stav (hodnota indexu barvy odpovídá číslu stavu). V případě, že je zvolen obrázek na pozadí, je každá barva nastavena jako částečně průhledná.

Výhody použití indexovaného obrázku jsou:

- menší spotřeba paměti,
- rychlejší vykreslení stavu automatu – hodnota stavu odpovídá hodnotě indexované barvy, již není třeba aplikovat mapování čísla stavu na barvu.

Nevýhodou použití indexovaného obrázku může být omezený počet barev, tato nevýhoda nám však nevadí. Počet barev zároveň i stavů byl omezen na 256, toto číslo je pro celulární automat více než dostačující.

5.4 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (GUI) lze rozdělit na GUI celulárního automatu a editor pravidla.

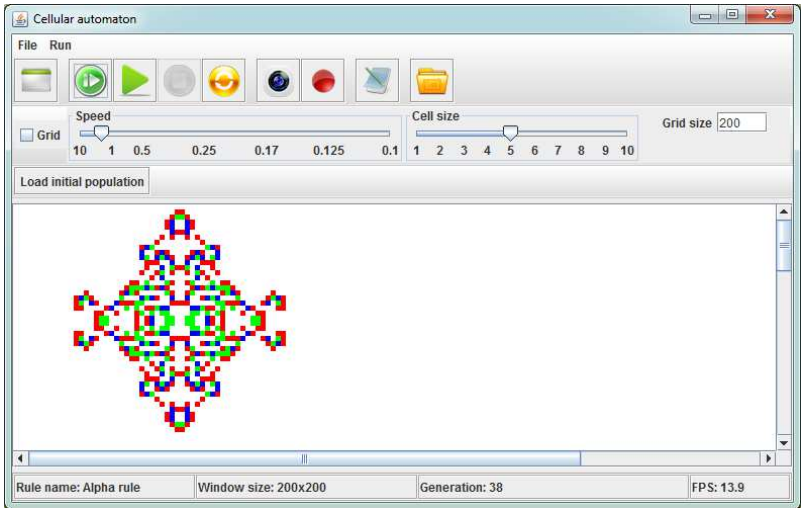
5.4.1 GUI celulárního automatu

GUI celulárního automatu je zároveň hlavním oknem aplikace a umožňuje uživateli ovládat automat. Uživatelské akce jsou:

- načtení pravidla ze souboru,
- spuštění simulace automatu,
- převedení automatu do další generace (jeden krok simulace),
- zastavení simulace automatu,
- restart automatu do počátečního stavu,
- uložení obrázku aktuálního stavu automatu,

- spuštění simulace se záznamem videa,
- otevření editoru pravidel,
- načtení obrázku pozadí,
- volba zobrazení mřížky,
- volba rychlosti simulace automatu,
- volba velikosti buněk automatu,
- změna velikosti automatu.

Hlavní okno (obrázek 5.1) lze rozdělit do několika částí. Hlavní částí je panel, ve kterém je zobrazen stav celulárního automatu. V horní části hlavního okna je nástrojová lišta, kde budou všechna tlačítka aplikace. Poslední částí hlavního okna je informační panel, který se nachází ve spodní části okna.



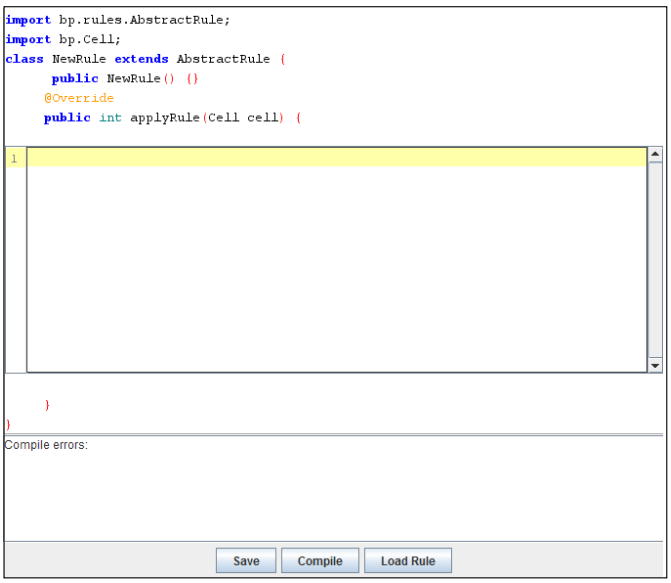
Obrázek 5.1: Hlavní okno aplikace

5.4.2 Editor pravidla

V editoru lze vytvořit, přeložit a načíst přeložené pravidlo do automatu. Možnosti při editaci pravidla:

- editace přechodové funkce – metody `applyRule` (obrázek 5.2),

- volba vlastností pravidla – typ okrajových podmínek, počet a barvy stavů (obrázek 5.3),
- vytvoření počáteční populace nakreslením nebo načtení populace ze souboru (obrázek 5.4), dále je možné načíst obrázek na pozadí plátna.

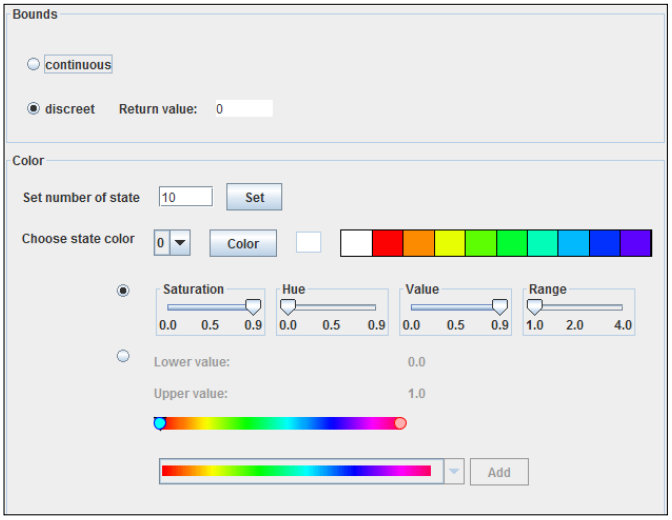


Obrázek 5.2: Okno editoru – záložka „Code“

Po implementaci `applyRule` je umožněno celou třídu zkompilovat anastavit do automatu. Pokud při kompilaci došlo k nějaké chybě, je o tom uživatel informován, stejně jako je informován o bezproblémovém překladu a nahrání. Také je zde umožněno celou třídu uložit jako soubor java.

Příklad metody `applyRule` implementující Hru života dle Johna Hortona Conwaye:

```
1  public int applyRule(Cell cell) {
2          /* Zjištění stavu buňky. */
3          int cellState = cell.getCellState();
4          /* Spočítání okolních sousedů buňky. */
5          int neighborsNumber = cell.aroundCells(1);
6
7          /* Pokud je buňka živá (ve stavu 1) */
8          if (cellState == 1) {
9              /* Pokud je počet sousedů aktuální buňky menší než 2 nebo větší než 3, buňka
10              * zemře (stav 0) */
11             if (neighborsNumber < 2 || neighborsNumber > 3) {
12                 cellState = 0;
13             /* Pokud je počet sousedů roven 2 nebo 3, buňka zůstává žít (stav 1). */
14             } else if (neighborsNumber == 2 || neighborsNumber == 3) {
15                 cellState = 1;
16             }
17         /* Pokud je buňka mrtvá (stav 0) */
18         } else {
19             /* Pokud je počet sousedů roven 3, buňka ožívá (stav 1). */
20             if (neighborsNumber == 3) {
```



Obrázek 5.3: Okno editoru – záložka „Properties“

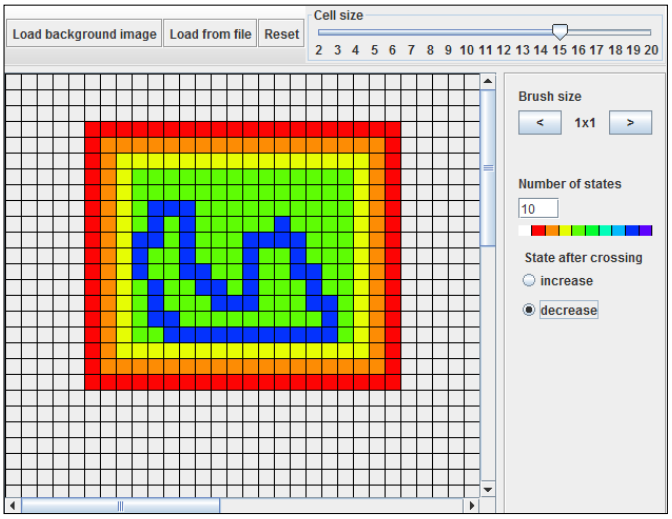
```
21                  cellState = 1;
22              /* Jinak buňka zůstává mrtvá (stav 0) */
23              } else {
24                  cellState = 0;
25              }
26          }
27          /* Vrací nový stav buňky. */
28          return cellState;
29      }
```

ConwaysRule.java

Sestavení pravidla

Sestavení pravidla spočívá ve vygenerování souboru obsahujícího třídu, která je oděděná od třídy „AbstractRule“. Vygenerovaný soubor je následně zkompilován a načten. Třída „AbstractRule.java“ je abstraktní třída, která implementuje třídu „RuleInterface.java“, tedy základní rozhraní pro vytváření pravidel. V „AbstractRule.java“ jsou nadefinovány všechny metody kromě abstraktní metody „applyRule“, kterou jako jedinou je potřeba v editoru nadefinovat pro funkčnost uživatelského pravidla.

```
1  import bp.rules.AbstractRule;
2  import bp.Cell;
3  class NewRule extends AbstractRule {
4      public NewRule() {}
5
6      /* Vytvoří počáteční populaci automatu. */
7      @Override
8      public boolean initialPopulation(int[][] matrix){
9          // zde se vytvoří populace
10     }
11
12     /* Vrátí počet stavů, ve kterých se automat může nacházet. */
```



Obrázek 5.4: Okno editoru – záložka „Population“

```
13      @Override
14      public int getNumberStates(){
15        // zde je definován počet stavů
16      }
17
18      /* Hlavní metoda, kde je definováno pravidlo automatu. */
19      @Override
20      public int applyRule(Cell cell){
21         // zde je definováno uživatelem pravidlo
22      }
23
24      /* Vrací barvu daného stavu. */
25      @Override
26      public java.awt.Color getColor(int state){
27        // zde se přiřadí barvy příslušným stavům
28      }
29
30      /* Vrací okrajové podmínky. */
31      @Override
32      public int getBoundsType(){
33        // zde jsou definovány okrajové podmínky
34      }
35  }
```

NewRule.java

5.4.3 Počáteční populace

Počáteční populaci lze načíst ze souboru. Podporovány jsou dva formáty. V prvním případě je stav 1 reprezentován hvězdičkou „*“ a stav 0 tečkou „.“ (viz tabulka 5.1). Druhý formát používá čísla, které reprezentují jednotlivé stavy (viz tabulka 5.2). Převod populace do matice je implementován ve třídě `Population` (metoda initialPopulationFromDotStar).

.	.	*	*	*	.	.		0	0	1	1	1	0	0
.	.	*	*	*	.	.	→	0	0	1	1	1	0	0
.	.	*	*	*	.	.		0	0	1	1	1	0	0

Tabulka 5.1: První způsob reprezentace populace

0,	0,	1,	2,	3,	2,	1,		0	0	1	2	3	2	1
0,	0,	1,	2,	3,	2,	1,	→	0	0	1	2	3	2	1
0,	0,	1,	2,	3,	2,	1,		0	0	1	2	3	2	1

Tabulka 5.2: Druhý způsob reprezentace populace

5.5 Výkon automatu

Důležitým hlediskem při posuzování kvality implementace celulárního automatu je jeho výkonnost. Výkonnost je posuzována v počtu generací, přes které je automat schopen přejít za jednotku času. Aplikace obsahuje synchronní vizualizaci aktuálního stavu automatu, to znamená, že každá generace je vykreslena v rámci stejného vlákna v němž byla vypočtena. Výkon vizualizace má tedy přímý dopad na výkon celé simulace a bylo nutné jej optimalizovat.

V rámci optimalizace výkonu vizualizace bylo použito indexovaných barev. Indexy barev stavů odpovídají číslům stavů, to znamená, že při vykreslování je nutné pouze přenést čísla stavů do jednorozměrného pole, jímž je následně nastaven rastr, který je předán k vykreslení.

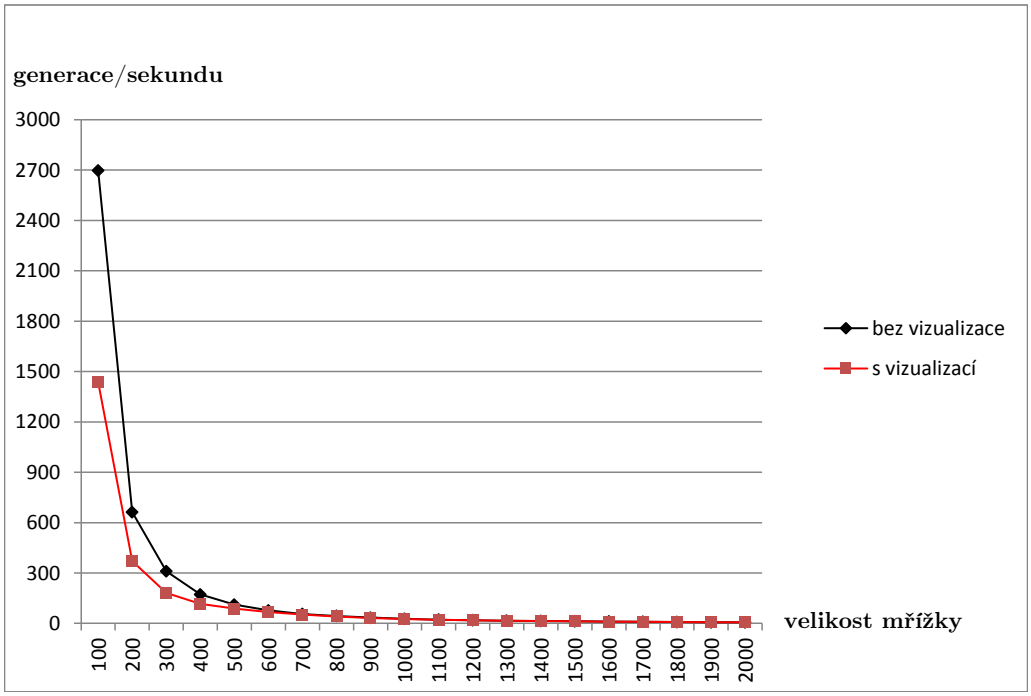
Přestože je v aplikaci možné volit velikost jedné buňky, data pro zobrazení jsou vždy připravena stejným způsobem (každá buňka o velikosti jednoho pixelu) a teprve při nastavování rastru jsou zvětšena na požadovanou velikost. Tento postup redukuje závislost výkonu aplikace na velikosti buňky.

Pro verifikaci dostatečného výkonu automatu se provedlo několik testů. Výkon byl měřen jak při simulaci bez vizualizace, tak při simulaci s vizualizací. Měření bylo provedeno na operačních systémech Windows 7 64 bit a Linux Ubuntu 64 bit. Parametry testovacího počítače jsou následující:

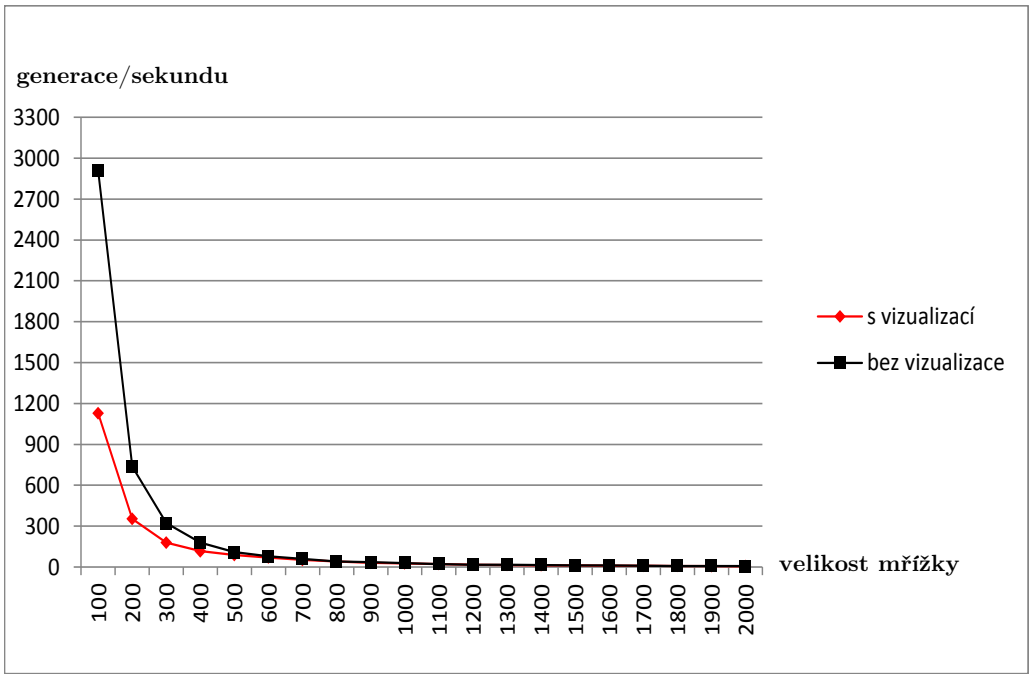
- Procesor: Intel Core i7-3517U CPU 1,90GHz,
- paměť: 4GB RAM,

- grafická karta: Intel HD Graphics 4000,
- otáčky disku: 5400 rpm.

Výsledky testů jsou vidět na obrázcích 5.5 a 5.6.



Obrázek 5.5: Výkonnostní test na operačním systému Linux Ubuntu 64 bit



Obrázek 5.6: Výkonnostní test na operačním systému Windows 7 64 bit

6 Implementovaná vzorová pravidla

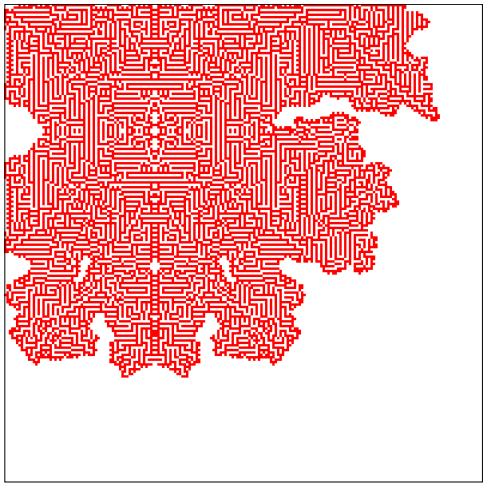
V rámci tvorby aplikace bylo naprogramováno několik různých pravidel s cílem demonstrovat možnosti při tvorbě pravidel.

6.1 Pravidlo ConwaysRule

Pravidlo `ConwaysRule` demonstruje, jak lze pravidlem načíst soubory reprezentující celulární automaty. `ConwaysRule` umí načíst libovolný textový soubor, ve kterém se musí nacházet řádka reprezentující přechodovou funkci a řádky reprezentující počáteční populaci.

Pravidlo je zapsáno jako text (například #R 12345/3), kde čísla před lomítkem značí při jakém počtu sousedů buňka zůstává živá a čísla za lomítkem při jakém počtu sousedů buňka ožije. Toto je zároveň Conwayův způsob zápisu pravidel, podle kterého je pravidlo pojmenováno. Simulace automatu s pravidlem `Maze` generace 355 je na obrázku 6.1.

Počáteční populace lze zapsat dvěma způsoby, které jsou popsány v kapitole 5.4.3.



Obrázek 6.1: Simulace ConwaysRule s pravidlem Maze, generace 355

6.2 Pravidlo ForrestFireRule

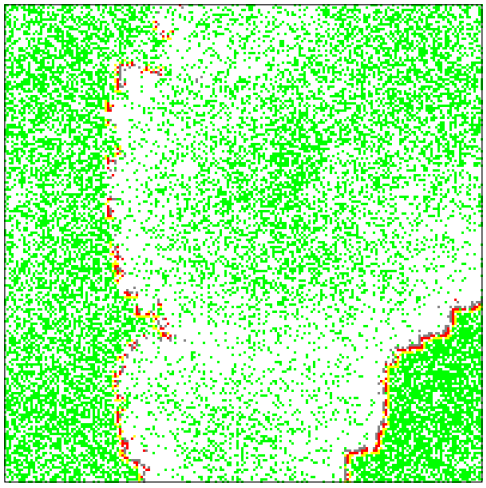
Pravidlo ForrestFireRule demonstruje simulaci lesního požáru. Pravidlo je reprezentováno následujícími přechody mezi stavy:

1. stav 0 (bílá), prázdné místo → stav 1 (zelená), strom
2. stav 1 (zelená), strom → stav 2 (oranžová), doutnání
3. stav 2 (oranžová), doutnání → stav 3 (červená), oheň
4. stav 2 (oranžová), doutnání → stav 1 (zelená), strom
5. stav 3 (červená), oheň → stav 4 (šedá), popel
6. stav 4 (šedá), popel → stav 0 (bílá), prázdné místo

přičemž:

- přechodové pravidlo 1 nastane s předem danou pravděpodobností,
- přechodové pravidlo 2 nastane s předem danou pravděpodobností nebo pokud je alespoň jedna sousední buňka ve stavu 2,
- přechodové pravidlo 3 nastane za předpokladu, že v jejím okolí jsou alespoň dvě buňky ve stavu 2,
- přechodové pravidlo 4 nastane s předem danou pravděpodobností,
- přechodové pravidlo 5 nastává vždy při přechodu do další generace,
- přechodové pravidlo 6 nastává vždy při přechodu do další generace.

Simulace automatu v generaci 247 je na obrázku 6.2.



Obrázek 6.2: Simulace lesního požáru v generaci 247

7 Závěr

Cílem práce byla implementace celulárního automatu v jazyce Java. Mezi požadavky v zadání práce na implementaci patřilo: možnost implementace přechodových funkcí, export vizualizace celulárního automatu do obrázkového formátu a export videosekvence vývoje celulárního automatu. Kromě toho byl dále implementován editor pravidel pro snažší vytváření přechodových funkcí.

Hlavním problémem při návrhu rozhraní pravidla bylo, aby implementace vlastních pravidel byla co nejvíce intuitivní. Jelikož při implementaci uživatelského pravidla je potřeba implementovat všechny potřebné metody třídy, což může být pro uživatele náročné, byl přidán do aplikace editor pravidel. V editoru pravidel je totiž možné doplnit pouze přechodovou funkci a popřípadě nakreslit počáteční populaci (nebo ji načíst ze souboru), což je narozdíl od implementování celého pravidla méně náročné.

Dalším problémem při implementaci bylo dosažení potřebného výkonu pří vykreslování automatu. Mezi některé optimalizace patří například použití indexovaných barev pro vykreslení vizualizace celulárního automatu nebo příprava dat pro vizualizaci jak je popsáno v kapitole 5.5.

Při implementaci exportu videosekvence vývoje celulárního automatu nastal problém s velikostí potřebné knihovny. Velikost knihovny nás omezuje vzhledem k zprovoznění aplikace jako Java Web Start, kde je nežádoucí stahovat velké množství dat. Problém s velikostí knihovny byl vyřešen absencí možnosti nahrávání videa při spuštění aplikace jako Java Web Start. Nahrávání videosekvence je dostupné pokud se aplikace spustí klasickým způsobem.

Do budoucna by bylo možné aplikaci rozšířit o volbu typu mřížky, reprezentaci stavů reálnými čísly, což by umožňovalo plynulejší barevný přechod mezi stavy, další typy okrajových podmínek, načtení uloženého java souboru zpět do editoru nebo možnost definice různých pravidel pro různé buňky či skupiny buněk.

Literatura

[1] DIEM, Alexandra. *Langton's Self-Replicating Loop* [online]. [cit. 2013-2-8].
Dostupné z <http://akdiem.files.wordpress.com/2012/11/report.pdf>.

[2] FLOREANO, Dario; MATTIUSSI, Claudio. *Companion slides for the book Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies* [online]. [cit. 2013-1-9]. Dostupné z <http://baibook.epfl.ch/slides/cellularSystems-slides.pdf>.

[3] HOMEYER, André. *A Brief Introduction To Cafun* [online]. [cit. 2013-2-13].
Dostupné z <http://www.cafun.de/information/a_brief_introduction_to_cafun_a4.pdf>.

[4] HOMEYER, André. *Cafun Simulation File Format Specification* [online]. [cit. 2013-2-13].
Dostupné z <http://www.cafun.de/information/cafun_simulation_file_format_specification_1.0_a4.pdf>.

[5] HUSÁKOVÁ, Martina. *Celulární automaty* [online]. [cit. 2013-1-14]. Dostupné z <http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt3/zt3_dokumenty/CelularniAutomaty.pdf>.

[6] HUTTON, Tim; *Codd's self-replicating computer* [online]. [cit. 2013-1-20].
Dostupné z <http://www.sq3.org.uk/papers/Hutton_CoddsSelfReplicatingComputer_2010.pdf>.

[7] CHO, Sung-Jin; CHOI, Un-Sook; HEO, Seong-Hun. *Design of double error correction codes based on cellular automata* [online]. [cit. 2013-2-10].
Dostupné z <http://www.mathnet.or.kr/mathnet/kms_tex/984159.pdf>.

[8] CHO, Sung-Jin; KIM, Han-Doo; PYO, Yong-Soo; PARK, Yong-Bum; HWANG, Yoon-Hee; CHOI, Un-Sook; HEO, Seong-Hun. *Single error correction code using PBCA* [online]. [cit. 2013-2-10]. Dostupné z <http://mathnet.kaist.ac.kr/mathnet/kms_tex/982303.pdf>.

[9] KROBAT, Roman. *Modelování proudění pomocí celulárních automatů* [online]. [cit. 2013-1-20]. Dostupné z <http://theses.cz/id/a1ecyo/114070-880428803.pdf>.

[10] PACKARD, Norman; WOLFRAM, Stephen. *Two-Dimensional Cellular Automata* [online]. [cit. 2013-2-2]. Dostupné z <http://new.math.uiuc.edu/im2008/dakkak/papers/files/wolfram.2dca.pdf>.

[11] PROIOS, Ivan. *Glitch: entropie jako původce estetické hodnoty* [online]. [cit. 2013-1-29]. Dostupné z <http://is.muni.cz/th/172600/fi_m/Thesis_Proios_Ivan_172600.pdf>.

[12] SUTNER, Klaus. *A Note on Culik-Yu Classes* [online]. [cit. 2013-1-15]. Dostupné z <http://www.complex-systems.com/pdf/03-1-8.pdf>.

[13] ŠALANDA, Vojtěch. *Predikce sekundární struktury proteinů pomocí celulárního automatu* [online]. [cit. 2013-1-23]. Dosupné z <http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=13748&file=t>.

[14] TOMAŠTÍK, Marek. *Celulární automaty (Game of Life)* [online]. [cit. 2013-1-5]. Dostupné z <http://autnt.fme.vutbr.cz/szz/2010/BP_Tomastik.pdf>.

[15] TREVORROW, Andrew; ROKICKI, Tom; HUTTON, Tim; GREENE, Dave; SUMMERS, Jason; VERVER, Maks; MUNAFO, Robert. *Golly* [počítačový program, online]. Ver. 2.4. [s.l.], 2012 [citováno 2013-1-20]. Dostupné z <http://sourceforge.net/projects/golly/files/golly/golly-2.4/>.

[16] WOLFRAM, Stephen. *A New Kind of Science*. Champaign, Illinois: Wolfram Media, 2002 [cit. 2013-2-12]. ISBN 1-57955-008-8.

[17] WOLFRAM, Stephen. *Statistical mechanics of cellular automata* [online]. [cit. 2013-4-20]. Dostupné z <http://www.wolframscience.com/nksonline/page-375#previous>.

[18] ŽALOUDEK, Luděk; SEKANINA, Lukáš; ŠIMEK, Václav. *Accelerating Cellular Automata Evolution on Graphics Processing Units* [online]. [cit. 2013-1-29]. Dostupné z <http://www.fit.vutbr.cz/ sekanina-/pubs.php?file=%2Fpub%2F9315%2Fzaloudek.pdf&id=9315>.

[19] *Boolean Hexagonal Automata* [online]. [cit. 2013-2-12]. Dostupné z <http://archive.vector.org.uk/art10002630>.

[20] *Cellular Automata Explained* [online]. [cit. 2013-2-4]. Dostupné z <http://www.generation5.org/content/2003/caintro.asp>.

[21] *Cellular Automaton Explorer* [online]. [cit. 2013-2-23]. Dostupné z <http://academic.regis.edu/dbahr/GeneralPages/CellularAutomata/CA_Explorer/CA_Explorer_Home.html>.

[22] *Cellular Automata files formats* [online]. [cit. 2013-2-20]. Dostupné z <http://www.mirekw.com/ca/ca_files_formats.html>.

[23] *FCAS - Cellular automata simulator* [online]. [cit. 2013-2-25]. Dostupné z <http://artax.karlin.mff.cuni.cz/~bardv5am/index.php?id=fcas>.

[24] *Difference between swing and awt* [online]. [cit. 2013-2-13]. Dostupné z <http://www.threedumbboys.com/2013/02/difference-between-swing-and-awt.html>.

[25] *Golly Help: File Formats* [online]. [cit. 2013-2-26]. Dostupné z <http://golly.sourceforge.net/Help/formats.html>.

[26] *Golly Game of Life Home Page* [online]. [cit. 2013-2-26]. Dostupné z <http://golly.sourceforge.net/>.

[27] *Java Swing Tutorial* [online]. [cit.2013-2-13]. Dostupné z <http://www.javabeginner.com/java-swing/java-swing-tutorial>.

[28] *Java Web Start Overview* [online]. [cit. 2013-2-15]. Dostupné z <http://www.oracle.com/technetwork/java/javase/jws-white-paper-150004.pdf>.

[29] *MediaTool Introduction* [online]. [cit. 2013-2-16]. Dostupné z <http://wiki.xuggle.com/MediaTool_Introduction>.

[30] *RSyntaxTextArea* [online]. [cit. 2013-2-16]. Dostupné z <http://fifesoft.com/rsyntaxtextarea/>.

[31] *The Dawn of Space and Time in a Selfconscious Quantum Universe* [online]. [cit. 2013-4-20].
Dostupné z <http://tonyb.freeyellow.com/id218.html>.

[32] *What is Mirek's Cellebration (MCell)?* [online]. [cit. 2013-2-20]. Dostupné z <http://www.mirekw.com/ca/whatis_mcell.html>.

[33] *What is Wireworld?* [online]. [cit. 2013-2-10].
Dostupné z <http://www.quinapalus.com/wires0.html>.

[34] *Wolfram MathWorld* [online]. [cit. 2013-3-2].
Dostupné z <http://mathworld.wolfram.com/Rule110.html>.

Příloha A Formát MCLife

Příloha A byla převzata z literatury [22].

```
#MCell 3.00
#GAME Generations
#RULE 3467/25/6
#SPEED 20
#BOARD 300x300
#WRAP 0
#D
#D The universe of Worms.
#D
#D Discovered by Mirek Wojtowicz
#D 1999.04.08
#L ..DE.DED$.CE.E.DD$BDBABBC$.CACCEC$.A.BD.B$..B3C$..
#L 7.4A$7.3A$7.3CB$6.B.DB.A$6.CECCAC$6.CBBABDB$..
```

Příloha B Formát Life 1.05

```
#Life 1.05
#R 23/3
#D Počáteční populace
********************
*.......*******.**..*
.*.****.*.*******.***
*.......*******.**..*
.*.****.*.*******.***
```

Příloha C Formát Macrocell

Příloha C byla převzata ze souboru *golly-2.4\Patterns\HashLife\puzzle.mc* v aplikaci Golly [15].

```
[M2] (golly 2.0)
#C At 2.3e12 (www.tweedledum.com/rwg/gottspuzz.png) this
#C pattern seems to have regularized: the last twelve bursts
#C from the left formed a geometric progression of
#C trapezoids. But the top two corners of the next
#C "trapezoid" are missing, as were several previous.
#C To see that the trapezoid progression is exactly doubling,
#C click with the up/down scaling cursor on the limit point.
#C Bill Gosper, 27 Feb 2006
$$$$$$$*$
4 0 0 1 0
5 0 0 2 0
6 0 0 3 0
7 0 0 4 0
*$*$
$$....*$.....*.*$......*$
$$*$$.*$.*$.*$
4 6 0 7 8
5 9 0 0 0
6 10 0 0 0
$$$$......**$.....*$
4 0 0 0 12
$$$$***$..*$..*$.*$
4 0 0 14 0
$$$$.......*$$.......*$
.....*$......*$..*...*$...*****$$**$$...*****$
..*...*$......*$.....*$
4 16 17 0 18
5 13 15 0 19
6 0 20 0 0
7 11 21 0 0
8 0 5 0 22
```