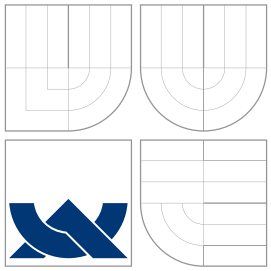
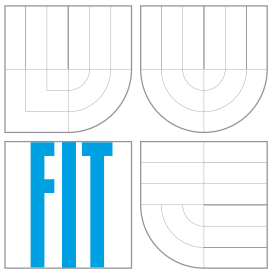


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR CELULÁRNÍCH AUTOMATŮ

CELLULAR AUTOMATA SIMULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — DOMINIK MARTINEK
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. PETR PERINGER
SUPERVISOR

BRNO 2008

Abstrakt

Tato práce popisuje návrh a implementaci simulátoru celulárních automatů. Práce je rozdělena na tři tématické celky. První část obsahuje popis jednotlivých druhů celulárních automatů a jejich použití. Následuje část, ve které je uveden návrh simulátoru celulárních automatů. V poslední části je uveden postup implementace a sada testovacích a ukázkových příkladů.

Klíčová slova

modelování a simulace, celulární automaty, simulátor, GUI

Abstract

This work describes concept and implementation of cellular automata simulator. It is resolved into three thematic parts. First part is devoted to describe many kinds of cellular automata.In second part is concept of simulator. Thirt part contains procedure of implementation and some test and exemplary models.

Keywords

modeling and simulation, cellular automata, simulator, GUI

Citace

Dominik Martinek: Simulátor celulárních automatů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Simulátor celulárních automatů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Dr. Ing. Petra Peringera. Rovněž jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dominik Martinek
9. května 2008

Poděkování

Děkuji svému vedoucímu práce Dr.Ing. Petru Peringerovi za poskytnutí cenných rad a nápadů, které mi pomohly tuto práci vytvořit.

Obsah

Kapitola 1

Úvod

Celulární automaty(dále též CA) tvoří podstatnou část informačních technologií. Jejich vznik a vývoj znamenal významný krok dopředu na celou informatickou oblast napříč všemi jejími obory. CA našly uplatnění v grafice, kde pomáhají vykreslovat náročné scény či v oblasti hardwaru, kde se využívají v aplikacích společně s genetickými algoritmy. Jednou z jejich nejvýznamnější oblastí je však použití při modelování a simulacích. CA totiž dovolují nahlédnout jak do mikroskopických tak makroskopických částí reálného světa a rovněž dovolují nahlížet na samotnou podstatu problému.

Pro pochopení a následné využívání celulárních automatů je však vhodné mít odpovídající simulátor, s jehož pomocí půjde modely vytvářet, upravovat a následně simulovat. Tato práce se zabývá návrhem a implementací takového simulátoru. Rozebírá většinu problémů, které se během tvorby aplikace vyskytují a snaží se najít několik možných řešení a z nich vybrat to nejvhodnější, které bylo nakonec použito.

Práce je rozdělena na tři tématické celky. První část obsahuje popis jednotlivých druhů celulárních automatů a jejich použití. Následuje část, ve které je uveden návrh simulátoru celulárních automatů. V poslední části je uveden postup implementace a sada testovacích a ukázkových příkladů.

Kapitola 2

Celulární automaty

Tato kapitola se věnuje vzniku a následným historickým vývojem CA. Rovněž ukazuje základní motivaci k tomu, proč bychom se měli vlastně CA zabývat. Dále se v této kapitole nachází definice CA a rozdělení automatů do skupin, podle nichž se rozlišují. Informace a skutečnosti uvedené v této kapitole jsem získal především z [5, 2].

2.1 Historie

Historie celulárních automatů se datuje do čtyřicátých let 20. století, kdy přišel John von Neumann s myšlenkou vytvořit stroj, který bude schopen řešit složité komplexní problémy. Jeho inspirací byl mozek, který je schopen mechanizmů jako sebekontrola a sebeopravování. Tehdy přišel s myšlenkou definovat život jako logický proces. Jedním z hlavních Von Neumannových cílů bylo vytvořit systém, který bude dostatečně komplikovaný, aby byl schopen sebereprodukce, čili vytvoření své vlastní kopie.

S možným řešením tohoto problému mu pomohl matematik Stanislaw Ulam, který nalezl inspiraci v přírodě. Každý organismus nebo systém ve vesmíru je totiž složen z určitého počtu diskrétních stavebních prvků, jakými jsou například molekuly, buňky, nebo nižší organizmy. Každý tento prvek je charakterizován svým vnitřím stavem, který bývá většinou vymezen v konečném počtu dosáhnutelných stavů(nabitý, nenabitý, živý, mrtvý,...).

Von Neumann tedy navrhnul systém jako soustavu jednotlivých buněk v diskrétních časových krocích, které fungují jako velmi jednoduché automaty, jenž dokážou vypočítat svůj následující stav. Tento výpočet se řídí podle jednoduchých pravidel, je totožný pro všechny buňky v systému a jedná se de facto o funkci, která má za vstup nejen stav dané buňky v diskrétním čase ale i stavy okolních buněk. Výstupem takovéto funkce je hodnota buňky v následujícím kroku, čili její další vývojový stupeň. Takovéto systémy byly nazvány jako celulární či buněčné automaty.

2.2 První úspěchy

První sebereprodukční celulární automat navržen von Neumannem byl tvořen dvoudimenzionální plochou se čtvercovými buňkami. Každá z těchto buněk mohla dosáhnout jednoho z 29 stavů. Pro evoluční pravidla byla využita hodnota buňky spolu s okolím buňky daným čtyřmi světovými stranami: sever, jih, východ, západ. Tento automat byl schopen posloupností instrukcí vytvořit svou vlastní kopii, která byla schopna tohotéž. Tím se podařilo postavit první stroj, který byl schopen vytvořit jiný stroj stejné úrovně komplexity, což

bylo velmi zajímavé, protože do té doby se očekávalo, že stroj může vytvořit jiný objekt pouze na menší úrovni komplexity, než je on sám. Von Neumannovi se podařilo tímto ukázat, jaký druh komplexity sebereprodukční stroj potřebuje. Později se podařilo sebereprodukční automat zjednodušit pouze na využívání osmi stavů.

Spolu se sebereprodukcí vznikaly otázky, zda se mohou automaty vyvíjet pomocí Darwinovy teorie evoluce. Byly proto vytvářeny automaty, které z jednoduchých pravidel vytvářely složité a komplexní struktury, které dokonce jevily jisté známky „života“. Snad i proto navrhnul v roce 1970 matematik John Conway hru „Life“, která se stala asi nejúspěšnější a nejznámější ukázkou celulárního automatu.

Jeho motivací bylo vymyslet jednoduchá pravidla, která budou sloužit k ovládání složitých a komplexních systémů. Automat tvoří velká plocha podobající se šachovnici s buňkami, které nabývají dvou hodnot: živá nebo mrtvá(1 nebo 0). Dále pak automat obsahuje opravdu jednoduchá pravidla: pokud jsou v okolí mrtvé buňky tři živé buňky, ožije; jsou-li v okolí dvě živé buňky, buňka si ponechává svou hodnotu a nakonec má-li živá buňka jiný počet okolních živých buněk, umírá. Jako okolí buňky se berou buňky v nejbližším okolí, čili sever, jih, východ, západ a druhé nejbližší buňky tj. nejbližší buňky po obou diagonálách.

Zkoumáním bylo objeveno, že rozličné počáteční konfigurace buněk na ploše umožňují vytváření i velmi složitých struktur, která se nekontrolovatelně rozrůstají nebo mizí. Conway z počátku odhadoval, že žádná konfigurace nemůže růst do nekonečna. V průběhu času však byly objeveny i konfigurace, které se takto rozrůstají. Ty byly nazvány jako Slider guns neboli kluzákové děla.

Během pokusů s hrou Life bylo zjištěno a dokázáno, že je co se týče výpočetní síly, ekvivalentní Turingovu stroji. Čili jakýkoliv výpočet, jenž lze provést na jakémkoliv Turingově stroji, lze provést i pomocí hry Life. Hra Life se dodnes používá pro simulace některých komplexních problémů.

2.3 Základní motivace

Celkovou schopnost využití celulárních automatů pro studování a modelování komplexních systémů a fyzikálních jevů shrnuli Tommaso Toffoli a Norman Margolus z MIT:

> „Celulární automaty jsou stylizované syntetické vesmíry, definované jednoduchými pravidly, dosti podobné deskovým hrám. Mají svou vlastní formu hmoty, která se složitě pohybuje ve svém vlastním prostoru a čase. Lze jich vymyslet ohromné množství. Lze je skutečně realizovat a pozorovat, jak se vyvíjejí.“[5]

Toffoli a Margolus se zabývali analogií mezi teorií informatiky a fyzikálními zákony. Přišli na to, že celulární automaty jsou vhodný prostředek pro modelování fyzikálních jevů, které by se jinak modelovaly velmi obtížně.

2.3.1 Dynamika tekutin

Modelování dynamiky tekutin je jedním z typických případů využití celulárních automatů při modelování. Dřívější metody využívaly převážně Navierových–Stokesových rovnic, které se dívaly na problém příliš obecně. Další možností je dívat se na kapalinu z molekulárního hlediska, avšak vzhledem k počtu molekul v objemové jednotce kapaliny je tato varianta téměř výpočetně nekonečná. Celulární automaty nabízejí při pohledu na kapalinu optimální kompromis.

Kapalina je totiž chápána jako soubory jednotlivých částic, které se pohybují po symetrické mřížce a řídí se fyzikálními zákony, kdy se při srážce částic zachovává energie a hybnost. Bylo dokázáno, že takovéto modely s mřížovým plynem jsou skutečně aproximací Navierových–Stokesových rovnic. V mnoha případech tak získáváme skutečně realističtější výsledky. Celulárních automatů se využívá například při modelování směsí ropy a vody, prosakování ropy vápencovými a pískovcovými horninami, výpočtech laminárního proudění, turbulencí a mnoha dalších jevů v kapalinách.

2.3.2 Wolframovy třídy 1D automatů

Stephen Wolfram je matematikem a fyzikem, kterému jednorozměrné celulární automaty poskytly prostředí pro studium vzniku komplexity. Zajímal se především o důsledek lokálních pravidel na globální systém. Šlo o to, že na náhodné uspořádání buněk byla aplikována určitá lokální pravidla a zkoumalo se, jak se bude automat vyvíjet v čase. Výsledkem je rozdělení pravidel do čtyř skupin:

1. automaty s těmito pravidly dospěly po několika krocích do stavu, kdy se řádek buď úplně vyprázdnil nebo úplně zaplnil
2. automaty s těmito pravidly se dostaly do stabilního stavu, kdy se jejich výstup po několika krocích periodicky opakoval a již se neměnil
3. tato pravidla způsobila to, že se automat dával za výstupy chaotické stavy, čili nebyla vidět žádná perioda ani systém ve výstupech
4. v této skupině pravidel se na výstupu automatu nepravidelně objevují a mizí některé periodické obrazce

V tomto ohledu je zajisté nejzajímavější skupina 4 do níž patří i hra Life.

2.3.3 Hardwarové využití CA

Je-li Turingův stroj paradigmatem pro sériové výpočty, pak se lze dívat na celulární automat jako paradigma paralelních výpočtů. Této teze si všimli někteří vědci již v 50. letech minulého století a stala se jim inspirací pro vývoj mnohem rychlejšího a efektivnějšího hardwaru. Margolus společně s Toffolim začli vyvíjet první paralelní počítač CAM (Cellular Automata Machine), na kterém běžela simulace celulárního automatu mnohem rychleji než na tehdejším superpočítači Cray.

V poslední době již existuje počítač CAM–8, který je integrován jako zásuvná karta do počítače. Pár těchto karet dokáže spolehlivě předstihnout superpočítač a jsou na nich modelovány jevy, jako vypařování kapky, lesní požáry, nebo stékání kapky po různých typech povrchu. Dnes se však jako nejvhodnější hardware pro implementaci celulárních automatů ukazuje FPGA (Field Programmable Gate Array), čili elektrickým polem programovatelné hradlové pole. Takovýchto obvodů se využívá pro evoluci genomů, čili pro modelování živých organismů, které slouží jako inspirace pro genetické algoritmy(bližší informace viz [3]).

2.3.4 Praktické využití CA

Celulární automaty se ukázaly jako silný prostředek pro modelování rozličných fyzikálních jevů. Díky tomu se objevují hodně v oblastech reálného života. Bývají například využívány ve stavebnictví, kde se pomocí nich modelují hydratace cementu a tím se vytvářejí nové a

lepší hydratační směsi. Tyto směsi zajišťují správnou mikroskopickou strukturu betonu a tím dosahují betony velmi dobrých mechanických vlastností. Dále je pomocí CA modelováno tuhnutí cementu a tím se zajišťuje, aby cement vytvrdnul tam, kde má. Uplatnění nachází i v metalurgii, kdy napomáhá najít lepší technologie žíhání materiálu.

CA nenacházejí uplatnění jen v mikroskopických oblastech, ale lze je uplatnit třeba při modelování dopravy, kde se jejich pomocí modelují dopravní zácpy, křižovatky, hledají se optimální cesty a rychlosti aj. Svou roli našly CA i v kryptografii, modelování šíření(požáru, znečištění, nemoci,...) a mnoha jiných oblastech. Bližší informace je možné nalézt v [2].

2.4 Definice CA

Celulární automat [1] je dán:

- pravidelnou sítí buněk pokrývající část d–dimenzionálního prostoru, anglicky *lattice*
- sadou $\mathbf{\Phi}(\vec{r},t) = \{\Phi_1(\vec{r},t), \Phi_2(\vec{r},t), \ldots, \Phi_m(\vec{r},t)\}$ stavů, které mohou nabývat hodnot z oboru reálných čísel, popisující pro každé místo $\vec{r}$ v síti jeho lokální stav v čase $t = 0, 1, 2, \ldots$
- pravidly (*rules*) $\mathbf{R} = \{R_1, R_2, \ldots, R_m\}$, která specifikují vývoj stavů $\mathbf{\Phi}(\vec{r},t)$ následujícím způsobem

$$\Phi_j(\vec{r},t) = R_j\left(\mathbf{\Phi}(\vec{r},t), \mathbf{\Phi}(\vec{r}+\vec{\delta}_1,t), \mathbf{\Phi}(\vec{r}+\vec{\delta}_2,t), \ldots, \mathbf{\Phi}(\vec{r}+\vec{\delta}_q,t)\right)$$

  kde $\vec{r}+\vec{\delta}_k$ popisuje okolí buňky $\vec{r}$. Pravidla bývají též nazývána lokálními přechodovými funkcemi.

- samotným okolím buňky (*neighborhood*)

Pro celulární automat jsou typické následující tři vlastnosti:

1. **paralelizmus** – všechny buňky jsou zpracovávány v jeden časový okamžik
2. **lokálnost** – pro každou buňku je důležitý pouze její vlastní stav a stavy buněk v jejím okolí
3. **homogenita** – na všechny buňky jsou aplikována stejná pravidla pro vývoj následujících stavů

2.5 Druhy prostředí a okolí

Jak již bylo zmíněno, pro CA je typická lokálnost. Pro každý typ úlohy řešené pomocí CA se však může hodit jiná strukruta mřížky oddělující buňky a pro každou úlohu může být typické určité okolí buněk, které se bude vyhodnocovat v přechodových funkcích. Nicméně existuje několik standardních typů struktur prostředí i okolí.

Co se týče struktur mřížky záleží samozřejmě na dimenzi prostoru. U 1D automatů je samozřejmostí používaní jedné linie buněk zobrazovaných pomocí čtverců, u 3D automatů je prostor pravidelně rozdělen na rovnoměrné krychle. U 2D automatů se však objevuje více alternativ reprezentace plochy.

Dvourozměrnou plochu jde beze zbytku rozdělit pouze pomocí pravidelných trojúhelníků, čtverců či pravidelných šestiúhelníků. Zejména se používají poslední dva jmenované, čili

čtverce a šestiúhelníky. Každá tato reprezentace má své výhody i nevýhody. Čtvercová plocha má podstatně jednodušší systém souřadnic a lze tedy každou buňkou jednoduše popsat souřadnicí x a y. Šestiúhelníková plocha se vyznačuje tím, že středy sousedních buněk jsou navzájem stejně vzdálené. Navíc jsou šestiúhelníkové buňky svým tvarem blíž skutečným organizmům, které mají často právě tento tvar. Při návrhu modelu, je tudíž nutné přemýšlet nad těmito skutečnostmi.

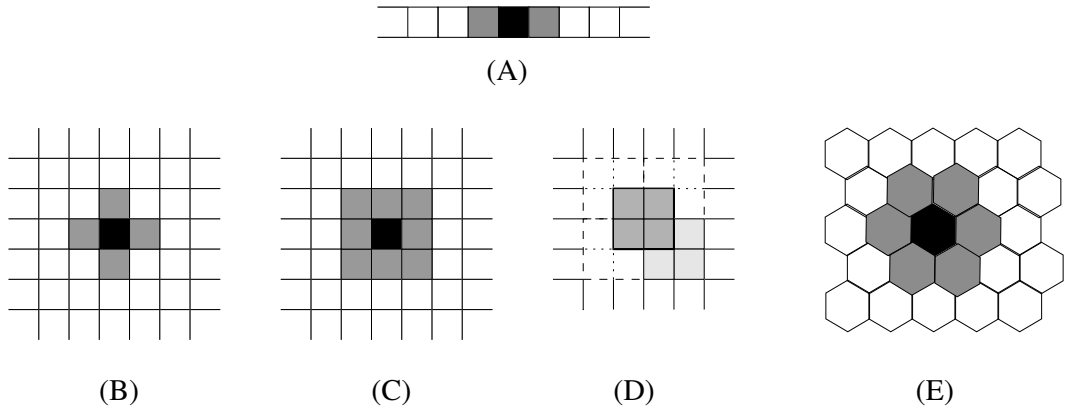
Oblast okolí buněk je neméně rozmanitá. Rovněž existuje několik typických okolí, která jsou mnohdy používána. Co se týče 1D automatů, tak je bezesporu nejpoužívanějším radiální okolí, čili kolem buňky je bráno okolí s poloměrem r na levou i pravou stranu. V mnoha případech je $r = 1$, což použil ve své práci i Stephen Wolfram.

U 2D automatů jsou nejtypičtějšími okolími při čtvercovém uspořádání okolí von Neumanovo, Moorovo a Margolusovo(je možné vidět na obr.2.1). Zde je nejčastěji používáno von Neumanovo okolí, protože poskytuje analogii s geografickými směry, jakými jsou sever, jih, východ a západ. Moorovo okolí vzniklo tak, že byly k von Neumanovu přidány ještě směry po obou diagonálách, tedy severo–východ, jiho–východ, jiho–západ a severo–západ. Toto okolí už popisuje všechny buňky v okolí které se dotýkají zkoumané buňky.

Ne vždy nám však postačuje pouze nejbližší okolí. V takovýchto případech nám slouží obdoba radiálního okolí, kterému se říká rozšířené Moorovo okolí(extended Moore neighborhood), kdy je poloměr r větší než 1.

Dalším zajímavým okolím je Margolusovo. De facto se nejedná o okolí nějaké buňky, jak ho známe, ale o to, že plocha je rozdělena na čtverce o velikosti 2 x 2 buňky. Tyto čtverce se vzájemně překrývají. Tohoto okolí lze výhodně použít třeba při modelování sypání písku, či obtékání nějaké překážky.

V mnoha případech však potřebujeme jiné, naprosto specifické okolí, proto je vhodné si jej navrhnout. Máme-li například model cesty, kdy buňky představují auta, zajímá nás především dlouhý úzký pás před námi, který představuje viditelnou cestu a krátký úsek za námi, jenž tvoří pohled do zpětného zrcátka.



Obrázek 2.1: (A)–radiální okolí u 1D automatu s $r = 1$; (B)–von Neumanovo okolí; (C)–Moorovo okolí s $r = 1$; (D)–Margolusovo okolí; (E)–šestiokolí u šestiúhelníkové mřížky

2.6 Typy pravidel pro CA

Pravidla, podle kterých se řídí vývoj buněk v čase jsou nejdůležitějším prvkem CA. Ony samy popisují chování systému nebo modelu. Samotná pravidla jsou taktéž rozdělena do několika tříd podle jejich charakteristiky. Základním rozdělením je rozdělení na deterministická a pravděpodobnostní pravidla. Podle těchto pravidel jsou rozděleny i automaty, tedy na deterministické a pravděpodobnostní automaty. Třídě deterministických automatů,čili těm CA, u kterých je následující stav odvozen ze stavů předchozích, se budu věnovat později.

2.6.1 Pravděpodobnostní pravidla a automaty

Jak již název pravděpodobnostní pravidla napovídá, bude se v těchto pravidlech pracovat s pravděpodobností. V takovýchto pravidlech se změna stavu buňky nemusí řídit stavem jejího okolí, ale může se třeba vybrat náhodně jeden z možných stavů s pravděpodobností p. V jiných případech může třeba pravděpodobnost potlačit provedení řádného pravidla a buňka zůstane v původním stavu.

Takto stavěná pravidla vnášejí do našeho modelu notnou dávku komplexity, protože tím mohou být simulovány různé přírodní abnormality jako „zásah vyšší moci“, genetická mutace aj. Typickým příkladem využití pravděpodobnostních pravidel budiž model hořícího lesa. Zde jsou pravidla pro šíření ohně mezi stromy, dále se ze spálené země s pravděpodobností p rodí nové stromy, ale s pravděpodobností f začínají zdravé stromy hořet(zásah bleskem, neopatrní trampové, ...).

V tomto modelu se zejména sleduje poměr f/p, který udává, jak se bude schopen les sám vypořádat s požárem. Tento model šíření požáru lze různě modifikovat například přidáním pravidla, že strom se od okolních stromů zapálí pouze s pravděpodobností $1 - q$, kde pravděpodobnost q může určovat například odolnost některých druhů stromů vůči zapálení. Dokonce může být tato pravděpodobnost i jiné pravděpodobnosti modifikovány funkcemi udávajícími další faktory určující meteorologické podmínky, zásah člověka aj.

Vidíme tedy, že pouhým přidáním několika jevů s určitou pravděpodobností se z jednoduchého modelu stal poměrně komplexní model, který již lze prohlásit za odpovídající model skutečného světa. Tímto modelem je možné třeba simulovat opravdové lesní požáry a navrhovat tvary a složení lesa, u kterých je největší šance sebeshášení. Lze prohlásit, že pravděpodobnost vnáší do modelu část reálného světa.

2.6.2 Totalistická pravidla a automaty

Tato pravidla by se dala označit také jako sumární. Principem těchto pravidel je průzkum okolí, při němž se spočítáná suma hodnot okolí nebo nebo počet výskytů některých hodnot. Z této sumy jsou pak vyvozovány důsledky pro vývoj buňky.

Mezi takovéto automaty lze zařadit i výše zmíněnou hru Life, kde se vývoj buňky rozhoduje podle počtu živých nebo mrtvých buněk v okolí. Využití sumy okolních buněk může být využito třeba i u modelu šíření teploty. Buňka si může vypočítat průměrnou teplotu okolí a podle své aktuální teploty, vlastností materiálu a samozřejmě termických zákonů, provést přírůstek nebo úbytek své vlastní teploty tak, aby se snažila o vyrovnání teplot.

2.6.3 Reverzibilní pravidla a automaty

Reverzibilita je velmi zvláštní vlastnost u reverzibilních automatů. Reverzibilita říká, že informace se nemůže v uzavřeném systému objevit, ale nemůže se ani sama vytratit. Podle [2] se jedná o takzvaný zákon zachování informace. U CA tato vlastnost říká, že z každého stavu v čase t, ve kterém se automat může nacházet, musí jít zjistit stav, ve kterém se automat nacházel v čase $t-1$.

U nereverzibilních automatů většinou platí, že se systém po určité době zacyklí a začne oscilovat. U reverzibilních automatů ale reverzibilita zaručuje, že systém se tak snadno do oscilace nedostane. Aby se dostal do nějakého stavu, ve kterém se již nacházel, musí zároveň projít stavem, ze kterého začínal. Toto zaručuje, že reverzibilní automat bude mít dostatečně velkou smyčku a tedy i čas pro evoluci.

Chceme-li tedy zkoumat model, u něhož předpokládáme dostatečně dlouhý čas simulace, je vhodné pro to použít právě automat s reverzibilními pravidly. Pro reverzibilní automaty se převážně využívá Margolusovo okolí. Pravidla musí být navržená tak, aby se při přechodu do následujícího stavu neměnil počet buněk. Tím lze zaručit, že informace nebudou ztraceny. Jsou–li navíc reverzibilní pravidla, je reverzibilní i celý automat. Těchto reverzibilních automatů lze využít například při simulaci šíření rázové vlny.

2.6.4 Wolframova pravidla

Stephen Wolfram se během svého zkoumání zaměřil na jednorozměrné automaty. Používal radiální okolí a pouze dvě hodnoty buněk a to 1 a 0. Hodnota buňky společně se svým okolím tak můžou tvořit osm možných kombinací. Bereme-li definici pravidel jako informaci, která udává výstup dané konfigurace buňky a jejího okolí, získáme tím 2^8 tedy 256 možných pravidel. Z tohoto důvodu se všechna tato pravidla očíslovala od 0 do 255. Objeví-li se nám například pravidlo 40, znamená to rozložení $40_{10} = 00101000_2$. Tato pravidla byla posléze aplikována na náhodné počáteční rozložení automatu a byl zkoumán jejich vývin v čase. Experimentálně tak byla pravidla rozdělena do výše zmíněných čtyř skupin automatů. Bližší výstupy a chování jednotlivých pravidel lze naléz v [6].

2.6.5 Další pravidla

CA poskytují velké pole pro nepřeberné množství různých pravidel. Velmi často vypadají pravidla jako funkce, které provádí z okolí buňky výpočet pomocí matematických operátorů nebo jiných matematických funkcí($\sin(), \cos(), \min, \max, \dots$). Máme–li buňky jenom s binárními hodnotami, velmi často se používají operátory Booleovy algebry(AND, OR, XOR,...). Další možností jak stavět pravidla je ta, kdy se buňka bude podle své aktuální hodnoty dívat v daném směru a na určitého souseda. Těchto pravidel se využívá zejména v dopravních modelech, kdy hodnota buňky, udávající rychlost, rozhoduje, do jaké vzdálenosti se bude buňka dívat. Dalším tímto využitím je třeba možnost, že budeme modelovat šíření určitým směrem. Některá výše zmíněná pravidla lze dokonce kombinovat a tím získat potřebný model.

Kapitola 3

Návrh simulátoru

Zadání hovoří pouze o vytvoření simulátoru celulárních automatů s grafickým uživatelským rozhraním. Toto zadání je však nutné blíže specifikovat. Tato specifikace hovoří o vytvoření programu, kterým bude možno simulovat 1D a 2D automaty s čtvercovým uspořádáním buněk. Simulátor by měl pracovat s takzvaným nekonečným polem. Samotný celulární automat má být řešen jako knihovna, která bude s uživatelským rozhraním komunikovat. Uživatelským rozhraním musí jít provádět základní ovládání, jakým je nastavení parametrů modelu, definování jeho základního rozložení, spuštění, krokování a zastavení simulace. Tato kapitola se tedy zabývá analýzou a návrhem řešení takto specifikovaného problému.

3.1 Analýza buněčné plochy

Čtvercová plocha buněk umožňuje, aby byly jednotlivé buňky velmi jednoduše identifikovány. Tato identifikace buňky může být řešena podle její souřadnice, čili podle pozice řádku a sloupce buňky. Máme-li vyřešenou identifikaci buňky, je nutné se zamyslet nad tím, že konkrétní buňka bude často vyhledávána ať za účelem získání její hodnoty, nebo za účelem zapsání hodnoty nové. Vyhledání buňky tedy musí být velmi rychlé, aby časté vyhledávání příliš nebrzdilo a nezpomalovalo celý systém. Nachází se zde několik možných řešení implementace plochy:

- **matice** – toto řesení spočívá v implementaci plochy jako klasické dvojrozměrné matice. Buňka je posléze vyhledávána přímo podle svých souřadnic, které určují ukazatele v poli. Tato možnost se vyznačuje tím, že vyhledávání dané buňky probíhá s konstantní složitostí, čili je stejně rychlé pro jakýkoliv počet buněk a vyhledávání je nejrychlejší. Velikou nevýhodou tohoto řešení je, že pro obrovská pole je nutné alokovat velké množství paměti. Vzniká tím velká paměťová náročnost.

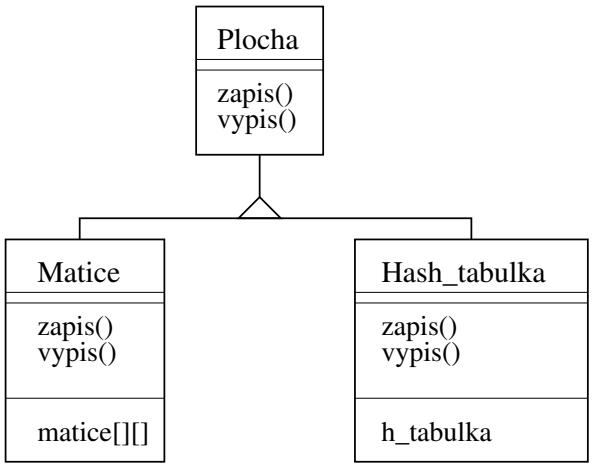
- **pole s rozptýlenými hodnotami** – někdy také nazývané „hashpole“. toto řešení předpokládá, že souřadnice buňky budou zpracovány pomocí hashovací funkce. Touto funkcí se získá index, který ukazuje na určitou pozici v poli lineárních seznamů. Při zápisu se prohledává seznam sekvenčně, již podle skutečných souřadnic, a až když nejsou souřadnice nalezeny, tak se zapíší na konec seznamu. Vyhledávání opět probíhá sekvenčně. Toto řešení poskytuje cenný námět a to ten, že v takovéto tabulce nemusí být uloženy nebo alokovány všechny buňky, ale jenom ty které mají jinou, než implicitní hodnotu. Má–li být posléze na hodnotu některé buňky implicitní hodnota, je tato buňka ze seznamu vymazána. Nevýhodou tohoto řešení je, že při určité konstalaci

buněk a nevhodné hashovací funkci se může stát, že bude většina buněk uložena v jednom lineárním seznamu, který se bude prohledávat velice dlouho. Vhodnější řešení je však použít následující řešení

- **šablona map<>** – C++ poskytuje ve své standardní knihovně STL[4] několik velmi užitečných šablon. Mezi tyto šablony patří pár–`pair<>` a asociativní pole–`map<>`. Pár umožní spojit dvě hodnoty v jeden objekt, lze ho tedy využít pro spárování x a y souřadnice konkrétní buňky. Tento objekt je využit v asociativním poli, které je vylepšenou implementací hash tabulky. Umožňuje vyhledávání podle hodnot souřadnic uložených v páru. Vnitřní implementace této šablony však provádí vyhledávání založené na binárním stromech a šablona tím vykazuje logaritmickou náročnost vyhledávání. Je tedy využita předchozí idea, ale je šablonou optimalizována. Jednou z nevýhod tohoto řešení je však fakt, že při přepsání buňky, která je již uložena v poli, je nutné tuto buňku nejdříve odstranit, protože přepsání hodnoty šablona neumožňuje.

- **kvadrantový strom** – dovoluje, aby byla plocha rozdělena na kvadranty, které se mohou rekurzivně dále čtvrtit, dokud nebude dosaženo elementární plochy, která již bude implementována maticí. Nebyla–li by tato plocha obsazena alespoň jednou buňkou, mohla by být odalokována do doby, kdy by na ni mělo být opět zapsáno. Dále pokud by určité plochy měly totožné hodnoty mohly by být nahrazeny pouze touto hodnotou a tím opět urychlit vyhledávání. Toto řešení se vyznačuje rovněž logaritmickou složitostí vyhledávání. Její zásadní nevýhodou je ale nutná vyšší režijní činnost, spojená s alokováním a dealokováním paměti a přeuspořádáním kořenů stromu, a implementační obtížnost.

Pro využití v simulátoru se mi zdá nejvhodnější souběžné použití variant matice a asociativního pole pomocí šablony `map<>`. Je nutné ale nalézt odpovídající hranici, přes kterou začíná být paměťová náročnost matice příliš velká na úkor vyhledávací rychlosti. Pro větší pole by bylo vhodnější použít mapu i když má horší vyhledávací náročnost.



Obrázek 3.1: Objektový návrh tříd plochy

Na obrázku 3.1 lze vidět objektový návrh tříd pro plochu buněk. Základem je třída `Plocha`, která obsahuje dvě virtuální metody – `zapis()` a `vypis()`. Tato třída je předkem tříd `Matice` a `Hash_tabulka`. Tyto třídy již implementují zděděné metody. Třída Matice

využívá pro uložení hodnot vnitřně implementovanou dvourozměrnou matici `matice[][]`. `Hash_tabulka` naopak využívá vnitřní hashovací tabulku `h_tabulka`.

3.2 Návrh řešení pro zadávání pravidel CA

Jak je patrno z části 2.6, celulární automaty umožňují použití nepřeberného množství způsobů definice různých pravidel. Aby byl simulátor dostatečně obecný, měl by umět interpretovat jejich většinu nebo alespoň jejich značnou část. Nachází se pro to několik možných řešení.

Prvním z nich je vytvoření jednoduchého analyzátoru a interpretu pravidel. Toto řešení by předpokládalo vytvoření vlastního jazyka a jeho syntaxe. Dále by si toto řešení vyžadovalo vytvoření syntaktického a sémantického analyzátoru, který by kontroloval syntaxi a sémantiku uživatelem zadaných pravidel. V poslední řadě je nutné jestě vytvoření interpretu, který by pravidla přeměnil na funkce, které by se aplikovaly při evolučních krocích automatu. Toto řešení je ale implementačně velice náročné a zdlouhavé. Existuje však lepší řešení, které přenechává syntaktickou a část sémantické analýzy někomu jinému.

Jedná se o využití dynamicky přilinkovaných knihoven. Pravidla se zapíší ve formě zdrojového kódu v C/C++, který se posléze přeloží kompilátorem, který provede syntaktickou a sémantickou analýzu. Získáme tím dynamickou knihovnu, kterou pomocí vhodného rozhraní nahrajeme do programu během jeho běhu.

3.3 Návrh výpočetního jádra simulátoru

Jak již říká definice v kapitole 2.4, pro celulární automat je význačný pojem paralelizmus. Tento pojem nám říká, že buňky se zpracovávají v jeden okamžik, tedy paralelně. Valná většina osobních počítačů ale funguje na sériovém mechanizmu. Paralelní zpracování je tedy nutné nějakým způsobem simulovat. Buňky se musí zpracovat sériově, čili jedna po druhé. S tím ale vzniká jeden problém. Na ploše musí mít všechny buňky pro výpočet hodnoty v čase $t + 1$, hodnotu v čase t. Na celé ploše tedy musí být buňky, které mají hodnotu ve stejném časovém okamžiku. Byla–li by buňka zpracována a její nová hodnota zapsána ihned zpět do pole, byla by porušena tato podmínka. Takový systém by již nefungoval správně, protože by se pro výpočtu kombinovaly hodnoty ve dvou časových okamžicích, což by způsobilo nesmyslné výsledky. Musíme tedy zajistit, aby se hodnoty v růzých časech nemíchaly. Existuje opět několik způsobů, jak se s tím vypořádat.

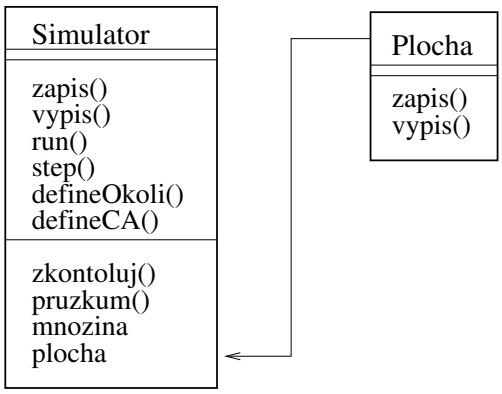
Nejtriviálnější způsobem je vytvoření kopie buněčné plochy. V praxi tento systém funguje tak, že máme dvě identické plochy a to pro časy t a $t+1$. Sekvenčně procházíme buňky v aktuálním čase t a aplikujeme na ně pravidla automatu. Jednotlivé výsledky ukládáme na identické pozice, avšak na plochu v čase $t + 1$. Projdeme–li všechny buňky, prohodíme časové označení obou ploch. Tím prohlásíme plochu $t + 1$ za výsledek jednoho kroku simulace. Tato metoda má opět své klady a zápory. Ke kladům patří rychlost, protože jedinnou potřebnou režijní funkcí je prohození časových označení ploch. Záporem této metody je paměťová náročnost, protože místo jedné plochy musíme v simulátoru udržovat dvě stejně velké.

Druhou možností je využití menšího odkládacího prostoru, do kterého budeme odkládat jenom hodnoty buněk, které vlivem pravidel změnily svou hodnotu. Po průchodu všech buňek vyjmeme hodnoty z odkládacího prostoru a jejich hodnoty zapíšeme zpět do pole. Díky tomu můžeme výrazně ušetřit místa v paměti. Nevýhodou je však vyšší režie spojená

se zapisováním hodnot zpět do pole. Přesto se osobně více klaním k této možnosti, protože pro obrovské rozměry plochy tím lze velké množství místa ušetřit.

Další optimalizací výpočtu je způsob procházení buněk. Triviální postup, kdy procházíme všechny buňky, jednu po druhé, má u velkých ploch příliš velkou časovou náročnost a je velice neefektivní, máme–li na ploše pouze několik buněk, které pracují. V takovém případě se prochází většinou buněk naprosto zbytečně. Buňka se totiž stává aktivní, nachází–li se v jejím akčním okolí aktivní jiná buňka. Aktivní buňkou se stává také ta buňka, která ve dvou po sobě jdoucích časových okamžicích změní svou hodnotu.

Prakticky se tato vlastnost dá využít tak, že implementujeme dvě množiny. V první množině jsou uloženy souřadnice buněk, které budeme procházet při současném kroku simulace. Tyto buňky v množině procházíme jednu po druhé a aplikujeme na ně daná pravidla. Změní–li buňka vlivem pravidla svou hodnotu, je prohlášena za aktivní a je ona i se svým okolím vložena do druhé množiny. Ta představuje množinu buněk, které jsou aktivní a je u nich pravděpodobné, že se v následujícím kroku změní. Je–li první množina vyprázdněna, znamená to, že jsme prošli její všechny buňky a provedli jsme tak jeden krok simulace. V tomto případě provedeme výměnu významů jednotlivých množin.



Obrázek 3.2: Objektový návrh výpočetního jádra simulátoru

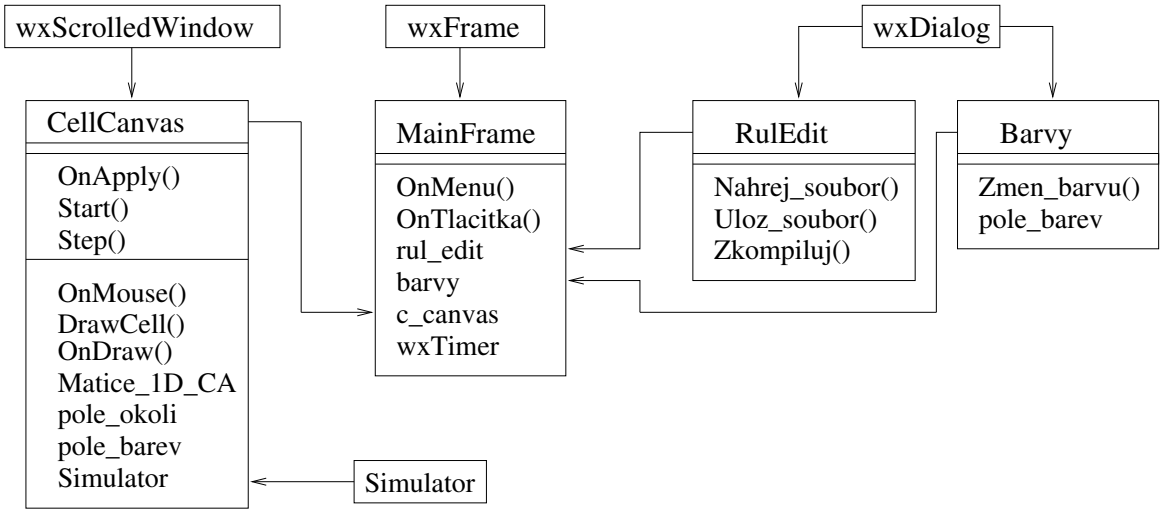
Na obrázku 3.2 vidíme návrh třídy Simulator. Veřejnými metodami, kterými probíhá komunikace s grafickým rozhraním, jsou metody `zapis()` a `vypis()`, kterými probíhá ukládání a vypisování buněk z plochy. Dále obsahuje tato třída veřejné metody `defineCA()` a `defineOkolí()`, kterými jsou nastavovány důležité vlastnosti CA, jakými jsou rozměry, rozsah hodnot a okolí. Nejdůležitějšími metodami CA jsou metody `run()` a `step()`. `Run()` slouží pro první spuštění simulačního modelu, pomocí metody `step()` je tento model krokován. Jako privátní složky této metody jsou uvedeny podpůrné metody, které jsou využívány při simulaci. Metoda `pruzkum()` provede každé buňce průzkum jejího akčního okolí. Výsledek se poté zpracuje metodou `zkontoluj()`, která na danou buňku aplikuje přechodová pravidla. Dále je zde implementován objekt `plocha`, který je instancí třídy `Plocha`(viz kapitola 3.1) a několik množin `mnozina`, které slouží pro ukládání souřadnic buněk, které budeme procházet v současném a následném kroku.

3.4 Návrh grafického rozhraní

Chceme-li při simulaci CA sledovat jeho vývoj, je vhodné k tomu použít grafické uživatelské rozhraní – GUI. Toto rozhraní by mělo poskytovat většinu zázemí pro tvůrce simulačního

modelu. Pomodí GUI by mělo jít v prvé řadě definovat vlastnosti plochy pro následnou simulaci. Uživatel by měl být schopen určit si její dimenzi a velikost, na níž bude simulace probíhat. Dále by měl nastavit vlastnosti buněk, respektive jejich hodnot. Důležitými údaji jsou minimální a maximální hodnota buňky, (min, max) a dále implicitní hodnota, kterou bude po inicializaci pokryto celé pole. Implicitní hodnota musí být z intervalu $< min, max >$ a měla by určovat hodnotu, která bude většinově zastoupena na ploše.

Definujeme–li všechny předchozí hodnoty, měla by být uživatelem provedena samotná definice CA, tedy definice okolí, definice přechodových pravidel a nastavení počátečního rozložení buněk. Pro definici okolí a rozložení buněk je vhodné použít kreslící plochu, na kterou tyto hodnoty vyneseme pomocí myši. Pro definici pravidel je vhodné použít vlastní dialogové okno, v němž půjde vytvářet nové knihovny pravidel, nahrávat a editovat již existující a rovněž provézt překlad nových, či editovaných pravidel. Nezbytnou součástí grafického rozhraní jsou samozřejmě ovládací prvky simulátoru, jakými jsou spuštění simulace, její zastavení a krokování jednotlivých kroků. Plochou buněk by mělo jít rolovat v horizontálním i vertikálním směru, jelikož větší pole se nemusejí vměstnat do základního rozměru, poskytovaného obrazovkou.



Obrázek 3.3: Objektový návrh tříd grafického rozhraní

Na obrázku 3.3 je vidět objektový návrh tříd grafického uživatelského rozhraní. Hlavní třídou celé aplikace je třída `MainFrame`, která je dědicem knihovní třídy `wxFrame`. Objekt této třídy představuje hlavní okno aplikace, které se uživateli po spuštění ukáže. Obsahuje tedy metody na obsluhu menu a různých tlačítek, která jsou také součástí této třídy.Dalším velmi důležitým prvkem třídy `MainFrame` je objekt `c_canvas`, který je instancí třídy `CellCanvas`.

Třída `CellCanvas` je dědicem knihovní třídy `wxScrolledWindow` a stará se převážně o správu kreslící plochy, na kterou jsou vykreslovány buňky. Využití třídy `wxScrolledWindow` umožňuje jednodušší ovládání scrollbarů, neboli posuvných proužků. Třída `CellCanvas` rovněž obstárává veškerou komunikaci se simulátorem, jelikož objekt třídy `Simulator`(viz kapitola 3.3) je součástí této třídy. Spojení hlavního okna s tímto objektem obstarávají metody `OnApply()`, `Start()` a `Step()`. Dále tato metoda obsahuje své privátní metody jako `OnPaint()`, která se stará o překreslování plochy při jakékoliv změně, dále metoda

`DrawCell()`, která zajišťuje vykreslení jednotlivé buňky a nakonec metoda `OnMouse()`, pomocí níž se zpracovávají události vyslané myší. Dále jsou součástí této třídy prvky jako `Matice_1D_CA`, která slouží jako pomocná matice při vykreslování jednodimenzionálních automatů, dále pak `pole_okoli`, do kterého se ukládají souřadnice okolních buněk při zadávání uživatelem a nakonec `pole_barev`, ve kterém jsou uloženy hodnoty barev odpovídající jednotlivým hodnotám buněk.

Aplikace má rovněž dvě dialogové okna, která jsou dědici knihovní třídy `wxDialog`. Prvním z nich je třída `RulEdit`, jejíž dialogové okno slouží k zadávání pravidel. Obsahuje metody jako `Nahrej_soubor()` a `Uloz_soubor()`, které slouží k uložení a nahrání souborů. Dále obsahuje metodu `Zkompiluj()`, která provede přeložení souboru a vytvoření dynamické knihovny. Druhou třídou pro dialogové okno je třída `Barvy`, pomocí níž lze nastavit barvy v poli `pole_barev`, které budou odpovídat jednotlivým hodnotám.

Kapitola 4

Implementace a testování

Simulátor byl implementován jazykem C++ v systému Linux s využitím grafické knihovny wxWidgets 2.8. Tato kapitola se zabývá samotnou implementací simulátoru, která byla tvořena podle předchozích návrhů. Na závěr jsou uvedeny příklady, kterými byl systém testován.

4.1 Počáteční inicializaze automatu

Před samotnou simulací je potřeba celý automat a všechny důležité proměnné správně inicializovat. O tuto funkci se starají tři metody třídy `Simulator` a to `defineCA()`, `defineOkoli()` a `run()`.

Jako první musí být zavolána metoda `defineCA()`. Tato metoda zajistí uložení důležitých proměnných, jakými jsou rozměry pole a minimální, maximální a implicitní hodnota buněk. Tyto hodnoty, jsou zapsány do privátních proměnných automatu. Dále tato metoda zajistí vytvoření plochy, do které se budou buňky nahrávat. Díky polymorfizmu se nabízejí dvě možné třídy, jejichž objekt může představovat plochu. Rozhodnutí, kterou z nich vybrat, závisí na počtu buněk, které bude plocha obsahovat. Jako zlomovou jsem určil hodnotu 16384 buněk, čili velikost pole 128×128 buněk. Je-li součin rozměrů pole menší než tato hodnota, je pro pole zvolena třída `Matice`. V opačném případě je zvolena třída `Hash_tabulka`. Dále je podle minimálních a maximálních hodnot vytvořeno pole `statistika`, které bude představovat počty jednotlivých hodnot buněk.

Druhou důležitou metodou simulátoru je metoda `defineOkoli()`. Tato metoda přijímá od grafického rozhraní jako své argumenty počet buněk tvořící okolí a pole obsahující souřadnice buněk tvořící okolí buňky. Tyto souřadnice se vztahují k buňce na pozici (0, 0). Jedná se tedy o absolutní posuny souřadnic a přičteme-li tyto hodnoty k souřadnici buňky kterou právě zkoumáme získáme tím skutečné buňky, které tvoří její okolí. Souřadnice v tomto poli jsou seřazeny podle své pozice vůči základní buňce a to zleva doprava a shora dolů. Toto pole se v této metodě zkopíruje do vlastního pole automatu. Dále je zkopírována do vnitřní proměnné hodnota velikosti okolí. Následně je ještě vytvořeno pole, do kterého budou při průzkumu okolí ukládány hodnoty jednotlivých buněk.

Poslední metodou sloužící k nastavení počátečních hodnot je metoda `run()`. Tato metoda slouží pro poslední nastavení před samotným spuštěním simulace. V prvé řadě se v této metodě nahraje dynamická knihovna s pravidly, čemuž je věnována samostatná podkapitola 4.2. Následnou funkcí této metody je průchod celým polem, při kterém se buňky s neimplicitní hodnotou uloží spolu se svým okolím do pomocné množiny, z níž se budou

vybírat buňky ke zpracování během prvního kroku simulace. V poslední řadě tato metoda zjistí početní zastoupení jednotlivých hodnot buněk na ploše. Během průchodu polem se vždy inkrementuje hodnota, jejíž pozice je v poli `statistika` určená hodnotou buňky.

4.2 Připojení dynamické knihovny s pravidly

Kvůli univerzálnosti simulátoru jsem se rozhodl pro řešení zadávání pravidel, ve kterém jsou pravidla psaná formou zdrojového kódu v jazyce C. Pro tuto metodu je však nutné připojení dynamických knihoven k programu za jeho běhu. Uživatel totiž napíše pravidla ve zdrojovém textu, který se následně přeloží tak, aby vznikla dynamická knihovna.

Další úloha již leží na programu, který chce takovouto knihovnu nahrát a používat. K tomu nám slouží funkce z rozhraní `<dlfcn.h>`. Řešení, které jsem použil spočívá v tom, že jednou z položek třídy Simulator je ukazatel na funci vracející hodnotu typu `int` se dvěma parametry typu `int` a jedním parametrem typu pole integerů. Zapsáno tedy jako:

```
int (*Zkontroluj)(int, int, int*);
```

Jedná se o přesně takový ukazatel na funkci, jaká je používána v dynamické knihovně. Spojení našeho ukazatele na funkci s funkcí v dynamické knihovně se děje pomocí funkcí `dlopen()` a `dlsym()`. Funkce `dlopen()` vytvoří nahraje dynamickou knihovnu a vytvoří handler, který na tuto knihovnu ukazuje. Z toho handleru je poté pomocí funkce `dlsym()` extrahována naše funkce, jejíž ukazatel je nahrán na adresu našeho ukazatele na funkci.

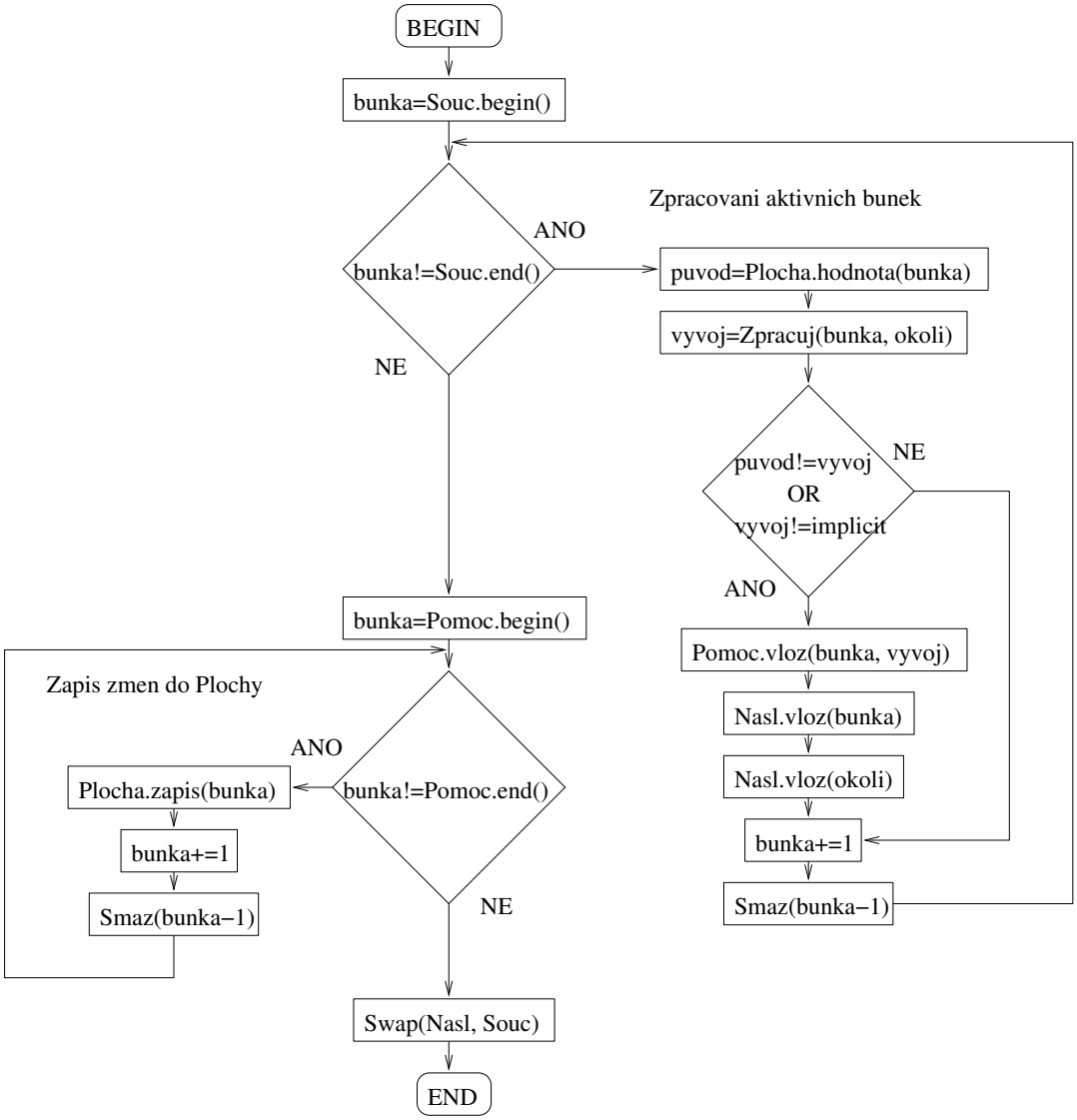
4.3 Implementace metody step()

Obrázek 4.1 popisuje algoritmus, kterým je implementována metoda `step()`. Tato metoda představuje jeden krok simulace.

Jsou využity celkem tři množiny a to `Souc`, `Nasl` a `Pomoc`. Tyto množiny jsou vytvořeny pomocí šablony `map<>`. Jsou v nich uloženy souřadnice buňky, zapouzdřené pomocí šablony `pair<>`, a hodnoty dané buňky. Množiny `Souc` a `Nasl` slouží k udržování informace o aktivních buňkách, které se nacházejí na ploše a bude je nutné zpracovat v současném kroku a k vytváření plánu, podle kterého se budou buňky zpracovávat v kroku následujícím.

Podle množiny `Souc` se tedy vybírají postupně buňky, které se zpracovávají. Je-li vyhodnocená hodnota odlišná od původní nebo pokud není stejná jako hodnota implicitní, je souřadnice této buňky spolu se souřadnicemi okolních buněk uložena v množině `Nasl`. Třetí množina – `Pomoc` slouží jako pomocné pole, do něhož se ukládají mezivýsledky získané při zpracování jednotlivých buněk.

Během zpracovávání buněk se neustále udržuje počet jednotlivých hodnot v poli `statistika`. Pokud projdeme a zpracujeme všechny buňky, jsou pomocí údajů z tohoto pomocného pole aktualizovány hodnoty v `Ploše`. Následně jsou prohozeny množiny `Souc` a `Nasl`. Tím se dosáhne, že v následujícím kroku budeme zpracovávat buňky, které jsme ukládali do množiny `Nasl`. Dále se do souboru `statistika.csv`, který je simulátorem generován nahraje aktuální počet jednotlivých hodnot z pole `statistika`. Získáme tím přehled o vývinu našeho automatu v jednotlivých krocích.



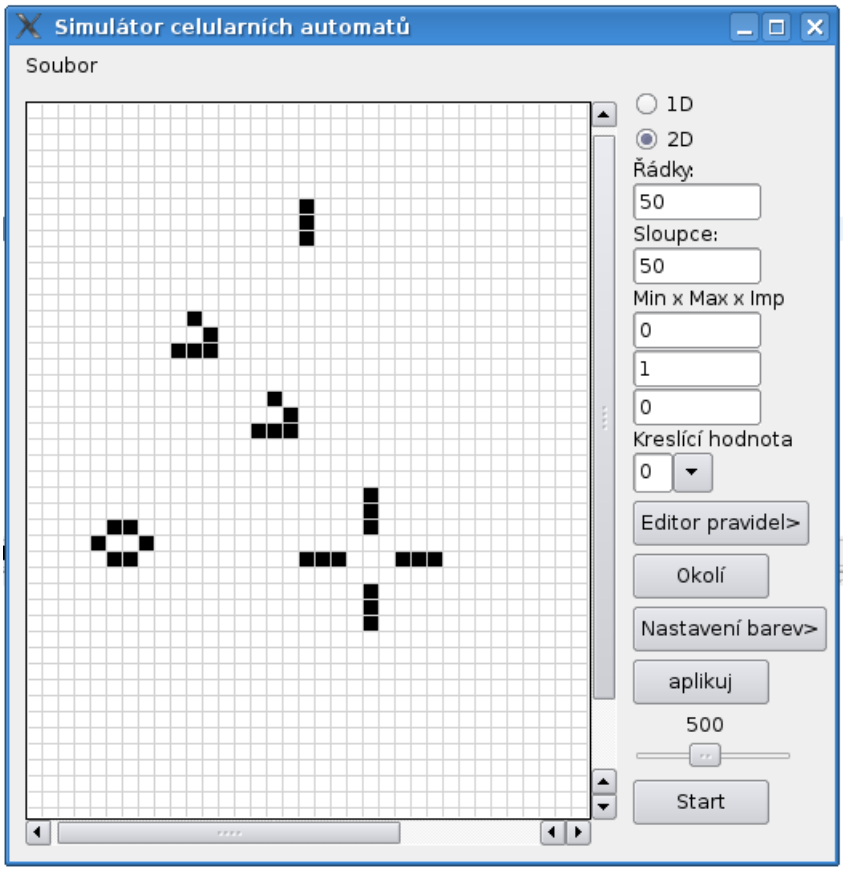
Obrázek 4.1: Vývojový diagram pro průchod jednoho kroku simulace

4.4 Implementace grafického rozhraní

Pro implementaci grafického rozhraní jsem zvolil knihovnu wxWidgets ze dvou důvodů. V prvé řadě je to knihovna pro C++, čili nevzniká problém s používáním knihovny pro simulátor a za druhé je to přenositelnost mezi různými systémy, kde si aplikace vždy zachovávají nativní vzhled podle daného systému.

Hlavní okno aplikace je implementováno třídou `MainFrame`, která je dědicem třídy `wxFrame`. Tato třída zajišťuje rozložení správu všech ovládacích prvků. `MainFrame` tvoří i rozhraní mezi uživatelem a kreslící plochou. Ta nese název CellCanvas a je potomkem třídy `wxScrolledWindow`. Třída `CellCanvas` zajišťuje veškerou obsluhu kreslící plochy a mimojiné provádí komunikaci se simulátorem, jehož instance je jednou z částí této třídy.

O vykreslování buněk na plochu se stará metoda `OnPaint()`, která při každém generování události pro kreslení projde buňku po buňce robrazované části plochy a tyto buňky



Obrázek 4.2: Ukázka grafického prostředí aplikace

vykreslí. Při zakreslování buněk na plochu pomocí myši se zvolená hodnota zapisuje do plochy uložené v simulátoru. Její vykreslení na viditelnou plochu je zajištěné právě generováním události pro kreslení. Velikost vykreslovaných buněk je dána vnitřní proměnnou cellsize. Tato proměnná lze za běhu programu měnit pomocí metody `Zoom`. Dosáhneme tím přiblížení nebo oddálení buněk. Na obrázku 4.2 lze vidět výslednou podobu grafického prostředí.

Další implementační zvláštností grafického rozhraní je implementace zobrazování stavů buněk u jednodimenzionálních automatů. Rozdíl oproti dvoudimenzionálním automatům je v tom, že u 2D automatů jsou změny vykreslovány po každé změně na plochu, čímž se předchozí stavy přepíší. Není tedy možné sledovat historii rozložení na ploše. Kdybychom se o to pokusili byl by rozbrazený vývoj zřejmě velmi nepřehledný. Jednodimenzionální automaty však tuto možnost vzhledem ke svým rozměrům poskytují.

Jednotlivé kroky lze vykreslovat na řádky pod sebe a tím získáme možnost vizuálně sledovat vývoj buněk. Vzhledem k tomu, že v simulátoru je uložena pouze jedna plocha, se kterou se pracuje je nutné ukládání jednotlivých kroků do pomocné paměti. Jako tuto paměť jsem zvolil 2D matici, ktará obsahuje 200 kroků vývoje simulace. Šířka jednotlivých řádků je dána rozměrem automatu. Ukládání funguje cyklicky, čili je-li dosaženo posledního řádku, začne se ukládat opět na první řádek. Buňky pro vykreslování na plochu pro uživatele jsou získávány z této paměti a indexace v matici je odvozena od počtu již provedených kroků a od rozměrů zobrazované plochy.

4.5 Testování a ukázkové příklady

Tato kapitola je věnována především ukázkovým příkladům, kterými byl systém testován. Příklady byly vybrány tak, aby dostatečně ukázaly použitelnost simulátoru a mohly sloužit jako ukázkové příklady demonstrující využití CA

4.5.1 Počítání druhé mocniny pomocí CA

Pro tento příklad jsem zvolil ukázku demonstrující schopnost CA počítat matematické funkce. Jedná se o počítání druhé mocniny přirozených čísel, které je možné najít a blíže se s ní seznámit v [2, 6].

Tento model využívá jednodimenzionální CA. Dále je nutné zajistit rozsah hodnot od 0 do 7 s implicitní hodnotou 0. Využito je radiální okolí s $r = 1$. Sada pravidel je podle [2] následující:

$$\{0, \_, 3\} \to 0, \{\_, 2, 3\} \to 3, \{1, 1, 3\} \to 4, \{\_, 1, 4\} \to 4, \{1|2, 3, \_\} \to 5,$$
$$\{p : 0|1, 4, \_\} \to 7 - p, \{7, 2, 6\} \to 3, \{7, \_, \_\} \to 7, \{\_, 7, p : 1|2\} \to p,$$
$$\{\_, p : 5|6, \_\} \to 7 - p, \{5|6, p : 1|2, \_\} \to 7 - p, \{5|6, 0, 0\} \to 1,$$
$$\{\_, p : 1|2, \_\} \to p, \{\_, \_, \_\} \to 0$$

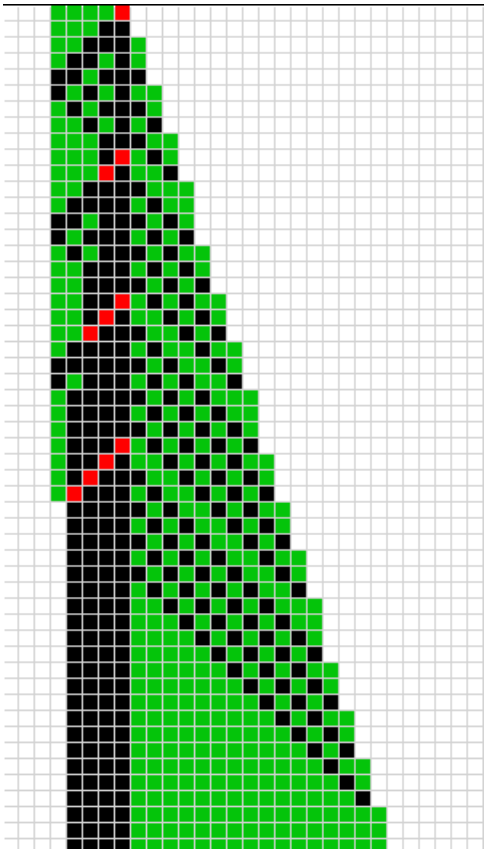
Kde

- $\{a, b, c\} \to d$ znamená, že buňka b se sousedy a a c se v následujícím kroku změní na hodnotu d
- $\_$ znamená libovolnou hodnotu
- $a|b$ znamená a nebo b
- $p : a|b$ znamená že $p = a$ nebo $p = b$

Celý příklad funguje tak, že zapíšeme do počáteční konfigurace automatu za sebe buňky s hodnotou 1 zakončené buňkou s hodnotou 3. Počet jedniček udává číslo, jehož druhou mocninu chceme vypočítat. Po spuštění simulace se začně automat vyvíjet podle výše uvedených pravidel. Po jisté době se automat dostane do stavu, z něhož se už nemění. Odečteme-li z tohoto stavu počet buněk s hodnotou 1 získáme tím výsledek.

Na snímku 4.3 je ukázán vývoj automatu pro výpočet mocniny čísla 4, to je reprezentováno čtyřmi zelenými buňkami s hodnotou 1 zakončenými červenou buňkou s hodnotou 3. Na konci simulace je již vidět, že se automat už nijak nevyvíjí a jedná se tedy o výsledek. Opravdu obsahuje 16 zelených buněk reprezentujících výsledek. Dále je možné se podívat na výsledek zapsaný v souboru statistika.csv, který je automatem automaticky generován, v němž je počet buněk s danou hodnotou v každém kroku.

```
generace;H0;H1;H2;H3;H4;H5;H6;H7;
0;25;4;0;1;0;0;0;0;
1;25;3;0;0;1;1;0;0;
⋮
49;11;14;4;0;0;0;1;0;
50;10;16;4;0;0;0;0;0;
51;10;16;4;0;0;0;0;0;
```



Obrázek 4.3: Výpočet mocniny čísla 4 pomocí CA

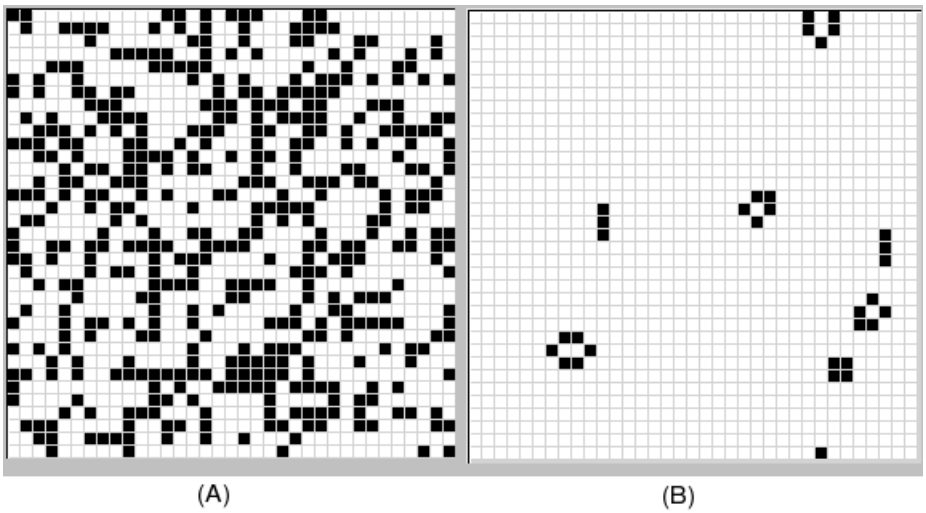
4.5.2 Hra Life

Conwayova hra Life je typickým příkladem ukázky CA. Tato hra se odehrává na dvourozměrné ploše se dvěma druhy buněk, 0 – mrtvá a 1 – živá. Tyto buňky prozkoumávají své nejbližší osmiokolí, jedná se tedy o Moorovo okolí(obr. 2.1). Jak již bylo uvedeno v 2.2, hra se řídí několika jednoduchými pravidly:

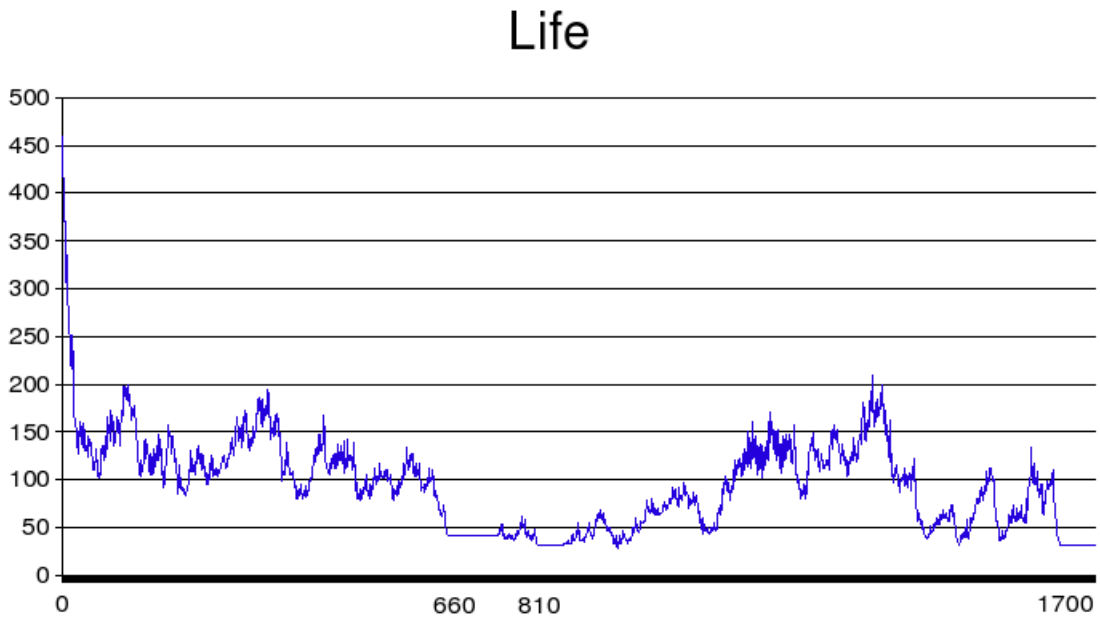
- **zrození** – buňka ožije, jsou–li v okolí 3 živé buňky
- **přežití** – buňka přežívá, jsou–li v okolí 2 nebo 3 živé buňky
- **uhynutí** – ve všech ostatních případech buňka umírá

Na obrázku 4.4 lze vidět počáteční a koncový stav hry pro velikost pole 35 na 35 buněk. Graf na obrázku 4.5 popisuje vývoj živých buněk na tomto poli v jednotlivých krocích. Počáteční stav živých buněk byl 449. Po začátku simulace se tento počet výrazně zmenšil a začal se pohybovat v rozmezí 100 a 150 buněk.

Okolo kroku 660 a 810 nastal zajímavý stav. V těchto okamžicích se počet buněk ustálil, ale byly vytvořeny tzv. „glidery“, čili shluky, které se pohybují polem a mají stále stejnou velikost. Po střetnutí se stabilními shluky se vývoj vždy opět odstartoval, až do kroku 1700, kdy byly vytvořeny nepohybující se útvary, které žačaly oscilovat mezi dvěma stavy.



Obrázek 4.4: Hra Life:(A)–počáteční konfigurace; (B)–koncový stav

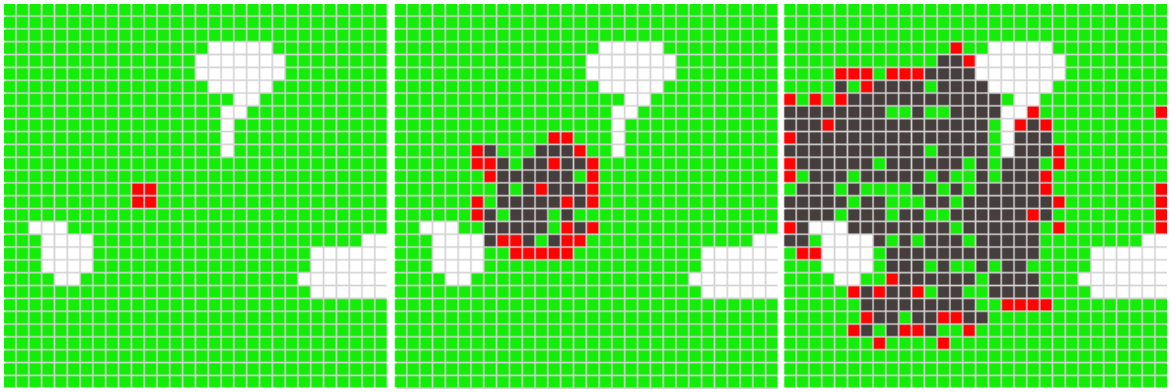


Obrázek 4.5: Hra Life: Vývoj živých buněk v jednotlivých generacích

4.5.3 Šíření požáru lesem

Tento příklad demonstruje využití pravděpodobnosti v modelech. Jedná se o model, kterým se simuluje šíření požáru lesním porostem a následná obnova spáleniště. V modelu jsou využívány dvě pravděpodobnosti – p a q, podle nichž se model vyvíjí. Model se řídí podle jednoduchých pravidel:

- je-li v okolí lesního porostu požár, vzplane les s pravděpodobností p
- oheň se v následujícím kroku změní v popel
- popel se s pravděpodobností q změní opět v lesní porost



Obrázek 4.6: Ukázka šíření požáru

Na obrázku 4.6 je vidět ukázka takového šíření požáru. Pravděpodobnost p udávající vznícení stromů je nastavena na 50 %. Nové stromy se z popela zrodí s pravděpodobností $q = 8\,\%$. Pro tento případ bylo využito standardní Moorovo okolí. Z obrázku je patrné, jak se pořár šířil do všech stran a spálené stromy se postupně měnily v nové stromy.

Kapitola 5

Závěr

Testování prokázalo, že simulátor je po funkční stránce v pořádku, což bylo potvrzeno množstvím testů při nichž byl program laděn. Domnívám se tedy, že systém splnil své poslání, pro které byl vytvořen. Měl být vyvinut tak, aby mohl sloužit a být využíván jako učební pomůcka v předmětu IMS – Modelování a simulace. Má ukázat uživateli fungování CA, dát mu možnost vytvořit si vlastní projekty a modely a tím u něj vzbudit zájem o vlastní možné využití CA. Toto vše program uživateli poskytuje.

Největší sílu mého programu spatřuji v jeho univerzálnosti. Na internetu jsou dostupné desítky simulátorů CA ať už jako samostatné programy, nebo aplety běžící na internetové stránce. Většina z nich má však nějakou omezující vlastnost. Ať už jde o pevně zadanou velikost, omezenou sadu přechodových pravidel, pevně zadaný počet možných hodnot nebo naprosto neintuitivní zadávání přechodových pravidel. Můj program dovoluje uživateli přístup, ve kterém nemusí hledat na internetu či v jiných zdrojích aplikaci, o níž by záhy zjistil, že mu nevyhovuje. Stačí, aby uměl průměrně programovat v jazyce C a může si vytvořit svůj vlastní celulární automat na míru a začít jej ihned používat.

Díky své univerzálnosti umožňuje vylepšení výuky předmětu Modelování a simulace, kde mohou být pomocí programu během přednášky ukázány studentům různé možnosti využití celulárních automatů v oblasti modelování a simulací. Díky pravidlům zapsaných v jazyce C může být pro studenta snadnější jejich pochopení a porozumění jejich principům, vlastnostem a fungování. Program lze dále využít během demonstračních cvičení, kde si mohou studenti prakticky vyzkoušet tvorbu simulačních modelů s využitím celulárních automatů. Jednou z dalších možností je využití simulátoru při řešení projektu, ve kterém mohou studenti plnohodnotně využít všech jeho vlastností k vytvoření vlastního simulačního modelu, který budou následně simulovat a získávat tak potřebná data.

Existuje ale několik věcí, které byly během vývoje a testování objeveny. Jejich odstraněním by se dosáhlo vylepšení a rozíření stávajícího systému. Jedná se především o následující body:

- přenositelnost – použití jazyka C++ a jeho knihovny wxWidgets dovoluje přenositelnost na jakoukoliv platformu. V současné době je však program funkční pouze pod operačním systémem Linux s nainstalovanou knihovnou wxWidgets 2.8. Po několika drobných úpravách kódu by neměl být problém zprovoznit program i pod systémem Windows.

- optimalizace způsobu vykreslování – systém je tak rychlý, jak rychle dokáže vykreslovat své změny. Zvláště při zobrazení velké části plochy způsobuje vykreslování velkého

množství buněk znatelné zpožďování. Bylo by tedy vhodnější pouze překreslovat buňky u nichž došlo ke změně stavu.

- více typů ohraničení plochy – v současné době je implementována pouze tzv. nekonečná neboli toroidní plocha. Tento systém však není vždy ten nejlepší, proto by měl mít uživatel možnost si vybrat mezi více variantami jakými jsou například fixní, reflektující nebo adiabatické ohraničení.
- schopnost vracet výsledek více buněk – toto je myšleno pro schopnost pracovat s Margolusovým okolím, kde se zpracovává více buněk najednou. V současném systému se zpracovává pouze jedna buňka.

Literatura

[1] Droz M. Chopard B. *Cellular automata: modeling of physical systems.* Cambridge University Press, Cambridge, 1998. ISBN 0-521-46168-5.

[2] Artificial Life Group FEI TU Košice. Planet alife. [online] Naposledy navštíveno 1.5. 2008. `http://alife.tuke.sk/`.

[3] Sekanina L. *Evolvable components: from theory to hardware implementations.* Springer-Verlag, 2004. ISBN 3-540-40377-9.

[4] Jossutis Nikolai. *C++ standardní knihovna a STL.* Computer Press, Brno, 2004. ISBN 80-251-O511-1.

[5] R.Highfield P. Conevey. *Mezi chaosem a řádem.* Mladá fronta, Praha, 2003.

[6] Wolfram S. *New kind of science.* Wolfram Media, USA, 2002. ISBN 1-57955-008-8.