VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

PLÁNOVÁNÍ CESTY MOBILNÍHO ROBOTU POMOCÍ CELULÁRNÍCH AUTOMATŮ
MOBILE ROBOT PATH PLANNING BY MEANS OF CELLULAR AUTOMATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Patrik Gofroj**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. Jiří Dvořák, CSc.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Patrik Gofroj**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**RNDr. Jiří Dvořák, CSc.**
Akademický rok:	2018/19

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Plánování cesty mobilního robotu pomocí celulárních automatů

Stručná charakteristika problematiky úkolu:

Úkolem systému pro plánování cesty robotu je najít cestu z výchozího do cílového bodu tak, aby se robot nedostal do kolize se známými překážkami, aby nebyla porušena případná omezení kladená na pohyb robotu a aby byla optimalizována nějaká kriteriální funkce. Jednou z možností, jak tento problém řešit, je použití celulárních automatů.
Celulární automat je diskrétní dynamický systém tvořený pravidelnou mřížkou buněk, z nichž každá se může nacházet v konečném počtu stavů. Počáteční generace je nastavena přiřazením stavu pro každou buňku. Další generace se tvoří podle nějakého pevného pravidla, které určuje nový stav každé buňky na základě jejího aktuálního stavu a stavů buněk v jejím okolí.

Cíle bakalářské práce:

Popsat přístupy k plánování cesty mobilního robotu.
Popsat problematiku celulárních automatů se zaměřením na plánování cesty.
Implementovat vybrané metody plánování cesty.
Provést a vyhodnotit ověřovací a srovnávací experimenty.

Seznam doporučené literatury:

MARTINS, L. G. A. et al. An improved robot path planning model using cellular automata. In: Giuliani, M. et al. (eds.) TAROS 2018, LNAI, vol. 10965, pp. 183-194, 2018.

FERREIRA, G. B. S., VARGAS, P. A., OLIVEIRA, G. M. B. An improved cellular automata-based model for robot path-planning. In: Mistry, M. et al. (eds.) TAROS 2014, LNAI, vol. 8717, pp. 25-36, 2014.

SYED, U. A., KUNWAR, F. Cellular automata-based real-time path-planning for mobile robots. International Journal of Advanced Robotic Systems, vol. 11, no. 7, 2014.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2018/19

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt

První část této práce se zabývá popisem způsobů plánování cesty mobilního robotu. Hlavním zaměřením této práce jsou celulární automaty a jejich využití v dané problematice hledání optimální cesty.

Abstract

This thesis describes certain ways of mobile robot path planning. The main concern of this thesis are cellular automata and their usage in this problem of searching of optimal path.

Klíčová slova

Plánování cesty, mobilní robot, celulární automat

Keywords

Path planning, mobile robot, cellular automaton

Bibliografická citace

GOFROJ, Patrik. *Plánování cesty mobilního robotu pomocí celulárních automatů.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/116801. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Dvořák.

Čestné prohlášení

Prohlašuji, že tato bakalářská práce je mým dílem, které jsem zpracoval samostatně pod vedením RNDr. Jiřího Dvořáka, CSc. a s použitím zdrojů uvedených v seznamu literatury.

V Brně dne 24.5.2019 Patrik Gofroj

……..

Poděkování

Rád bych poděkoval vedoucímu práce RNDr. Jiřímu Dvořákovi, CSc. za věnovaný čas, připomínky a cenné rady při tvorbě této bakalářské práce.

Obsah

1 Úvod

Problematika plánování mobilních robotů je v posledních několika letech až desetiletích jednou z největších a zároveň nejdůležitějších témat v oblasti programování robotů. Robot ze své podstaty potřebuje způsob, jakým ovlivňovat a jak se sám chovat na základě svého okolí, ať už jde o prostý automatický vysavač, který pokryje celou podlahu bytu nebo například hasičský robot, který úspěšně přijede ke správné hořící budově. Většina metod plánování cesty má avšak jeden kámen úrazu a tím je statičnost celé diskrétní soustavy, protože vyhledávací funkce prohledává daný prostor za předpokladu statických počátečních podmínek a to takových, kdy se žádná z překážek, cíl cesty ani počáteční poloha robota nijak není schopná měnit v čase (jde o tzv. globální plánování). Toto vede sice v některých případech k nalezení optimální cesty, avšak v momentě, kdy by se měla objevit dynamická překážka (například v případě robotického vysavače by to mohl být domácí mazlíček který se rozhodne uvelebit se uprostřed pokoje za běhu vysavače), robot by nebral v úvahu vzniknuvší překážku a mohlo by dojít ke kolizi. Užití metody celulárních automatů tomuto problému předchází samotnou podstatou „plánování“, kdy robot během každého kroku v diskrétním systému zvolí další vhodný krok na základě svého okamžitého okolí. Celulární automaty tedy plánují cestu lokálně.

Tato práce se zabývá přesně touto problematikou, kdy byl naprogramován rozhodovací systém na bázi celulárních automatů, který je v průběhu práce porovnán s jinými, klasičtějšími způsoby hledání cesty. Nejlepších výsledků poté dosahují algoritmy kombinující jak lokální, tak globální přístupy k plánování cesty.

2 Cíl práce

Hlavním cílem této práce je navrhnout a popsat způsob hledání cesty mobilního robotu za použití celulárních automatů. Práce je tudíž rozvržena do několika bodů, a to:

- Rešeršní část zabývající se popisem klasických algoritmů hledání cesty
- Rešeršní část zabývající se popisem a vysvětlením fungování celulárních automatů
- Praktická část navržení a implementace algoritmu hledání cesty na základě celulárních automatů
- Vyhodnocení porovnávacích experimentů mezi klasickým a zde navrženým způsobem hledání cesty

3 Plánování cesty

Plánování cesty je termín objevující se hlavně (ale nejen) v oblasti robotiky, kde se jedná o soubor algoritmů a procesů, které mají za cíl vytvořit optimální cestu mezi počátečním a cílovým bodem v daném prostoru. Takto vytvořená trajektorie pohybu musí splňovat několik požadavků, jako může být např. co nejkratší délka trasy či absence kolizí s překážkami. V některých případech je robot omezován stupni volnosti sebe samotného (například automobil se nemůže volně pohybovat do stran). Takový robot se nazývá neholonomním. [1;2]

Každý algoritmus plánování cesty pracuje v daném pracovním prostoru, tento pracovní prostor je téměř bez výjimek 2D plocha pozorovaná z ptačí perspektivy anebo 3D prostor. Podle typu pracovního prostoru a typu předpokládaného robota se odvíjí tzv. konfigurační prostor C. Konfigurační prostor obsahuje všechny možné konfigurace robota v prostoru, pro 2D prostor jde buďto o vektor polohy (x,y) anebo v případě uvažování rotace robota okolo osy z o vektor polohy a natočení (x,y,θ). Pro případ 3D prostoru se potom jedná (bez předpokladu omezení vztažených na pohyb) o vektor polohy (x,y,z) a vektor Eulerových úhlů (α,β,γ). Konfigurační prostor se dále skládá z několika podmnožin C. Část pracovního prostoru, ve kterém se robot může volně pohybovat bez kladených omezeních se nazývá volný prostor C_{free}, jeho doplňkem je potom tzv. prostor překážek C_{obs}, což je přímo vymezená část, ve které se robot pohybovat nemůže a při kontaktu s hranicí C_{obs} by došlo ke kolizi. Cílový prostor je předem danou podmnožinou C_{free}, která je zároveň místem, do něhož se robot má dostat. Při globálním plánování trasy je poloha cílového prostoru předem známa subjektu, ale při čistě lokálním plánování pohybu se robot může dostat do stavu, kdy přímo „nevidí“ cíl, tudíž předtím musí projít několika virtuálními cílovými prostory, které jsou v dané viditelnosti robota. Posledním základním prvkem konfiguračního prostoru jsou tzv. nebezpečné zóny. Tyto zóny jsou opět podmnožinou C_{free}, ovšem jsou to místa, kterým se bude robot snažit vyhnout do momentu, kdy by mu byl úplně zamezen pohyb, například může jít o blízké hrany překážek anebo změnu terénu, která by ztížila pohyb robota. [1;2]

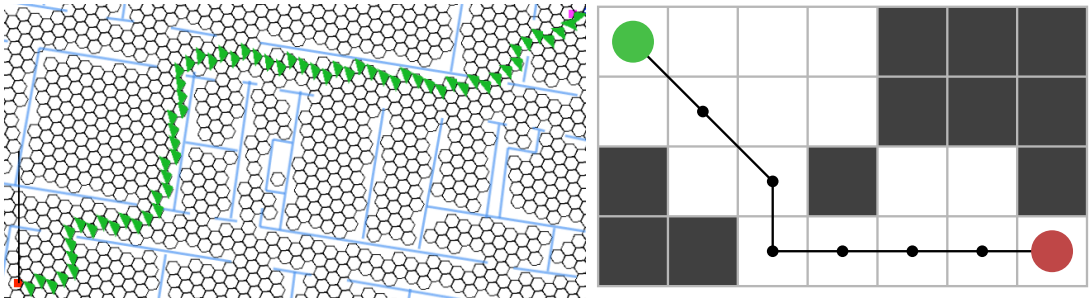
3.1 Globální algoritmy plánování cesty

V této podkapitole budou popsány některé globální přístupy k plánování cesty.

3.1.1 Mřížková metoda (Grid-based search)

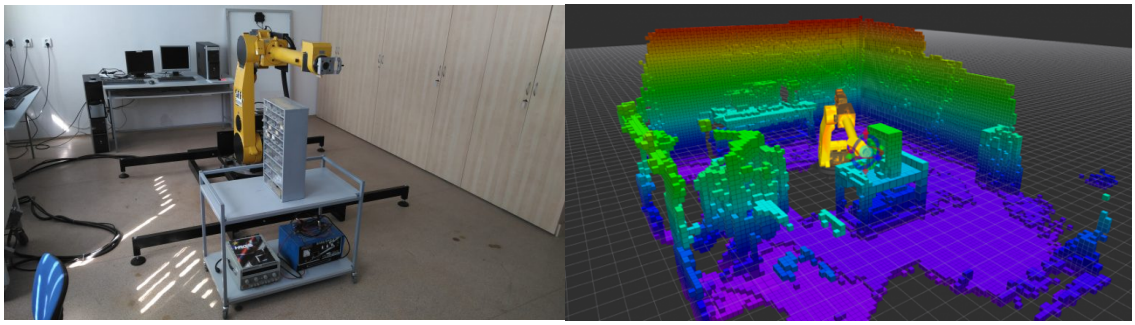
Při použití mřížkové metody se celý pracovní resp. konfigurační prostor překryje a následně zpracuje ze spojitého volného prostoru a překážek na diskrétní systém buněk, kde každá buňka obsahuje informaci o tom, zda spadá pod množinu C_{free}, C_{obs}, cíl, aktuální polohu robota, popřípadě nebezpečnou zónu. Jednotlivé buňky mohou nabývat různých tvarů, defaultně uvažujeme čtvercovou 2D síť, ale není nezvyklé uvažovat i například hexagony. Pro případ čtvercové 2D sítě poté existují buďto 4 anebo 8

sousedících buněk, do kterých se robot může přesunout v závislosti na vlastním provedení; pro 2D hexagony jde o 6 sousedů. Výpočetní náročnost lze ovlivnit rozlišením překrývající mřížky. Jako optimální rozlišení volíme takové, kdy není omezena realističnost vypočteného řešení malým rozlišením, ale zároveň je splněn požadavek na výpočetní techniku, tzn. problém je v reálném čase vypočitatelný dostupnými prostředky. [2]



Obrázek 1 - Příklad hexagonální a čtvercové mřížky při plánování cesty

Mřížková metoda má své uplatnění i při transformování 3D prostoru na diskrétní systém, ovšem výpočtová náročnost rapidně roste kvůli kvadratickému nárůstu počtu buněk v konfiguračním prostoru (pro kubickou mřížku a robota pouze s možností translace ve všech směrech a původních 8 sousedech se stane robot s 26 možnostmi dalšího kroku). Takto transformovaný prostor se označuje jako oktomapa. [2]



Obrázek 2 - Příklad rozložení pracovního prostoru okolo manipulátoru do oktomapy

Nejrozšířenějším příkladem vyhledávacího algoritmu využívajícího diskrétní rozložení prostoru na mřížku je algoritmus A*. [2;3]

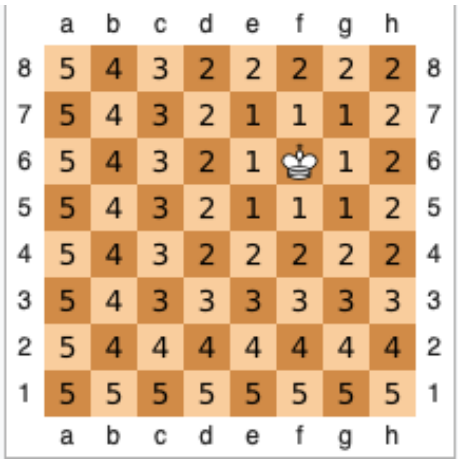
Algoritmus A*

Tento vyhledávací algoritmus v základní formě hledá cestu mezi několika předem známými uzly, kde každý uzel má určenou vlastní hodnotu. [3] A* vychází svou strukturou z Dijkstrova vyhledávacího algoritmu, ale zároveň k němu přidává heuristickou funkci, která obvykle značně urychluje výpočetní rychlost. Základní funkcí, kterou se A* řídí je

$$f(v) = h(v) + g(v),$$

kde *h(v)* představuje onu přidanou heuristickou funkci a *g(v)* je vzdálenost mezi počátečním bodem cesty a bodem *v*. Mezi heuristické funkce patří:

1. Manhattanská metrika
    - Předpokládá se, že pohyb je možný pouze ve 4 směrech (vychází se z pravoúhlého systému ulic na Manhattanu)
    - Je popsána vztahem $d = |p_1 - q_1| + |p_2 - q_2|$, kde (p_1,p_2) jsou souřadnice aktuální pozice a (q_1,q_2) souřadnice cíle
2. Diagonální (Chebyshevova, Octile) metrika
    - Je možný pohyb v 8 směrech. Za vzdálenost mezi dvěma uzly v mřížce považuje největší z rozdílů jejich souřadnic (viz obr. 3)
    - Octile metrika je popsána vztahem $d = \max(|p_1 - q_1|, |p_2 - q_2|)$
    - Chebyshevova metrika je popsána vztahem $d = \max(|p_1 - q_1|, |p_2 - q_2|) + (\sqrt{2} - 1) * \min(|p_1 - q_1|, |p_2 - q_2|)$



Obrázek 3 - Ukázka Chebyshevovy metriky aplikovaná na posuv krále po šachovnici

3. Euklidovská metrika
    - Je možný pohyb v jakémkoliv směru
    - Je popsána vztahem $d = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$, kde (p_1,p_2) jsou souřadnice aktuální pozice a (q_1,q_2) souřadnice cíle

Při situaci, kdy $h(v) = 0$, tzn., že není užito žádné heuristiky, jde o variantu Dijkstrova algoritmu, která na rozdíl od něj nehledá nejkratší cesty do všech ostatních uzlů, ale jen nejkratší cestu ze startu do cíle. Výše zmíněné heuristiky berou v potaz pouze vzdálenost a možnosti pohybu robota. V případě reálného terénu je vhodné do dané heuristiky zakomponovat cenu přesunu o jedno pole, například přesun přes písek stojí virtuálně víc než jízda po asfaltu. [3]

Využití nachází tento algoritmus v mnoha odvětvích, například v počítačových hrách. Původně byl navrhnut jako algoritmus pro hledání cesty v grafu (Graph Traversal Algorithm) jakožto součást Shakey projektu[1], ve kterém šlo o vytvoření autonomního robota, jenž by byl schopen samostatného rozhodování o vlastním pohybu. Autory algoritmu byli Nils Nilsson, Bertram Raphael a Peter Hart. [3]

[1] http://www.ai.sri.com/shakey/

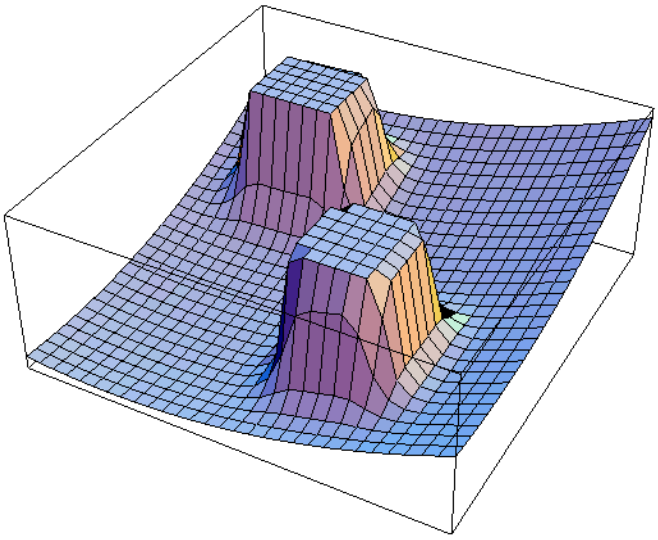
Algoritmus A* je zjednodušeně popsán pomocí následujícího pseudokódu. [4] Množina expandovaných (prozkoumaných) buněk je nazvána Closedset a množina buněk čekajících na expanzi (prozkoumání) je nazvána Openset.

```
function A*(start, cíl)
    closedset := prázdná množina                    // Množina již uzavřených uzlů.
    openset := množina obsahující pouze počáteční uzel // Množina otevřených uzlů.
    g_skore[start] := 0                             // Délka aktuální optimální cesty.
    h_skore[start] := heuristický_odhad_vzdálenosti(start, cíl)
    f_skore[start] := h_skore[start]                // Předpokládaná délka cesty mezi startem a cílem jdoucí přes y.
    while openset is not empty
        x := otevřený uzel s nejmenší hodnotou f_skore[]
        if x = cíl
            return rekonstruuj_cestu(přišel_z[cíl])
        vyjmi x z openset
        přidej x do closedset
        for each y in sousední_uzly(x)
            if y in closedset
                continue
            stávající_g_skore := g_skore[x] + d(x, y)

            if y not in openset
                add y to openset
                stávající_je_lepší := true
            elseif stávající_g_skore < g_skore[y]
                stávající_je_lepší := true
            else
                stávající_je_lepší := false
            if stávající_je_lepší = true
                přišel_z[y] := x
                g_skore[y] := stávající_g_skore
                h_skore[y] := heuristický_odhad_vzdálenosti(y, cíl)
                f_skore[y] := g_skore[y] + h_skore[y]
    return failure

function rekonstruuj_cestu(aktuální_uzel)
    if přišel_z[aktuální_uzel] is set
        p = rekonstruuj_cestu(přišel_z[aktuální_uzel])
        return (p + aktuální_uzel)
    else
        return aktuální_uzel
```

*Obrázek 4 - Pseudokód popisující algoritmus A**

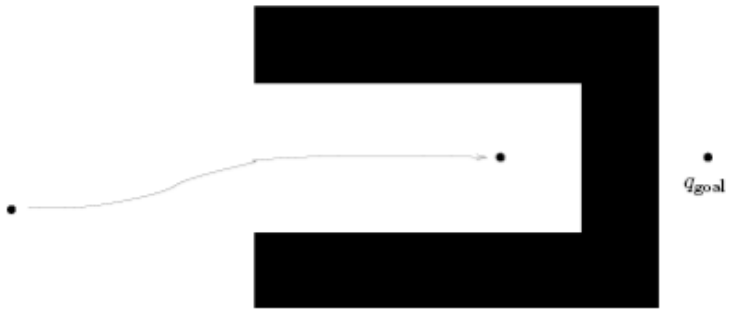
3.1.2 Potenciálová pole (Potential fields)

Základní ideou algoritmu užívajících potenciálová pole je představa, že celá naše pracovní plocha je překryta polem, kde každý bod, potažmo buňka, má danou hodnotu virtuální potenciální energie. Počáteční poloha robota má vysokou hodnotu tohoto virtuálního potenciálu, stejně tak mají vysokou hodnotu uvažované překážky, oproti tomu cílová destinace má nejnižší hodnotu potenciálu v celém grafu. Robot má potom za cíl přesunout se tímto polem do bodu s nejnižším potenciálem, tedy se pohybuje ve směru záporného gradientu potenciální energie. Na obrázku 4 je představa takovéhoto pole, kde startovní pozice je v pravém horním rohu (nejvyšší bod grafu) a cíl je v levém dolním rohu (nejnižší bod grafu). Na tomto modelu je též dobře viditelné „odpudivé pole“ okolo kořenů překážek. [5]



Obrázek 5 - Potenciálové pole

Jiný způsob, jak si lze toto pole představit, je uvažovat robota nabitého kladným elektromagnetickým nábojem, cíl záporným nábojem a překážky opět kladným nábojem. V této představě bude robot „přitahován“ silou k cíli, ale zároveň odpuzován od překážek. Rizikem použití čistě této metody je možnost uvíznutí robota v lokálním minimu, viz obrázek 5. [5]



Obrázek 6 - Riziková situace uvíznutí robota v lokálním minimu

3.1.3 Intervalová metoda (Interval-based search)

Intervalová metoda hledání cesty obdobně jako mřížková metoda překryje celý pracovní prostor, ale v jejím případě ne mřížkou, ale tzv. „dlážděním“ (eng. Pavings). [6] Dláždění C_{free} je poté dále rozloženo do jemnějších dláždění (sub-pavings) C^+ a C^-, kde platí:

$$C^- \subset C_{free} \subset C^+$$

Z toho vyplývá, že subjekt se může volně pohybovat v C^- a zároveň nemůže překročit hranice C^+. Pro každou podmnožinu je poté vytvořen samostatný graf okolí a následně může být použit pro výpočet optimální cesty zvolený algoritmus vyhledávání cesty, jakým může být například již zmíněný A*. Tato metoda, stejně jako mřížková, trpí nevýhodou, kdy při nárůstu dimenze problému exponenciálně roste i potřebná výpočetní síla. [2;6]

Na obrázku 6 lze vidět aplikace intervalové metody, kde červené pole (sjednocení všech červených čtverců) je množinou C^-, tj. volný prostor; dále modré pole jsou dané překážky. Množina obsahující zelený okraj překážek a červená pole je dlážděním C^+. Černá křivka je vypočtenou cestou. [2]

Obrázek 7 - Intervalová metoda

3.1.4 Odměnová metoda (Reward-based search)

Myšlenka, na které je postavena odměnová metoda plánování cesty, spočívá v tom, že robot nemá explicitně zadáno, v jakých podmínkách se má pohybovat, ale na základě metody pokus-omyl sám „zjistí“, které volby nesly větší zisk ve formě nějaké odměny. [7]

Systém postavený na takovémto posilovaném učení (z ang. Reinforcement Learning) se skládá ze 4 stavebních kamenů. *Politika systému* určuje, jak se agent v daný moment vždy zachová, *odměnová funkce* definuje zvolený cíl a udává, co se v daném okamžiku jeví jako vhodná volba, *hodnotová funkce* dále udává, co se jeví jako vhodná volba z dlouhodobé perspektivy a v některých případech *model prostředí* imituje chování celé soustavy (například při zadání stavu a akce může predikovat další akci a získanou odměnu). [7]

4 Celulární automaty

První koncept celulárních automatů (CA) vzniknul v roce 1940 zásluhou Stanislawa Ulama a Johna von Neumanna v Národní Laboratoři Los Alamos. V roce 1970 byla tato idea představena širší veřejnosti se vznikem Hry Života (angl. Game of Life). Dalším velkým pionýrem ve výzkumu CA se stal Stephen Wolfram, jenž od r. 1980 vytvářel komplexní myšlenkový model popsaný v jeho díle A New Kind of Science (publ. 2002).

Celulární automaty matematicky modelují komplexní systémy, tj. dynamické systémy, jež se mění nejen v závislosti na čase, ale i aktivně reagují na změnu celého svého prostředí podle předem pevně daných pravidel. [8] Dobrým případem komplexních systémů mohou být například jakékoliv živé organismy. Všechny CA splňují těchto několik podmínek:

1. Existují ve formě diskrétní mřížky (*n* rozměrné, nejčastěji 1D, 2D, 3D).
2. Jednotlivé buňky mřížky jsou homogenní – všechny jsou identické a mohou nabývat stejných hodnot
3. Každá buňka může nabývat v daném okamžiku jeden z diskrétních předem určených stavů
4. Každá buňka reaguje pouze na sousední buňky, tj. na své nejbližší okolí
5. Každá buňka v mřížce aktualizuje svou hodnotu při přechodu do časového okamžiku $t + 1$ podle definované přechodové funkce, jejímž vstupem je stav dané buňky a buněk okolí v čase t.

4.1 Elementární celulární automaty

Elementární celulární automaty jsou nejjednoduššími modely CA. Jde o jedno-dimenzionální CA, kde každá z buněk může nabývat jednoho ze dvou stavů, a to 1 nebo 0. Okolím je poté centrální buňka a její dva nejbližší sousedi. [9]

Wolframův kód je systém číslování vytvořený Stephenem Wolframem pro vytváření takzvaných rule-sets, neboli pravidel, podle kterých se bude měnit další generace buněk, jinými slovy jde o přechodovou funkci buňky $f(okolí_{t-1})$. [9] Při uvažování okolí o velikosti 3 buněk poté existuje $2^3 = 8$ možností, jakou hodnotu bude mít nová centrální buňka. Podle Wolframova kódu jsou tyto možnosti seřazeny sestupně podle konfigurace původního okolí, a to:

Tabulka 1 - Wolframův kód - Číslování buněk

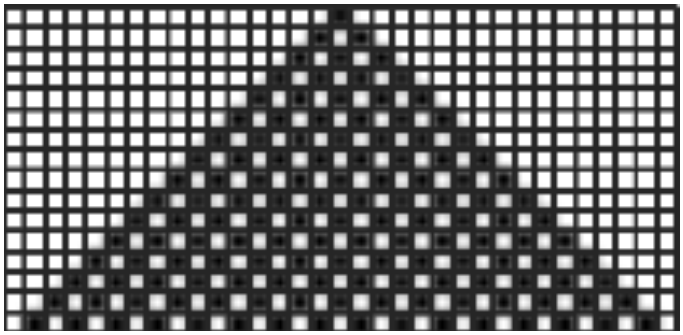
111	110	101	100	011	010	001	000

Každé z konfigurací je poté přiřazena hodnota centrální buňky v nové generaci. Toto pravidlo je nazývané již zmíněným rule-setem. [9]

Tabulka 2 - Wolframův kód - pravidlo 250

111	110	101	100	011	010	001	000
1	1	1	1	1	0	1	0

V tabulce 2 je znázorněn příklad pravidla s číslem 250. Tento rule-set dává jako výsledek šachovnicový vzor mřížky, viz obr. 7. Na tomto obrázku lze vidět časoprostorový průběh elementárního celulárního automatu, kde vodorovně vidíme konfiguraci prostoru v jednom okamžiku a svisle je znázorněn časový průběh, tzn. každý řádek je novou generací. Uvažujeme 0 = bílá buňka a 1 = černá buňka. [9]



Obrázek 8 - Pravidlo 250

Číslování pravidel vychází přímo z konfigurace daného pravidla, například při příkladu výše, kde vycházíme z konfigurace 11111010 (viz tab. 2). Binární kombinace 11111010 se dá zároveň napsat jednodušeji jako decimální číslo 250. Za předpokladu okolí o 3 buňkách, tedy 8 možnostech kombinace, můžeme uvažovat celkem $2^8 = 256$ pravidel. [9]

Toto číslo ovšem vychází již ze zmíněného daného okolí a z předpokladu 2 diskrétních stavů jednotlivých buněk, kdybychom navýšili počet možných stavů i jen o jeden další, v případě barev by šlo o přidání možnosti šedé buňky, navýšil by se nám počet všech možných pravidel na číslo 7 625 597 484 987.[2] [9]

4.1.1 Vlastnosti vzniklých struktur

Elementární CA po průchodu vyšším (řádově desítky až tisíce) počtem generací projevují jeden ze čtyř typů chování. Důležitou zajímavostí je, že tyto čtyři typy chování jsou naprosto univerzální nejen pro nejzákladnější elementární CA, ale projevují se u všech typů 1D CA bez ohledu na to, o kolik složitější budou námi zadaná počáteční pravidla. Některá pravidla se na první pohled zároveň mohou jevit jako jiný druh než který reálně jsou, příkladem budiž pravidlo 62, které se vycházejíc z náhodně vygenerovaného prostoru po několika iteracích jeví jako 4. kategorie, avšak po více iteracích se CA stabilizuje do formy reprezentující 2. kategorii (popis kategorií níže). [9]

1. **Kategorie:** Nejočekávanějším a nejrozšířenějším typem chování při aplikaci jednoduchých pravidel je prosté opakování stále stejného vzoru. Asi dvě třetiny

[2] A new kind of science – Ch.3, p.60

z 256 možností tvoří obrazce s fixní velikostí, které připomínají do nekonečna pokračující linku. Tu třetí třetinu tvoří z největší části rostoucí, ale staticky se neměnící vzor, viz Obr. 7.

2. **Kategorie:** Druhý typ zahrnuje tzv. hnízdící struktury. To znamená že existuje nesložitý obrazec, jako například trojúhelník, který se při přibližování fraktálně rozvíjí do čím dál menších stále stejně koncipovaných trojúhelníkových struktur, viz Obr. 11. Takovýchto hnízdících struktur se v 256 uvažovaných možnostech objevuje 24.

3. **Kategorie:** Třetím typem chování je naprostá náhodnost a nepředvídatelnost, jak se systém bude vyvíjet v následujících krocích. Z našeho výběru pravidel je 10 takových, které spadají do této kategorie. Příkladem může být Obr. 10.

4. **Kategorie:** Poslední čtvrtý typ má v elementárních CA jen ojedinělé zástupce, nejprozkoumanější je pravidlo 110. Časoprostorové znázornění této kategorie spojuje dohromady lokálně pravidelné a náhodné prvky, kde spolu jednotlivé emergentní struktury reagují komplexním způsobem.

4.1.2 Specifická pravidla

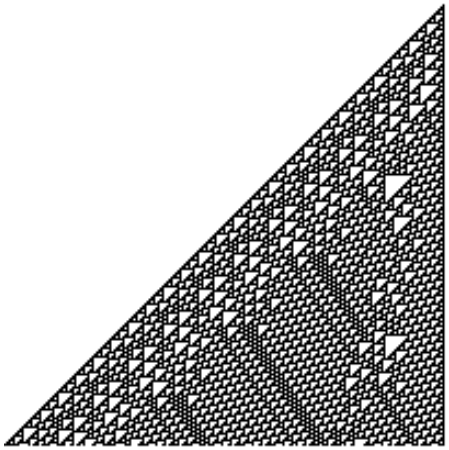
Z již zmíněných 256 kombinací pravidel se jich několik vyznačuje specifickými vlastnostmi. Některá z těchto pravidel budou popsána v této podkapitole.

Pravidlo 110

Jak již bylo zmíněno, pravidlo 110 se nachází na hranici chaosu a stabilní struktury, tudíž spadá do 4. kategorie chování. Zajímavostí pravidla 110 je, že je to prozatím jediné prokázané CA pravidlo, které je zároveň Turingovsky kompletním, to znamená, že je naprosto výpočetně univerzální – jakýkoliv program anebo výpočet dokáže být zpracován tímto pravidlem. [10]

Tabulka 3 - Pravidlo 110

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0



Obrázek 9 - Vizualizace pravidla 110

Jak je vidno na obrázku 8, aplikace pravidla 110 na jednu počáteční buňku má za následek nekonečný rozvoj stabilních a nestabilních struktur, rozvíjející se pouze do levé strany. Pro dokázání turingovské kompletnosti bylo užito tzv. hvězdných lodí (angl. Spaceships), tj. menších struktur uvnitř prostoru CA, jež zdánlivě „proplouvají“ svým stabilním okolím. [10]

Pravidlo 30

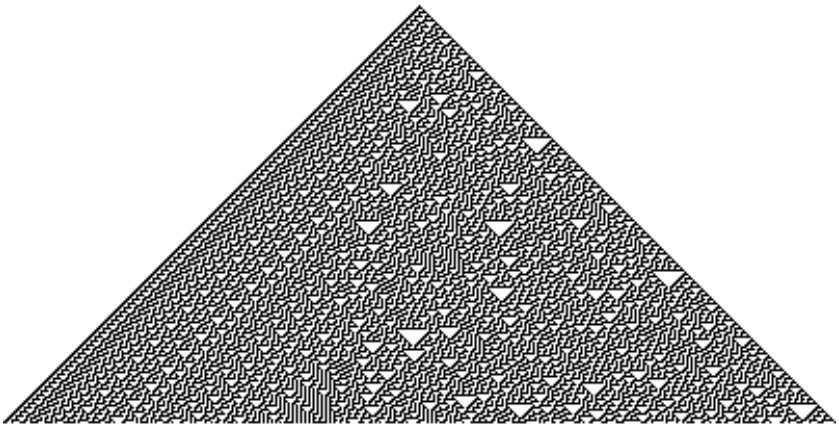
Je typickým příkladem 3. kategorie, protože jasně ukazuje chaotický časoprostorový rozvoj na základě jednoduchých pravidel, zároveň je jeho průběh velmi závislý na počátečních podmínkách. To vede k úvaze, že i v přírodě se vyskytující komplexní vzory a chování může být následkem ne-komplexní podstaty věcí. [9]

Tabulka 4 - Pravidlo 30

111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0

Reálným příkladem, kdy se v přírodě projeví pravidlo 30, je ulita hlemýždě Conus Textile, kde je jasně viditelná podobnost s vzorem produkovaným CA. Toto pravidlo zároveň nachází široké užití i v praxi, kdy například slouží jako základ generátoru náhodných čísel v programu Mathematica. [10]

Obrázek 10 - Conus Textile



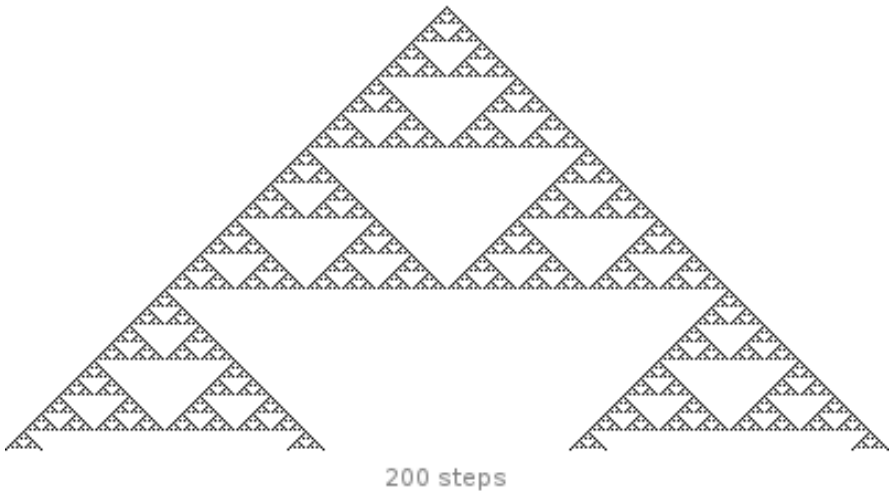
Obrázek 11 - Vizualizace pravidla 30

Pravidlo 90

Je založené na exkluzivní nebo (XOR) funkci. Spadá do 2. kategorie chování CA, tudíž při výchozí jedné buňce po několika iteracích vznikne opakující se fraktální vzor, jímž je v tomto případě Sierpińskiho trojúhelník. Při výchozí náhodné konfiguraci buněk vznikne na první pohled náhodná struktura sestávající z mnoha trojúhelníkových oken různých velikostí. Vzor vytvářený pravidlem 90 se objevuje od raného středověku na mnoha italských tapiseriích. [10]

Tabulka 5 - Pravidlo 90

111	110	101	100	011	010	001	000
0	1	0	1	1	0	1	0



Obrázek 12 - Vizualizace pravidla 90

4.2 Dvojrozměrné celulární automaty

Elementární celulární automaty dobře slouží k popisu základních vlastností a principu fungování CA, avšak pro reálnou aplikaci nenacházejí takového uplatnění jako celulární automaty o více dimenzích. Dalším logickým krokem je tedy přejít k popisu 2D CA, které jsou už v praxi značně užívanější.

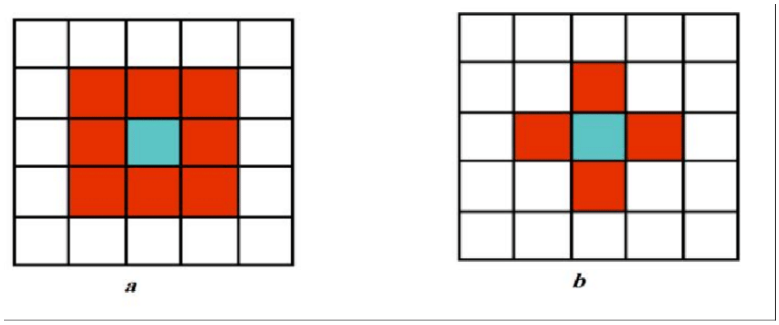
Obdobně jako v podkapitole 3.1.1, CA o dvou rozměrech v základu sestávají z pravidelné mřížky, která tvoří celý pracovní prostor. Pro jednoduchost budeme uvažovat situaci, kdy buňky mohou nabývat jednoho ze dvou stavů, a to 0 a 1, stejně jako v kapitole 4.1, a to podle daného pravidla označeného dále ϕ. [11]

Při uvažování dvou rozměrů automaticky dojde k rozšíření *okolí*. [10;11] Jsou zavedeny dva modely *okolí*. Prvním je *okolí* sestávající z 5 buněk, takzvané von Neumannovo, popsané rovnicí

$$a_{i,j}^{(t+1)} = \phi[a_{i,j}^{(t)}, a_{i,j+1}^{(t)}, a_{i+1,j}^{(t)}, a_{i,j-1}^{(t)}, a_{i-1,j}^{(t)}]$$

kde a je vybraná buňka na pozici (i,j) a t značí časový okamžik. Druhým je *okolí* sestávající z 9 sousedních buněk, takzvané Mooreovo, popsané rovnicí

$$a_{i,j}^{(t+1)} = \phi[a_{i,j}^{(t)}, a_{i,j+1}^{(t)}, a_{i+1,j}^{(t)}, a_{i+1,j+1}^{(t)}, a_{i,j-1}^{(t)}, a_{i-1,j}^{(t)}, a_{i-1,j-1}^{(t)}, a_{i+1,j-1}^{(t)}, a_{i-1,j+1}^{(t)}]$$



Obrázek 13 - a) Moorovo okolí b) von Neumannovo okolí

Pro zjednodušení se mnohdy neuvažují funkce beroucí v potaz všechny stavy všech sousedních buněk, ale bere se pouze suma všech hodnot sousedních buněk. V tomto případě se funkce zjednodušují na

$$a_{i,j}^{(t+1)} = f[a_{i,j}^{(t)} + a_{i,j+1}^{(t)} + a_{i+1,j}^{(t)} + a_{i,j-1}^{(t)} + a_{i-1,j}^{(t)}]$$

Rovnice výše platí pro von Neumannovo okolí a obdobná funkce platí pro Moorovo okolí. Tyto soubory pravidel jsou nazývány totalistickými, příkladem budiž Hra života, popsána níže. [10]

Při rozšíření pracovního prostoru prudce narůstá počet všech možných pravidel. Při uvažování jednoduchého okolí u elementárních CA bylo dosaženo maximálně 256 různých pravidel. Ve 2D, při uvažování von Neumannova okolí toto číslo naroste na 2^{32} možností, a při uvažování Mooreova okolí se dostaneme na soubor 2^{512} možností. Z tak enormního počtu jasně vyplývá, že nepřipadá v úvahu studovat každé jedno pravidlo

samostatně a zároveň přímá matematická analýza nepřináší uspokojivé výsledky pro predikci a popis vlastností jednotlivých pravidel. Namísto toho se užívá z velké části experimentální přístup, kdy se zvolí jedno pravidlo a proběhne přímá simulace CA, na jejímž základu proběhne analýza a aproximace obecných vlastností takového systému. Většinou je užito náhodných vzorků vybraných ze souboru všech pravidel s tím předpokladem, že jejich zjištěné vlastnosti jsou typickými pro spoustu dalších systémů. [10]

Stejně jako u elementárních celulárních automatů, i dvojrozměrné CA se dají kategorizovat do 4 celků, stejných jako v kapitole 4.1.1. [10]

Využití 2D CA nachází v mnohých oblastech, například dendritický růst krystalů vykazuje známky podobné 2D CA, dále Navier-Stokesova rovnice pro proudění kapalin může být modelována užitím vhodného pravidla. Dalším příkladem je cíl této práce, a to využití CA pro plánování cesty mobilního robotu. [10]

4.2.1 Hra života (Conway`s Game of Life)

Asi nejznámější příklad dvojrozměrného celulárního automatu je Hra života (angl. Game of Life, zkráceně GoL), jejímž tvůrcem je John Conway. [GoL není hrou v pravém slova smyslu, neexistují žádní hráči, uživatel má pouze možnost zadat počáteční podmínky a dál se celá simulace vyvíjí podle vlastních definovaných pravidel. [12]

Popularitu si tato hra získala hlavně kvůli podobnosti s běžně pozorovanými systémy v přírodě a okolním světě. Celý „životní cyklus“ takto modelovaného světa zahrnuje vše od vzniku, proměn až po zánik, stejně jako reálná společenství žijících organismů. [12] Svět se chová na základě určitého „genetického zákona“, na jehož základě jednotliví členové (buňky) přežívají, umírají, či se rodí nové. Tento „genetický zákon“ není nic jiného než specifické pravidlo jako u předešlých CA, pouze byl cíleně zvolen tak, aby splňoval následující podmínky:

- Neexistuje žádné počáteční seskupení buněk, pro které existuje jednoduchý důkaz, že populace buněk bude růst neomezeně
- Existují počáteční seskupení buněk, které rostou bez omezení
- Existují jednoduché počáteční seskupení buněk, které budou určitou dobu procházet změnami a růst, dokud se neustálí v jednom ze tří stavů, a to 1. úplně vyhynou (z důvodu přeplnění nebo příliš malé koncentrace); 2. dojdou do stabilního neměnného stavu; 3. začnou oscilovat mezi dvěma nebo více stavy donekonečna

Pravidlo GoL se může označit jako totalistické, protože nezáleží na umístění jednotlivých buněk, pouze jde o funkci sumy celého Mooreova okolí. [12] Pravidlo má potom toto znění (pro zjednodušení uvažujeme, že živá buňka má hodnotu 1, mrtvá buňka má hodnotu 0):

- Buňka přežije za předpokladu, že má právě 2 anebo 3 žijící sousedy
- Buňka zemře za předpokladu, že má 4 a více živých sousedů (z důvodu přemnožení) anebo když má 1 nebo 0 sousedů (z důvodu izolace)

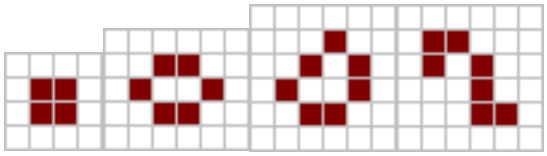
- Nová buňka bude narozena za předpokladu, že má mrtvá buňka právě 3 žijící sousedy

Toto pravidlo má za následek vytvoření emergentně-komplexního systému, tzn. že z velmi jednoduchých pravidel vznikne celé „živé univerzum“ s mnoha specifickými strukturami, z nichž některé budou dále popsány. [12;13]

Výběr některých struktur v Game of Life

Struktury objevující se v GoL se v některých případech, kdy se u nich objevují podobné vlastnosti, dají zařadit do tří kategorií, a to do stabilních struktur, oscilátorů a vesmírných lodí. [12; 13]

Stabilní struktury (angl. Still life) jsou takové konfigurace buněk, které se v závislosti na čase nemění, potažmo oscilují v rámci jedné generace. Stabilní struktury jsou častým jevem objevujícím se po několika generacích vycházejících z náhodného počátečního stavu. Nejčastější takovou strukturou jsou tzv. Blocks. Dalšími strukturami mohou být například *Hive*, *Loaf*, *Tub*, etc. Mohou se vyskytovat samostatně či ve skupinách stejných struktur. Speciálním příkladem stabilních struktur jsou takzvaní *Eaters*, tj. konfigurace buněk, jež jsou schopny při kontaktu s jinou letící strukturou tuto strukturu „pozřít“ a zpět se vrátit do svého původního stavu. Nejznámějším příkladem může být tzv. *Fish-hook*. [12;14]



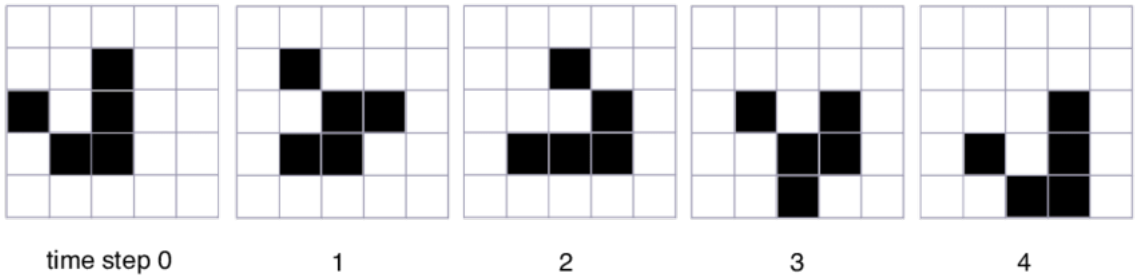
Obrázek 14 - Příklady stabilních struktur - zleva Block, Hive, Loaf, Fish-Hook

Oscilátory jsou taková uskupení buněk, která se po dvou a více generacích vrací opakovaně do výchozího stavu a tento proces opakují donekonečna. Perioda oscilátoru je počet generací, který zabere oscilátoru navrácení do původní konfigurace. Nejjednodušším oscilátorem je tzv. *Blinker*, tj. struktura o třech vodorovných (svislých) sousedech, která se v další generaci přemění na strukturu o třech svislých (vodorovných) sousedech, má tedy periodu = 2. Dalšími oscilátory jsou například *Star*, *Cross*, *Clock*, etc. [12;15]

Vesmírné lodě jsou takové CA struktury, které oscilují, ale zároveň jejich způsob vytváření nových a zanikání starých buněk způsobuje „pohyb“ v jednom směru. Periodou je opět nejmenší počet generací, kterými musí struktura projít, než se navrátí do původního tvaru. Další vlastností vesmírných lodí je jejich rychlost, vztahovaná k *c*. Rychlost je obecně definována jako počet buněk, o které se loď posune za svou periodu, dělená délkou periody. Tedy je-li perioda lodi 5 a během této periody se loď posune o 1 buňku, je její rychlost (1/5)**c*. *C* je potom maximální rychlost, jakou se loď může pohybovat, to je jedna buňka za jednu periodu, pokud je perioda = 1. [12;16]

Nejznámější vesmírnou lodí je tzv. *Glider*, tj. konfigurace 5 živých buněk s periodou = 4, cestující rychlostí rovnou *c*/4. Glidery mají velký význam z toho důvodu,

že jsou schopny přenášet informace na libovolné vzdálenosti a zároveň jsou jednoduše tvořeny jinými strukturami. Užitím Gliderů byla dokázána Turingovská kompletnost celulárních automatů. Vhodným „posíláním“ Gliderů na Blocks může být vytvořen jistý čítač. Správnou kombinací vysílaných Gliderů je možno vytvořit logické operátory jako AND, OR nebo NOT. [12;16]



Obrázek 15 - Příklad vesmírné lodi - Glider

Další speciální struktury zahrnují Zbraně (Guns) schopné neomezeně produkovat vesmírné lodě, jako Glidery, a vysílat je do prostoru, anebo speciální Vlaky, které za sebou nechávají stopu živých buněk. Posledním velkým objevem bylo vytvoření struktury Gemini, tj. kompletně sebe-replikující struktura, která po 33,6 milionech generací vytvoří svůj přesný klon, během čehož zničí svého předchůdce. Celý tento princip je postavený na velmi dlouhé „pásce“ Gliderů, které přenesou postupně celou informaci o původní struktuře, novou strukturu postaví a tu původní zároveň vymažou. [17]

5 Implementace vlastního kódu

Praktickou částí této práce bylo navržení a implementace vlastního algoritmu plánování cesty mobilního robotu na bázi celulárních automatů. Jako vývojového prostředí bylo užito programu MATLAB. MATLAB byl zvolen hlavně ze dvou důvodů: jedním jsou osobní zkušenosti s tímto programem a tím druhým je dobrá implementace práce s maticemi, což v mnohém značně zjednodušuje vytváření celulárních automatů. Nevýhodou je složitější vytváření GUI (grafických uživatelských rozhraní). Proto byl při návrhu algoritmu namísto nich využit program Microsoft Excel, v němž byly vytvořeny tabulky zadávající počáteční stav prostředí (počátek, překážky, cíl).

5.1 Algoritmus plánování cesty užitím CA

Většina algoritmů pro plánování tras (a všechny doposud zmíněné v kapitole 3 této práce), jsou globálními algoritmy. To znamená, že vyhledávání probíhá na předem známé statické ploše, avšak při přidání dynamických překážek do pracovního prostoru by tyto způsoby výpočtu selhaly.

Jedním z méně užívaných a prozkoumaných způsobů je použití CA pro plánování cesty. V posledních několika letech bylo této problematice věnováno několik článků a prací, primárně na konferencích TAROS (2018, 2014). V této práci implementovaný algoritmus nespadá čistě do globálních ani lokálních plánovačů, protože je předem v pracovním prostoru známá absolutní poloha startu i cíle cesty, výpočet ale probíhá za chodu programu a další kroky jsou voleny adekvátně k okamžitému stavu soustavy, což umožňuje bezproblémové zavedení jak statických, tak pohybujících se překážek. Z důvodu špatné prezentace animovaných výsledků v tištěné podobě budou v další kapitole provedeny srovnávací testy pouze ve statickém prostoru.

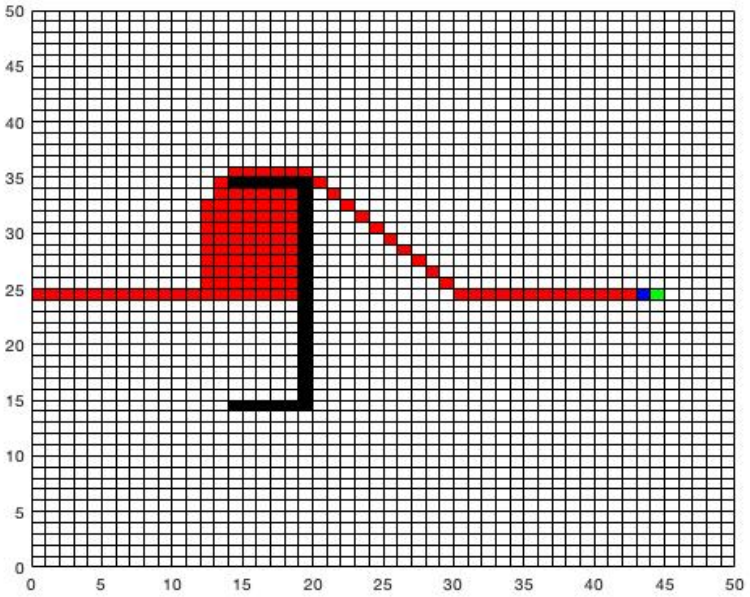
5.1.1 Princip algoritmu

Zjednodušený princip celého programu spočívá v tom, že samotný plánovač postavený na předvedeném matematickém modelu ve své základní podobě sice vyhledá cestu ze začátku do konce, ale cesta to není v mnoha případech zdaleka ideální. Z tohoto důvodu v první fázi (v prvním programu) proběhne výpočet prvotní cesty, která je v druhé fázi předána druhému algoritmu, který na podobném principu jako první část sleduje cestu zpětně od cíle do počátku a tím navrhne mnohem lepší finální trajektorii, viz příklad – obrázky 15 a 16 na další straně.

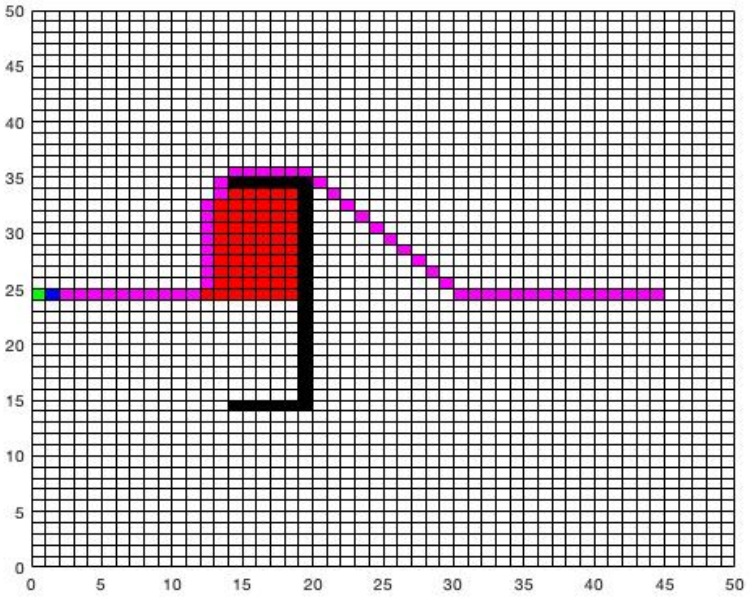
Na obrázcích je též jasně vidět rozložení celého prostoru. Naší pracovní plochou je mřížka o rozměrech 50 × 50 prázdných buněk, matematicky popsaná maticí nul o stejném rozměru. Dalšími významné prvky:

- Cílový bod – zelená barva – matematická hodnota = 1
- Aktuální poloha robota – modrá barva – hodnota = 2

- Překážky – černá barva – hodnota = 3
- Doposud vykonaná trasa – červená barva – hodnota = 4
- Zpětně optimalizovaná trasa – fialová barva – hodnota = 5



Obrázek 16 - Výsledek po první fázi



Obrázek 17 - Výsledek po druhé fázi

Z předchozího odstavce tedy vyplývá, že při modelování chování robota se nelze uspokojit pouze se dvěma možnými stavy buňky, ale pro náš případ je jich užito dohromady 6 (zahrnujíc volný prostor = 0).

V algoritmu je užíváno již zavedeného Moorova okolí, to z důvodu předpokládaných přesnějších výsledků, které zároveň lépe kopírují reálnou situaci.

Samotná rozhodovací funkce (v předchozích kapitolách též pravidlem nebo rule-setem CA) vychází čistě z euklidovské geometrie. Při zanedbání veškerých překážek, například z důvodu špatné viditelnosti, je nejkratší cestou ze startu do cíle přímka spojující tyto dva body. V globálním souřadnicovém systému se potom dá velikost této přímky vypočítat z Pythagorovy věty jako

$$d = \sqrt{(g_x - r_x)^2 + (g_y - r_y)^2}$$

Souřadnice (g_x,g_y) jsou souřadnicemi cílového bodu; (r_x,r_y) jsou souřadnice robota v daný okamžik.

V každém okamžiku pro robota existuje osm sousedních (potenciálně dceřiných) buněk. Pro každou z těchto buněk je vypočítána hodnota absolutní vzdálenosti d od cíle, načež jsou tyto buňky seřazeny vzestupně a je defaultně vybrána buňka s nejmenší vzdáleností k cíli jako potenciální další poloha robota. Následně proběhne kontrola, zda-li nemá takto zvolená buňka už přiřazenou hodnotu 3 nebo 4, tj. není překážkou nebo některou z předchozích pozic robota. Pokud splňuje zvolená buňka tento předpoklad, pak je zvolena jako následník. Zároveň s přesunem robota do nové polohy je původní rodič označen hodnotou 4. Pokud by zvolená buňka nesplňovala jeden z těchto požadavků, pak je zvolena buňka s druhou nejmenší vzdáleností k cíli, atd.

Je důležité označovat buňky, kde se již robot nacházel, nejen z důvodu zpětného sledování trasy, ale hlavně kvůli podchycení možnosti, kdy se robot zacyklí v oscilování mezi dvěma body podél překážky.

Celý tento proces se opakuje do doby, než se robot na „dotykovou vzdálenost“ přiblíží k žádanému cíli, načež se cyklus zastaví a do paměti se uloží takto vytvořená matice. Ve druhé fázi se tato matice vezme jako počáteční stav s mírnými úpravami, který podle obdobného principu sleduje cestu z cíle do startu. Jediné dva rozdíly mezi těmito programy jsou zadané počáteční podmínky (výměna startu a cíle, nový pracovní prostor) a rozhodovací funkce pro výběr následující polohy robota, kdy se volí vedle té buňky s nejmenší vzdáleností k cíli právě jedna z buněk, která se nachází ve stavu 4. Robot potom za sebou nechává obdobně „stopu“ buněk o hodnotě 5, tyto ukazují finálně vytvořenou trasu.

Pozn.: Při uvažování tohoto algoritmu jako lokálního plánovače by samozřejmě robot prošel celou červenou trasu, fialovou zpětnou by absolvoval pouze v případě, kdy bychom chtěli, aby se ze cíle navrátil do počátku. Přesto je tato trasa zahrnována, kvůli již zmíněné optimalizaci potenciální trasy.

6 Prezentace výsledků srovnávacích experimentů

Po navržení a vytvoření daného algoritmu bylo dalším krokem provedení alespoň obrazných srovnávacích experimentů pro porovnání s již známým algoritmem. Jako vhodná volba se jevil algoritmus A* pro jeho rozšířenost a univerzalitu. Experimenty samotné proběhly jednak pro CA algoritmus přímo v MATLABu na osobním počítači, jednak pro A* algoritmus na internetovém serveru poskytujícím webovou aplikaci pro srovnání a ukázku algoritmů sledování cesty.[3]

V případě CA bylo voleno pracovní prostředí (tedy mřížka) o velikosti 40 × 40 buněk. Pro zjednodušení zadávání celého prostoru bylo použito pěti tabulek vytvořených v MS Excel (všechny jsou zahrnuty v přílohách). Tyto tabulky / konfigurační prostory jsou seřazeny vzestupně od nejjednoduššího po nejsložitější, viz samotné experimenty. V případě A* bylo užito Octile metriky a byl umožněn pohyb v diagonálním směru.

6.1 Jednotlivé experimenty

Celkem bylo provedeno pět srovnávacích měření. Pro délku trasy bylo uvažováno 1 pro pohyb doleva / doprava / nahoru / dolů a √2 pro pohyb diagonální. Časové srovnání objektivně nemělo smysl z důvodu dvou naprosto odlišných výpočetních zařízení, tudíž nebude uváděno popřípadě bráno v potaz. Počtem operací uvažujeme to, kolikrát proběhne rozhodovací algoritmus (v případě CA smyčka while, která určuje další vhodný krok) během chodu programu než je dosaženo cíle.

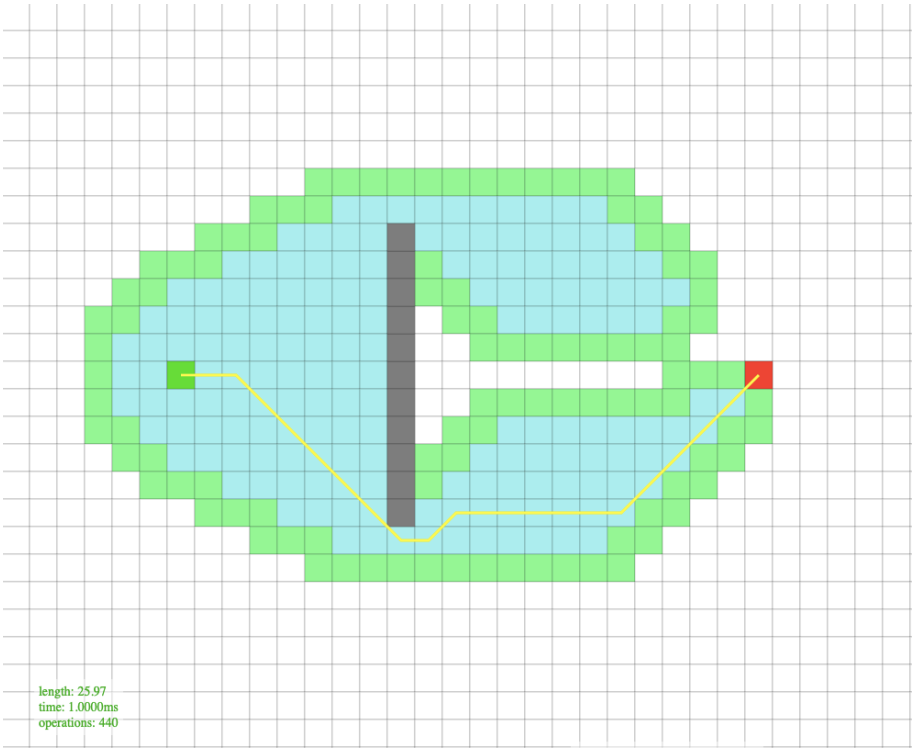
Na obrázcích znázorňujících výsledky A* algoritmu je několik typů buněk (viz popis A* v kapitole 3.1.1). Na obrázcích jsou zelené buňky, které spadají do Opensetu a světle modré buňky reprezentující Closedset.

6.1.1 První mapa

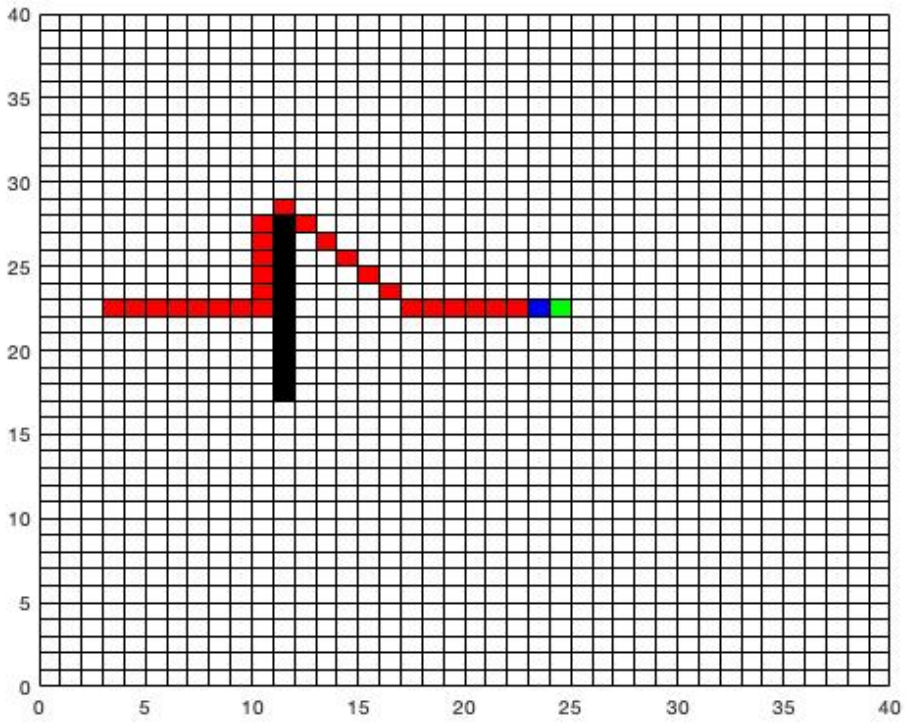
První uvedená mapa obsahovala pouze jednoduchou svislou překážku, kde není vynakládána náročnost na výpočetní sílu nebo univerzalitu algoritmu, jsou tím pádem velmi podobné výsledky obou měření, viz obrázky 17, 18 a 19.

Jak je vidno z tabulky 6, algoritmus A* našel kratší cestu, ale to za cenu mnohem většího počtu operací, a to i přesto, že je sečten celkový počet operací prvního i zpětného chodu. Podobný trend můžeme sledovat i u následujících experimentů.

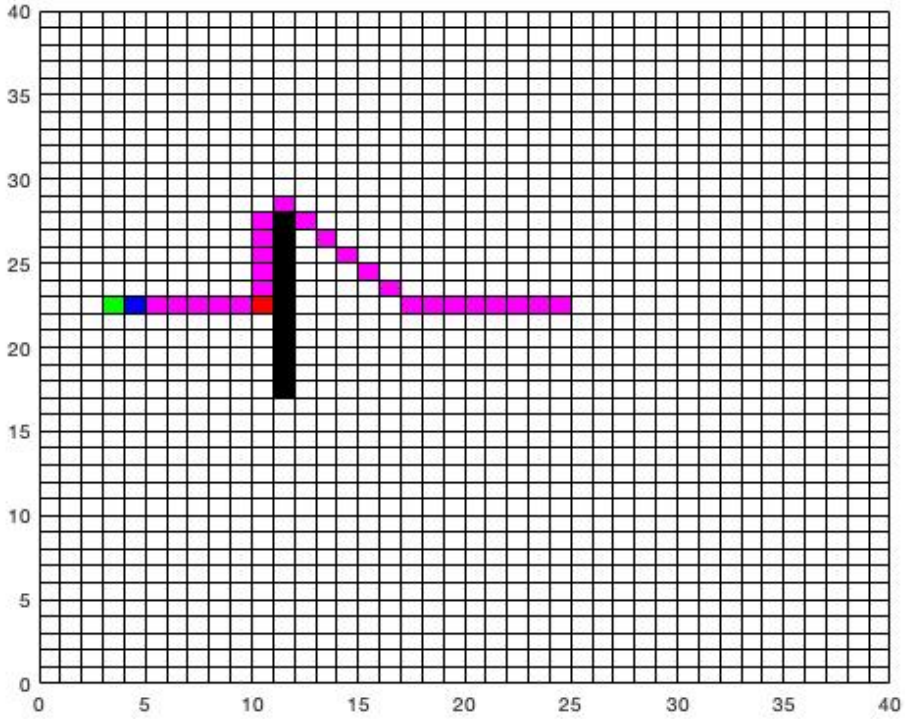
[3] Odkaz na webovou aplikaci: http://qiao.github.io/PathFinding.js/visual/



*Obrázek 18 - Výsledek 1. experimentu užitím A**



Obrázek 19 - Výsledek 1. experimentu CA



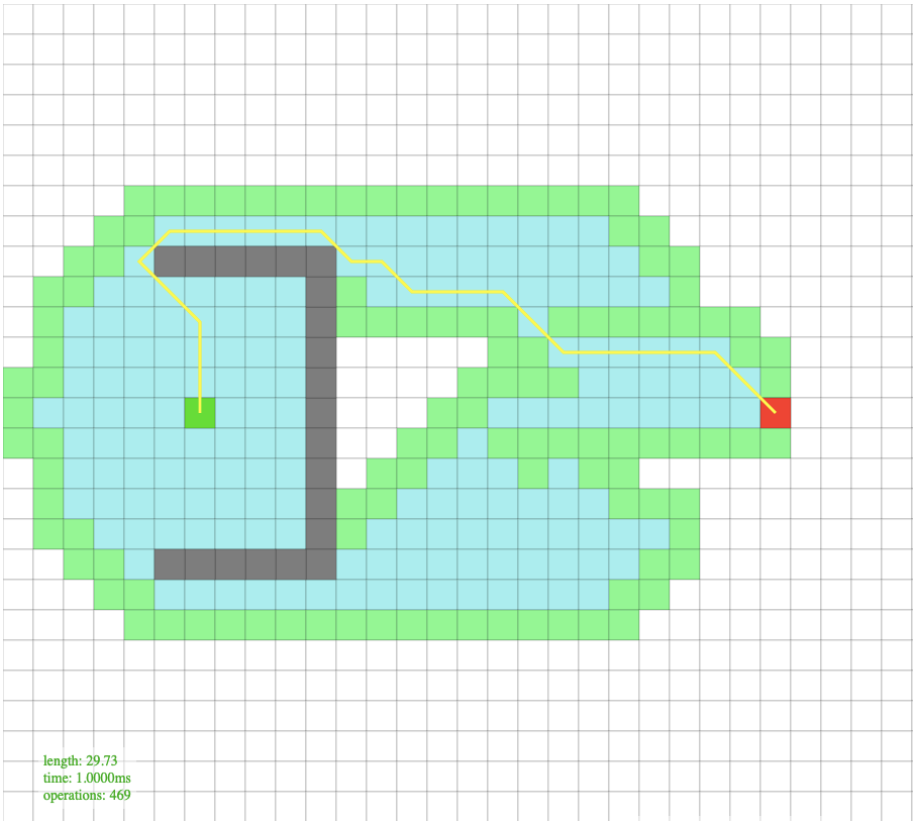
Obrázek 20 - Výsledek 1. experimentu CA - zpětný chod

Tabulka 6 - Výsledky 1. experimentu

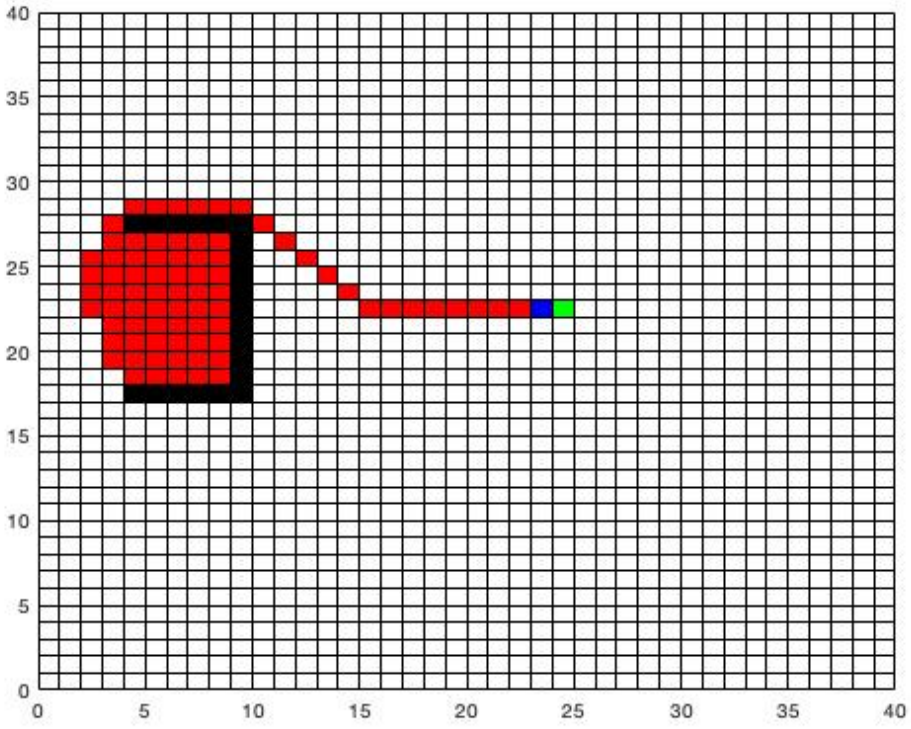
1. mapa	Délka	Počet operací
A*	26	440
CA	27,9	46
CA zpětný chod	27,3	46

6.1.2 Druhá mapa

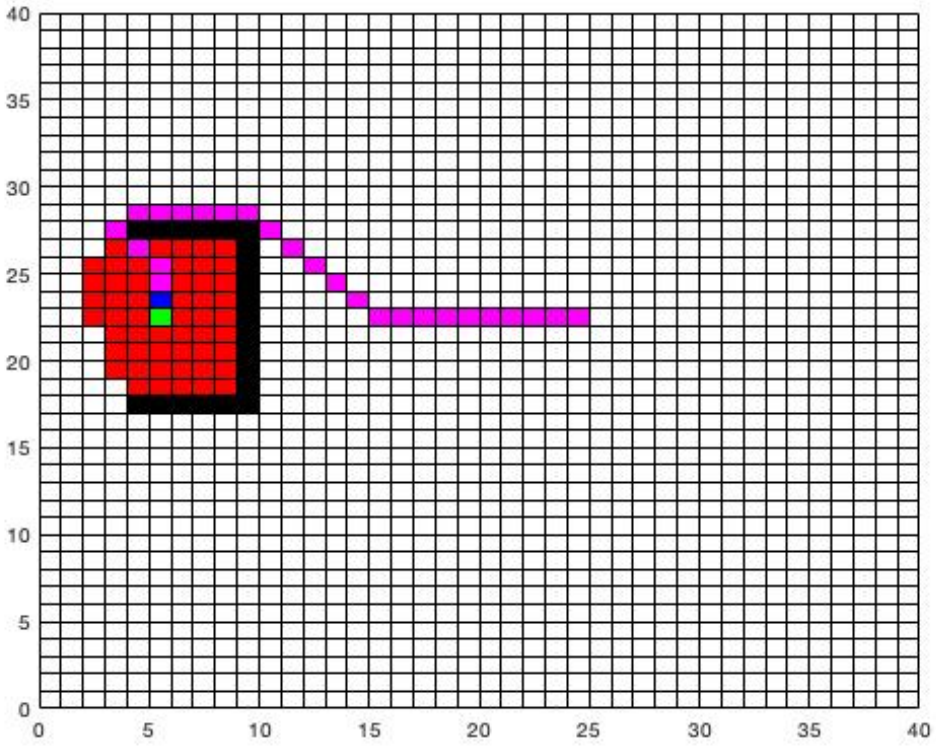
Druhá mapa je opět velmi jednoduchou mapou o jedné překážce, ale je zahrnuta z toho důvodu, že zrovna náš CA algoritmus má s tímto typem překážek problém, kdy prozkoumá nejprve celou vnitřní část a až poté se dostane za překážku, jak je vidno na obrázku 21. Zhodnocení výsledků je v tabulce 7, opět pozorujeme stejný trend větší vzdálenosti při CA algoritmu, ale mnohem menšího počtu operací. Start na obrázku 21 není vidět, ale můžeme jej pozorovat na obrázku 22, jedná se o bod zhruba uprostřed prostoru ohraničeného překážkou..



*Obrázek 21 - Výsledek 2. experimentu užitím A**



Obrázek 22 - Výsledek 2. experimentu užitím CA



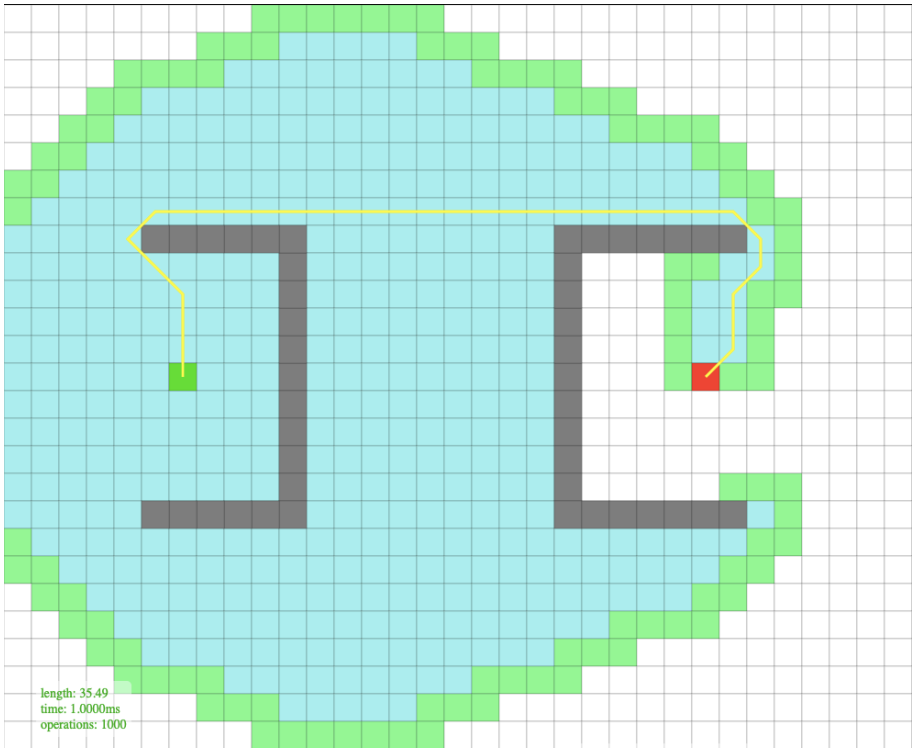
Obrázek 23 - Výsledek 2. experimentu CA - zpětný chod

Tabulka 7 - Výsledky 2. experimentu

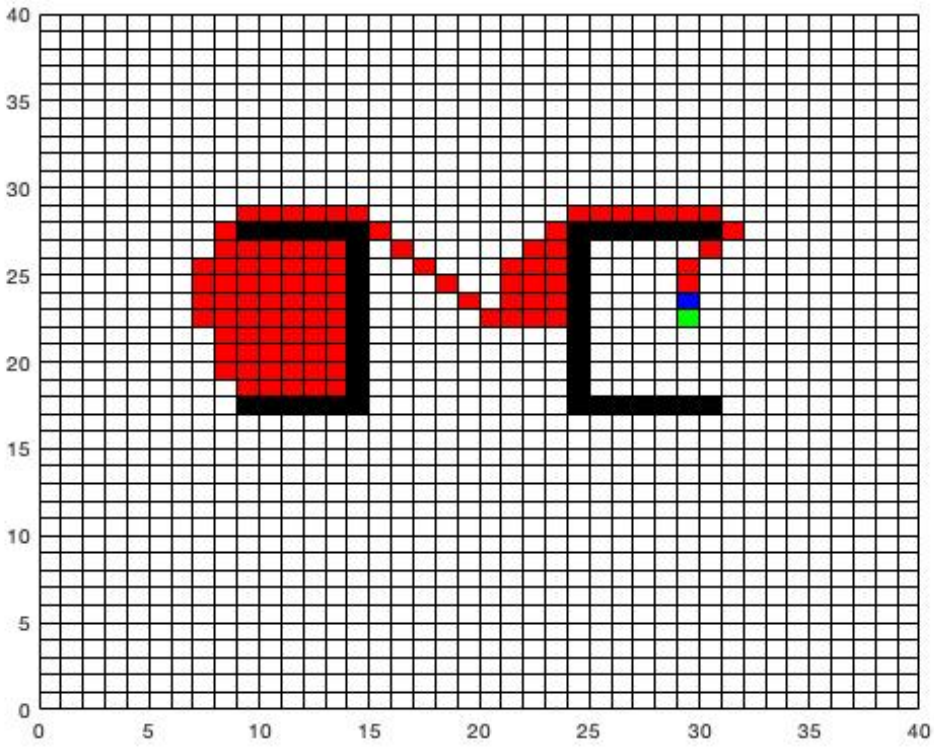
2. mapa	Délka	Počet operací
A*	29,7	469
CA	86,50	283
CA zpětný chod	28,7	53

6.1.3 Třetí mapa

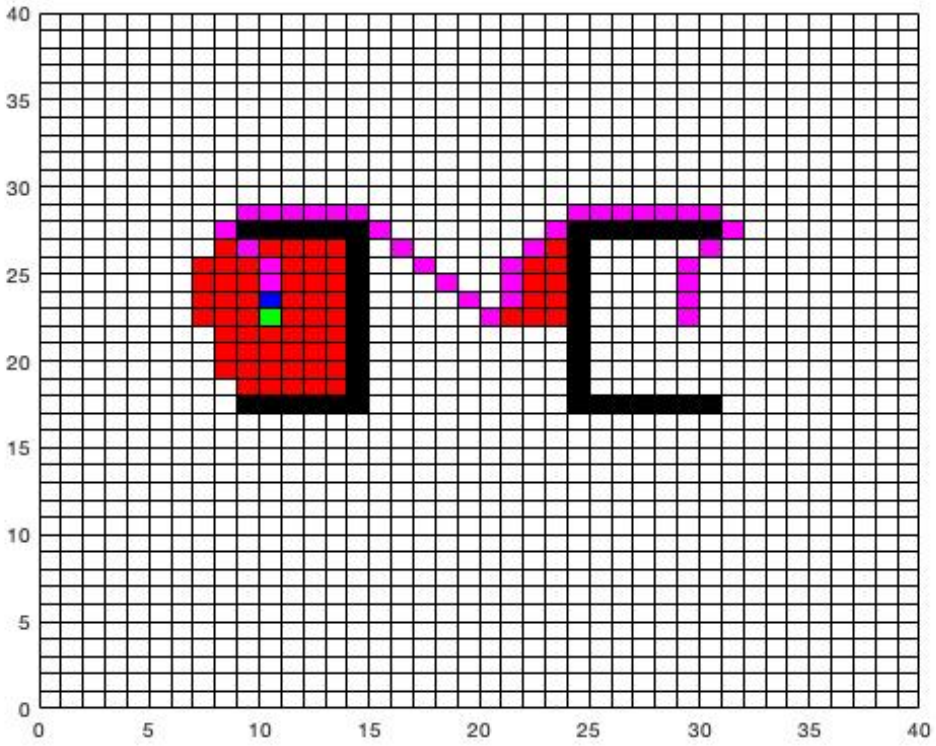
U třetí mapy částečně narostla náročnost, ale stále se jedná o velmi jednoduché prostředí. Rozdílem mezi druhou a třetí mapou je pouze přidání zrcadlově otočené překážky okolo cílového bodu. Nejkratší cestu ukazují obrázky 23, 24, 25 a tabulka 8 shrnuje výsledky třetího experimentu.



*Obrázek 24 - Výsledek 3. experimentu užitím A**



Obrázek 25 - Výsledek 3. experimentu CA



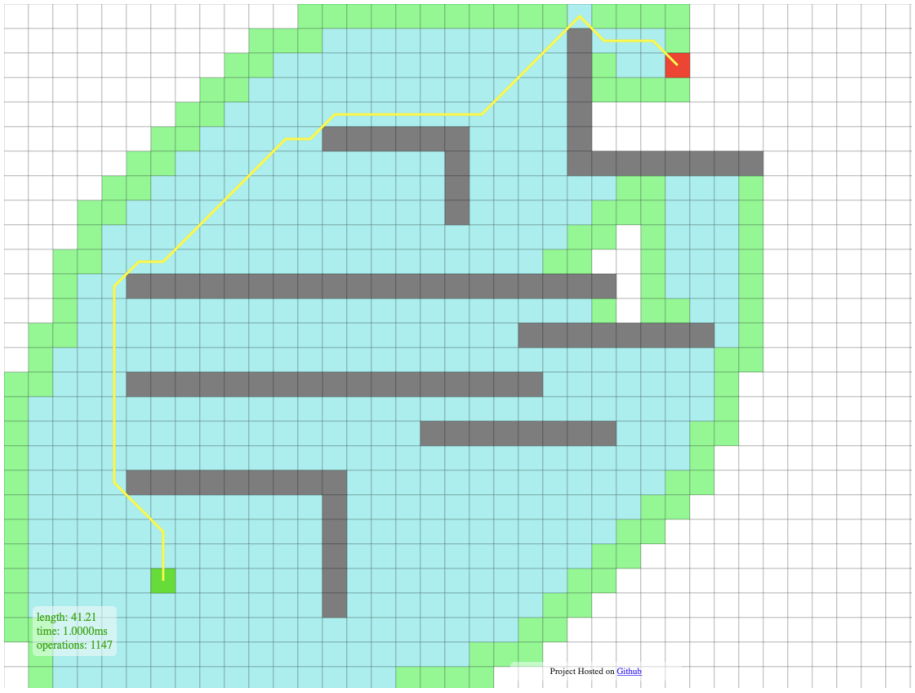
Obrázek 26 - Výsledek 3. experimentu CA - zpětný chod

Tabulka 8 - Výsledky 3. experimentu

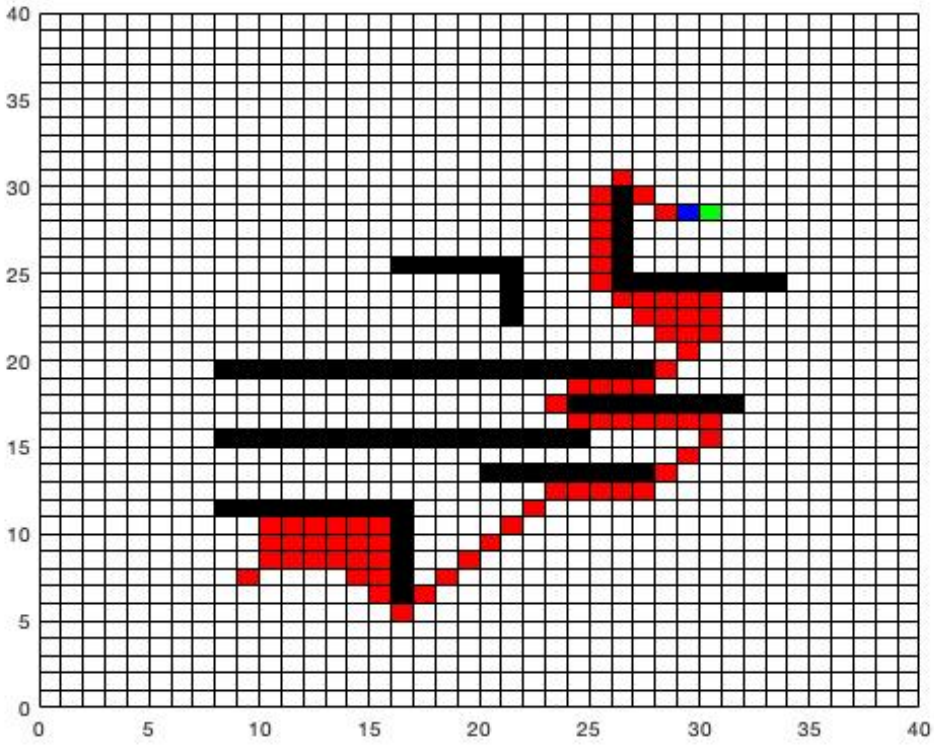
3. mapa	Délka	Počet operací
A*	35,5	1000
CA	108,4	366
CA zpětný chod	40,6	101

6.1.4 Čtvrtá mapa

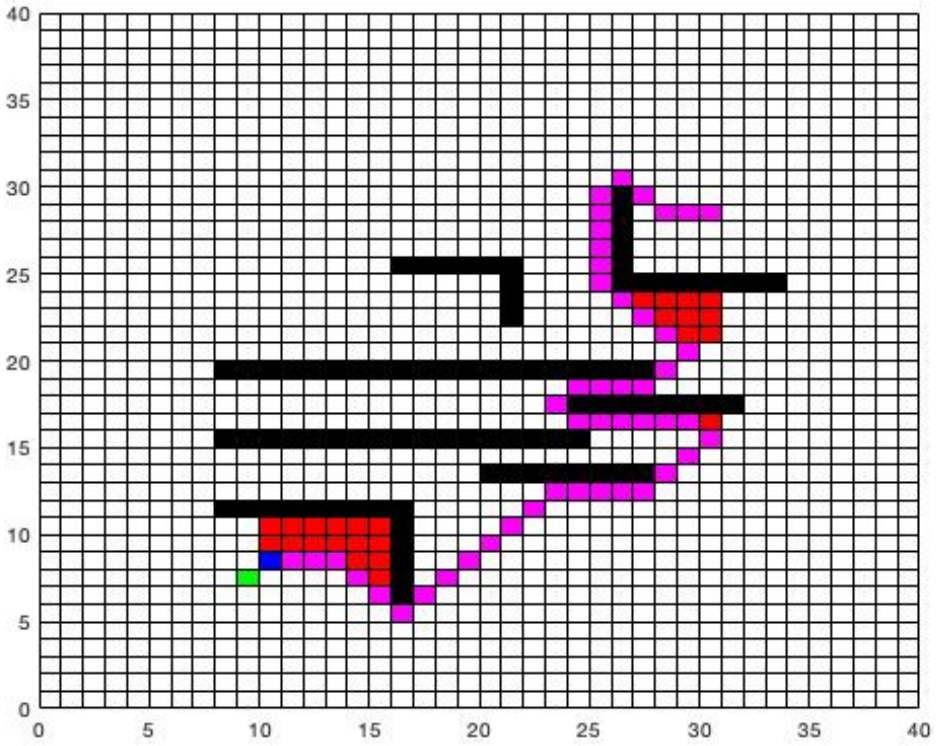
Čtvrtá mapa je obsahuje první náročnější prostor s překážkami náhodněji rozmístěnými po prostoru mezi cílem a koncovým bodem, přesto pozorujeme stále stejný trend, kdy algoritmus A* nachází kratší cestu za cenu většího počtu operací. Nalezené cesty jsou vidět na obrázcích 26, 27, 28 a výsledky jsou ukázány v tabulce 9.



*Obrázek 27 - Výsledek 4. experimentu užitím A**



Obrázek 28 - Výsledek 4. experimentu CA



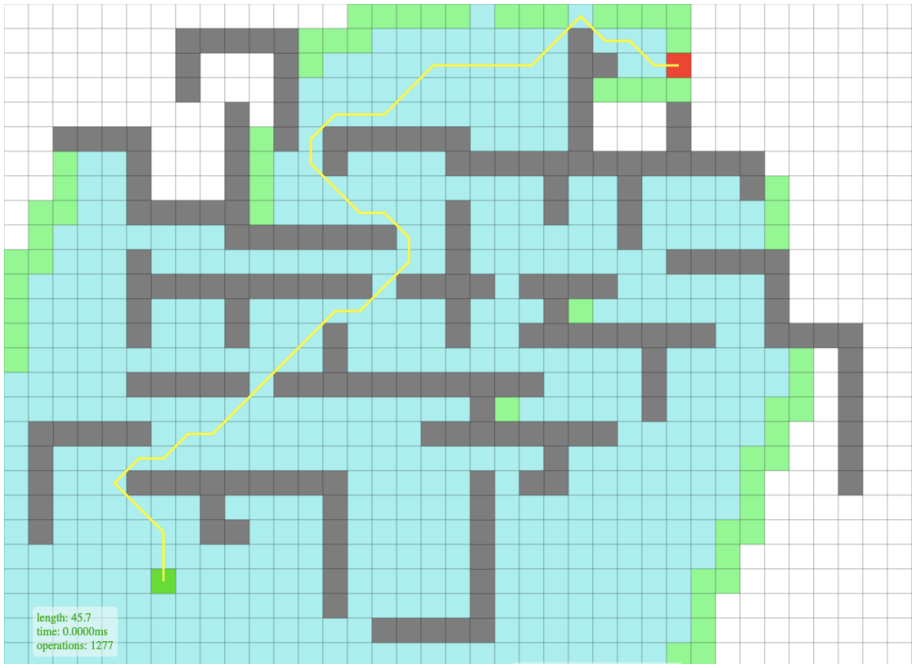
Obrázek 29 - Výsledek 4. experimentu CA - zpětný chod

Tabulka 9 - Výsledky 4. experimentu

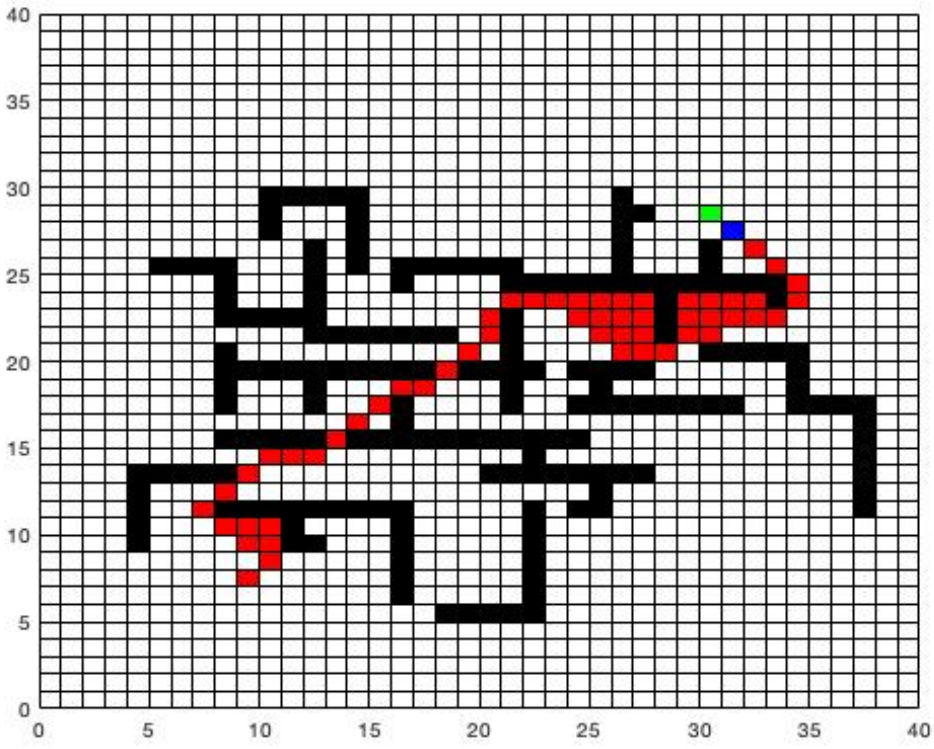
4. mapa	Délka	Počet operací
A*	41,21	1147
CA	84,8	238
CA zpětný chod	57,35	131

6.1.5 Pátá mapa

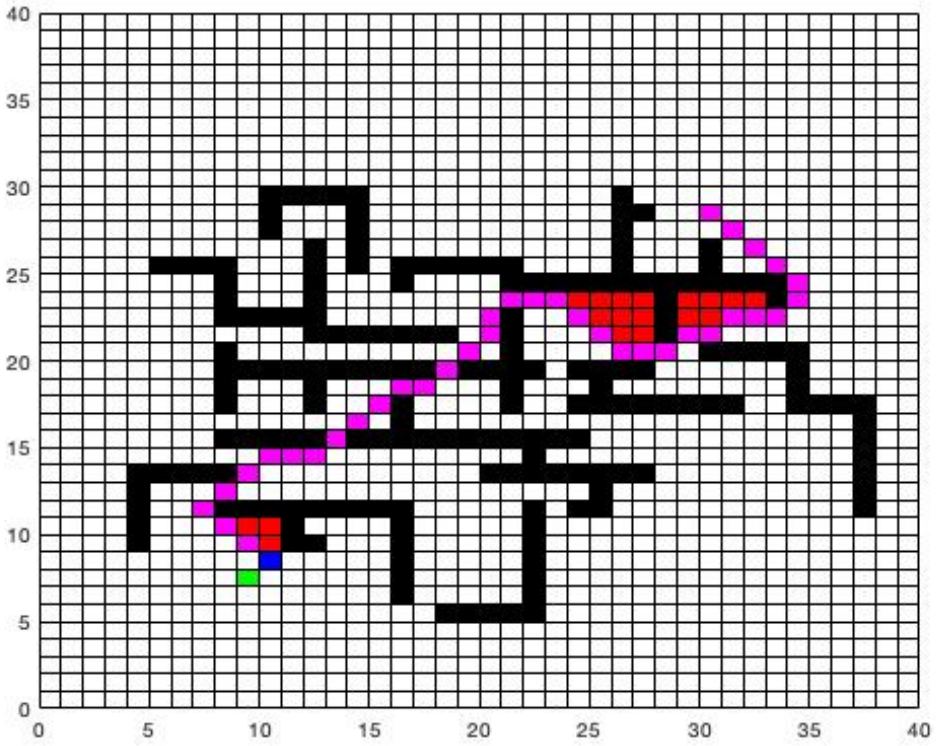
Poslední experiment proběhl na nejkomplexnější mapě, která byla těžkou modifikací předchozí (4.) mapy. I přesto, že působí na první pohled mnohem složitěji, pro oba dva vyhledávací algoritmy se jevila jako jednodušší jak co se počtu kroků, tak celkové vzdálenosti týče. Trend, kdy A* nachází kratší cestu za cenu většího počtu operací se dále potvrzuje. Zajímavé je, že při zpětném chodu se délka cesty A* a CA jeví až na desetiny stejně dlouhá. Zobrazení a výsledky můžeme vidět na obrázcích 29, 30, 31 a v tabulce 10 níže.



*Obrázek 30 - Výsledek 5. experimentu užitím A**



Obrázek 31 - Výsledek 5. experimentu CA



Obrázek 32 - Výsledek 5. experimentu CA - zpětný chod

Tabulka 10 - Výsledky 5. experimentu

5. mapa	Délka	Počet operací
A*	45,7	1277
CA	64,8	160
CA zpětný chod	45,9	84

6.2 Zhodnocení experimentů

Všechny dosažené výsledky ukazují na superioritu algoritmu A* při hledání optimální cesty ve známém prostředí, za předpokladu, kdy výpočetní síla není nijak omezena. Navzdory původním předpokladům si ale CA algoritmus nevede špatně, vezmeme-li navíc v úvahu zpětný chod, neboli optimalizaci již nalezené cesty. V případech 1., 2. a 5. experimentu totiž takto nalezená trasa je řádově maximálně v jednotkách rozdílná od trasy nalezené A* algoritmem.

7 Závěr

Cílem této bakalářské práce bylo provést rešerši na téma hledání cesty a celulárních automatů, což bylo vypracováno v kapitolách 3 a 4. Dalším cílem bylo navržení a implementace vlastního algoritmu hledání cesty na bázi celulárních automatů. Tato část je obsažena v kapitole 5, kde je popsáno základní fungování navrženého algoritmu. Poslední (6.) kapitola poté dává do srovnání proběhlé experimenty na několika mapách s výsledky vyhledávacího algoritmu A*

Můj prvotní předpoklad, že takto navržený algoritmus nebude v porovnání se známým A* dosahovat dobrých výsledků byl vyvrácen, naopak při některých konfiguracích se jeví jako podobně dobrou volbou. Uvážíme-li k tomu fakt, že se jedná o algoritmus, jenž si je schopen poradit s dynamickými překážkami a proměnlivým prostředím, může být v některých případech považován za vhodnější než některé jiné čistě globální algoritmy.

Problémem tohoto algoritmu je potenciální riziko zaseknutí se v některých rozích, protože všechny do té doby navštívené buňky se uzavřou do stavu, kdy se do nich robot nemůže vrátit. V našem případě bylo této skutečnosti ale využito spíše ku prospěchu, zamezilo se tak oscilování na hranách některých překážek a zároveň takovéto značení předchozích poloh se dá využít ke zpětnému sledování cesty ke startu z cíle a s tím spjatá optimalizace potenciální trasy.

Do budoucna by bylo vhodné vyřešit výše zmíněný problém „motání se“ robota okolo některých překážek, popřípadě potenciální riziko zaseknutí se. Dalším krokem by mohlo být vytvoření plně funkčního uživatelského rozhraní pro snadnější zadávání překážek, možnosti přidání dynamických překážek uživatelem a jednodušší zadání startu a cíle.

Seznam obrázků

Zdroje použitých obrázků

Obr. 1	https://www.researchgate.net/figure/Hexagon-route-elements-marked-as-green-triangles-in-result-of-A-pathfinding-within_fig7_310497498 https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html
Obr. 2	http://www.smartroboticsys.eu/?p=2849&lang=en
Obr. 3	https://en.wikipedia.org/wiki/Chebyshev_distance
Obr. 4	https://cs.wikipedia.org/wiki/A*
Obr. 5	https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf
Obr. 6	https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf
Obr. 7	https://en.wikipedia.org/wiki/File:Graph1sivia.jpg
Obr. 8	S. Wolfram – A new kind of science – Chapter 2, p. 25
Obr. 9	https://www.wolframalpha.com/input/?i=rule+110
Obr.10	https://en.wikipedia.org/wiki/Rule_30
Obr.11	https://en.wikipedia.org/wiki/Rule_30
Obr.12	https://www.wolframalpha.com/input/?i=rule+30
Obr.13	https://www.researchgate.net/figure/a-Moore-neighborhood-b-Von-Neumann-neighborhood_fig6_262844415
Obr.14	https://en.wikipedia.org/wiki/Still_life_(cellular_automaton)
Obr.15	https://www.researchgate.net/figure/Subsequent-stages-of-the-glider-pattern-on-Conways-Game-of-Life-cellular-automaton-grid_fig1_263596638

Seznam použité literatury

[1] KLOBUŠNÍKOVÁ, Z. Plánování cesty mobilního robotu, Brno, Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018, 62 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, CSc.

[2] Motion Planning. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Motion_planning

[3] A*. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/A*_search_algorithm

[4] A*. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/A*

[5] CHOSET, Howie. *Robotic Motion Planning: Potential Functions* [online]. In: . 2007 [cit. 2019-05-19]. Dostupné z https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf

[6] DESROCHERS, Benoît. *Motion planning using interval analysis* [online]. ENSTA Bretagne, 2014 [cit. 2019-05-19]. Dostupné z: https://www.ensta-bretagne.fr/jaulin/rapport_desrochers_projetindus.pdf. Zpráva o průmyslovém projektu. ENSTA Bretagne. Vedoucí práce Luc Jaulin.

[7] ARANIBAR, Dennis Barrios a Pablo Javier ALSINA. *Reinforcement Learning-Based Path Planning for Autonomous Robots* [online]. Campus Universitário [cit. 2019-05-19]. Dostupné z: https://pdfs.semanticscholar.org/4b42/a803f54694b8b317b8d887ec9dcd59a9c397.pdf. Universidade Federal do Rio Grande do Norte.

[8] *The Mathematics of Models of Reference* [online]. [cit. 2019-05-19]. Dostupné z: http://www.mmdr.it/provaEN.asp

[9] WOLFRAM, Stephen. *A new kind of science* [online]. Champaign, IL: Wolfram Media, c2002 [cit. 2019-05-19]. ISBN 15-795-5008-8. Dostupné z: https://www.wolframscience.com/nks/

[10] Cellular Automaton. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Cellular_automaton

[11] PACKARD, Norman H. a Stephen WOLFRAM. Two-dimensional cellular automata. *Journal of Statistical Physics*[online]. 1985, **38**(5-6), 901-946 [cit. 2019-05-19]. DOI: 10.1007/BF01010423. ISSN 0022-4715. Dostupné z: http://link.springer.com/10.1007/BF01010423

[12] Conway's Game of Life. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

[13] GARDNER, Martin. Mathematical Games. *Scientific American*. 1970, **223**(4), 120-123. DOI: 10.1038/scientificamerican1070-120. ISSN 0036-8733. Dostupné také z: http://www.nature.com/doifinder/10.1038/scientificamerican1070-120

[14] Still Life (cellular automaton). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Still_life_(cellular_automaton)

[15] Oscillator (cellular automaton). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Oscillator_(cellular_automaton)

[16] Spaceship (cellular automaton). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2019-05-19]. Dostupné z: https://en.wikipedia.org/wiki/Spaceship_(cellular_automaton)

[17] COOK, Matthew. *Universality in Elementary Cellular Automata* [online]. Caltech, Pasadena, California, 2004 [cit. 2019-05-19]. Dostupné z: http://wpmedia.wolfram.com/uploads/sites/13/2018/02/15-1-1.pdf. Caltech.

[18] MARTINS, L. G. A. et al. An improved robot path planning model using cellular automata. In: Giuliani, M. et al. (eds.) TAROS 2018, LNAI, vol. 10965, pp. 183-194, 2018.

[19] FERREIRA, G. B. S., VARGAS, P. A., OLIVEIRA, G. M. B. An improved cellular automata-based model for robot path-planning. In: Mistry, M. et al. (eds.) TAROS 2014, LNAI, vol. 8717, pp. 25-36, 2014.

[20] SYED, U. A., KUNWAR, F. Cellular automata-based real-time path-planning for mobile robots. International Journal of Advanced Robotic Systems, vol. 11, no. 7, 2014.