Vizualizace algoritmů pro soutěž Micromouse

Slavomír Pagerka

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Slavomír Pagerka**
Osobní číslo:	**A15613**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Vizualizace algoritmů pro soutěž Micromouse**
Téma anglicky:	**The Visualisation of Algorithms for the "Micromouse" Competition**

Zásady pro vypracování:

1. **Prostudujte existující algoritmy hledání nejrychlejší cesty v bludišti pro soutěž Micromouse.**
2. **Vyberte vhodný programovací jazyk a knihovny pro vizualizaci takových algoritmů.**
3. **Navrhněte architekturu simulátoru, která umožní vizualizovat různé algoritmy pro Micromouse s cílem přiblížit jejich funkci ve výuce kurzů programování.**
4. **Implementujte simulátor ve vybraném programovacím jazyce.**
5. **Otestujte simulátor s několika různými algoritmy, popř. navrhněte algoritmus vlastní.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **LACKO, Branislav, ed. Šedesát let kybernetiky. Brno: Akademické nakladatelství CERM pro Asociaci strojních inženýrů, 2009. ISBN 978-80-7204-662-1.**
2. **ICFCC 2009: International Conference on Future Computer and Communications : proceedings, 3-5 April 2009, Kuala Lumpar [sic], Malaysia. Los Alamitos, Calif.: IEEE Computer Society, c2009. ISBN 978-0-7695-3591-3.**
3. **SITIS 2008: 4th International Conference on Signal Image Technologies and Internet Based Systems : proceedings : November 30 - December 3, 2008, Dynasty Resort, Bali, Indonesia. Los Alamitos, Calif: IEEE Computer Society, c2008. ISBN 978-0-7695-3493-0.**
4. **SPONSORED BY THE UNIVERSITY OF BRADFORD, IEEE, IEEE COMPUTER SOCIETY a IEEE TCSC .. Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. Piscataway: IEEE, 2010. ISBN 9781424475476.**
5. **EDITED BY SABINA JESCHKE, Honghai LIU a Daniel SCHILBERG. Intelligent robotics and applications 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, proceedings. Berlin: Springer, 2011. ISBN 9783642254895.**

Vedoucí bakalářské práce:	**Ing. Tomáš Dulík, Ph.D.** Ústav informatiky a umělé inteligence
Datum zadání bakalářské práce:	**15. prosince 2017**
Termín odevzdání bakalářské práce:	**25. května 2018**

Ve Zlíně dne 15. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Jméno, příjmení:

Název bakalářské/diplomové práce:

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 23.5.2018

.....
podpis diplomanta

ABSTRAKT

Cieľom tejto bakalárskej práce bolo vytvoriť aplikáciu, ktorá vizualizuje základné časti algoritmov používaných pri súťaži Micromouse. V teoretickej časti zhrniem základné pravidlá tejto súťaže a popíšem niektoré často využívané grafové algoritmy. V praktickej časti je popísaná architektúra aplikácie, jej grafické používateľské rozhranie a niekoľko príkladov jej využitia. Zdrojový kód aplikácie bude dostatočne zdokumentovaný a výstižný aby užívateľ mohol jednoducho implementovať napr. svoje algoritmy.

Kľúčové slová: Micromouse, Bludisko, Algoritmy hľadania najkratšej cesty, Java

ABSTRACT

The aim of this bachelor thesis is to create application that visualize the basic parts of algorithms used in Micromouse competition. In theory section, I will summarize basic rules of this competition and describe some most commonly used graph algorithms. In method section, I described architecture of my application and it’s graphical user interface and some use cases. Source code of the application will be sufficiently documented and straightforward in order to allow user implement his own algorithms for instance.

Keywords: Micromouse, Maze, Algorithm for finding the shortest paths, Java

Rád by som poďakoval vedúcemu svojej práce pánovi Ing. Tomášovi Dulíkovi, Ph. D. za konzultácie a rady, ktoré mi pomohli pri písaní tejto práce.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 SÚŤAŽ MICROMOUSE **11**
1.1 HISTÓRIA 11
1.2 PRAVIDLÁ SÚŤAŽE 12
1.3 BLUDISKO 13
2 KONŠTRUKCIA ROBOTA **15**
2.1 MIKROPOČÍTAČ 15
2.2 GYROSKOP 15
2.3 SENZORY 16
2.4 MOTORY 16
2.4.1 Jednosmerné motory 16
2.4.2 Krokový motor 17
3 GRAFOVÉ ALGORITMY **18**
3.1 BFS (BREADTH-FIRST SEARCH) 18
3.2 DFS (DEPTH-FIRST SEARCH) 19
3.3 LEFT (RIGHT) HAND ALGORITHM 20
3.4 FLOOD-FILL ALGORITHM 21
3.5 DIJKSTRA'S ALGORITHM 22
II PRAKTICKÁ ČÁST **24**
4 ARCHITEKTÚRA APLIKÁCIE **25**
4.1 HLAVNÉ OKNO 25
4.1.1 Mapa s robotom 25
4.1.2 Tlačítka 25
4.1.3 Menu 26
4.2 OKNO NAČÍTANIA MAPY 26
5 IMPLEMENTÁCIA **27**
5.1 VÝBER PLATFORMY 27
5.2 POPIS IMPLEMENTÁCIE 27
5.2.1 Balíček gui 28
5.2.2 Balíček maze 28
5.2.3 Balíček micromouse 28
5.2.4 Balíček dataStructures 28
5.2.5 Balíček files 28
5.3 UKÁŽKOVÉ IMPLEMENTÁCIE GRAFOVÝCH ALGORITMOV 28
5.3.1 DFS 28
5.3.2 BFS 31
ZÁVĚR **34**
SEZNAM POUŽITÉ LITERATURY **35**
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK **37**
SEZNAM OBRÁZKŮ **38**

SEZNAM TABULEK.....39
SEZNAM PŘÍLOH.....40

ÚVOD

V súčasnej dobe je informatika a robotika dostatočne spopularizovaná vo viacerých médiách, čo umožnilo rozvoju aj rôznych súťaži v týchto odborných disciplínach [1; 15]. Jednou kategóriou v týchto súťažiach sú tzv. robotické súťaže, kde jednotlivý účastníci sami fyzicky zostavujú a programujú robotov, ktorý sú schopný nejakého autonómneho správania. Za výsledky svojich robotov v danej disciplíne dostávajú hodnotenie od školených rozhodcov, čo im umožňuje ďalej napredovať a zdokonaľovať svoje riešenia.

Bezo sporu ďalším prínosom zostavovania robotov pre súťažiaceho je to, že si musí najprv dôsledne zvážiť aké súčiastky použije pri jeho konštrukcii vzhľadom na parametre súťaže (rušenie z okolitého prostredia, rozmiestnenie objektov v prostredí a spôsoby ich detekcie pomocou senzorov...). Pri hľadaní správnych komponentov do svojho robota, autor získava prehľad o rôznych typoch elektrotechnických zariadení a ich jednotlivých parametroch. Navrhovanie algoritmu je osobitá časť, ktorá rozvíja logické a abstraktné myslenie, pretože autor musí pri svojom riešení aj uvažovať nejakú komunikáciu medzi jednotlivými súčiastkami a ich vzájomnú spoluprácu pri vykonávaní nejakej činnosti.

Aplikácia pre vizualizáciu grafových algoritmov by mohla napomôcť najmä začínajúcim záujemcom o robotiku, priblížiť nejaké základné myšlienky a princípy systematického prehľadávania mapy, ktoré na prvý pohľad nie sú celkom badateľné, čo by mohlo znepríjemniť alebo odradiť ich záujem o túto problematiku.

Jednou z úloh tejto práce je aj vytvoriť takéto vizualizačné prostredie v nejakom programovacom jazyku.

I. TEORETICKÁ ČÁST

1 SÚŤAŽ MICROMOUSE

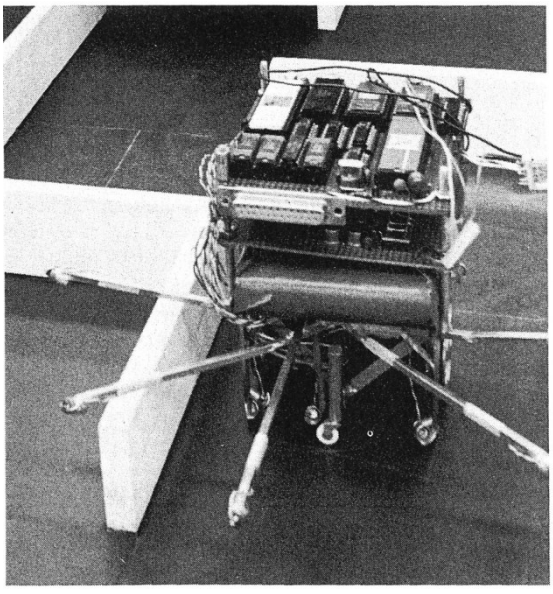
Hlavnou úlohou tejto súťaže je dostať robota štartovacieho do cieľového poľa v najkratšom čase. Pozostáva z 2 častí:

V prvej časti má robot za úlohu iba prejsť nejako bludiskom a nájsť cieľ. A ho úspešne nájde, vyštartuje z neho a snaží sa odkryť, čo najväčšiu časť mapy aby si bol istý, že v druhom kole vyberie tú ktorej priechod zaberie najmenej času (nemusí to byť tá z najmenším počtom polí v ceste k cieli). Táto časť môže byť časovo vymedzená organizátormi súťaže.

V druhej časti je robot preložený naspäť na štart a oddiaľ skúša prejsť bludisko v čo najkratšom čase. Dôležité je aby algoritmus robota dokázal vypočítať aj maximálnu rýchlosť na jednotlivých úsekoch mapy. Niektoré dokážu dosiahnuť rýchlosti aj 3,3 m/s [2].

1.1 História

IEEE Spectrum časopis ako prvý predstavil koncept Mikromouse [8]. V máji roku 1977 Spectrum ohlásila súťaž ‘Amazing Micromouse Competition’, ktorá bola plánovaná na rok 1979 v New Yorku. Bolo prítomných iba 15 súťažiacich z 6000 ohlásených (nedostavili pre organizačné a najmä technické problémy s robotami). Bludisko bolo dimenzované na 10 x 10 polí. Víťazom sa stal robot (Moonlight Flash) s algoritmom sledovania steny, ktorý sa avšak dokázal pohybovať rýchlejšie. To bolo prvým impulzom pre organizátorov prehodnotiť niektoré pravidlá (napr. umiestnenie cieľa v bludisku, rozmery bludiska, povolené rozmery robotov atď.).



Obrázok 1 Víťaz súťaže Moonlight Flash z roku 1979 [9]

V roku 1980 sa konala prvá súťaž tohoto typu v Európa a to v Londýne (organizácia Euromicro). Základná zmena prichádzala v tom, že cieľom už nebolo dostať sa z bludiska von, ale do jeho stredu, čo zamedzilo vyhrať myšiam s logikou sledovania steny a požívajúcu pravidlo zatáčania vpravo alebo vľavo. Z cca. 100 prítomných sa iba 1 myš (Nick Smith's Sterling Mouse) zvládla dostať do stredu a to s priemernou rýchlosťou 0.18 m/s.

1985 bol rok, kedy sa konala prvá celosvetová súťaž Micromouse v Japonsku (Tsukuba) pod názvom „First World Micromouse Competiton". Všetkých 6 prvých pozícií boli obsadené súťažiacimi z japonskej delegácie až na siedme miesto, ktoré získal Dave Woodfield z England Enterprise.

Prvá Micromouse súťaž v USA sa konala v roku 1987 v Atlantic City organizovaná IEE.

1.2 Pravidlá súťaže

Okrem pravidiel týkajúcich sa priamo disciplíny Micromouse existujú aj tzv. **Asimove 3 zákony** [3] týkajúce sa robotiky celkovo:

1) Robot nesmie ublížiť človeku alebo svojou nečinnosťou dopustiť, aby bolo človeku ublížené.
2) Robot sa musí podvoliť príkazom človeka, okrem prípadu, keď sú tieto rozkazy v rozpore s prvým zákonom.
3) Robot sa musí chrániť pred poškodením, okrem prípadov, kedy je táto ochrana v rozpore s prvým alebo druhým zákon.

Väčšina súťaží sa snaží zachovávať konzistentné pravidlá súťaži, ktoré sú spísané na hlavnej stránke robogames. Avšak organizátor si ich môže prispôsobiť podľa vlastných potrieb, pričom by mal účastníkov súťaže o zmenách včas informovať.

Základný 5 pravidiel [4] znie:

1) Micromyš sa musí pohybovať samovoľne a nemôže byť napájaná externe (žiadne napájacie káble) ani nemôže využívať spaľovací typ motoru.
2) Mykromyš nemôže počas priechodu bludiskom zhadzovať svoje jednotlivé časti (za účelom zníženia celkovej hmotnosti a následnom zvýšení rýchlosti).
3) Mykromyš nemôže lietať, skákať nad úroveň výšky stien bludiska a ani nijakým spôsobom poškodzovať jednotlivé časti bludiska
4) Mykromyš sa musí zmestiť do 16 x 16 cm rozmerov a ani počas priechodu bludiskom nemôže nadobudnúť nejaký rozmer väčší ako 16 cm.

5) Akékoľvek porušenie týchto pravidiel znamená okamžitú diskvalifikáciu zo súťaže a nemožnosť nárokovať si odmenu (v prípade, že dosiahol už nejaké výsledky medzičasom)

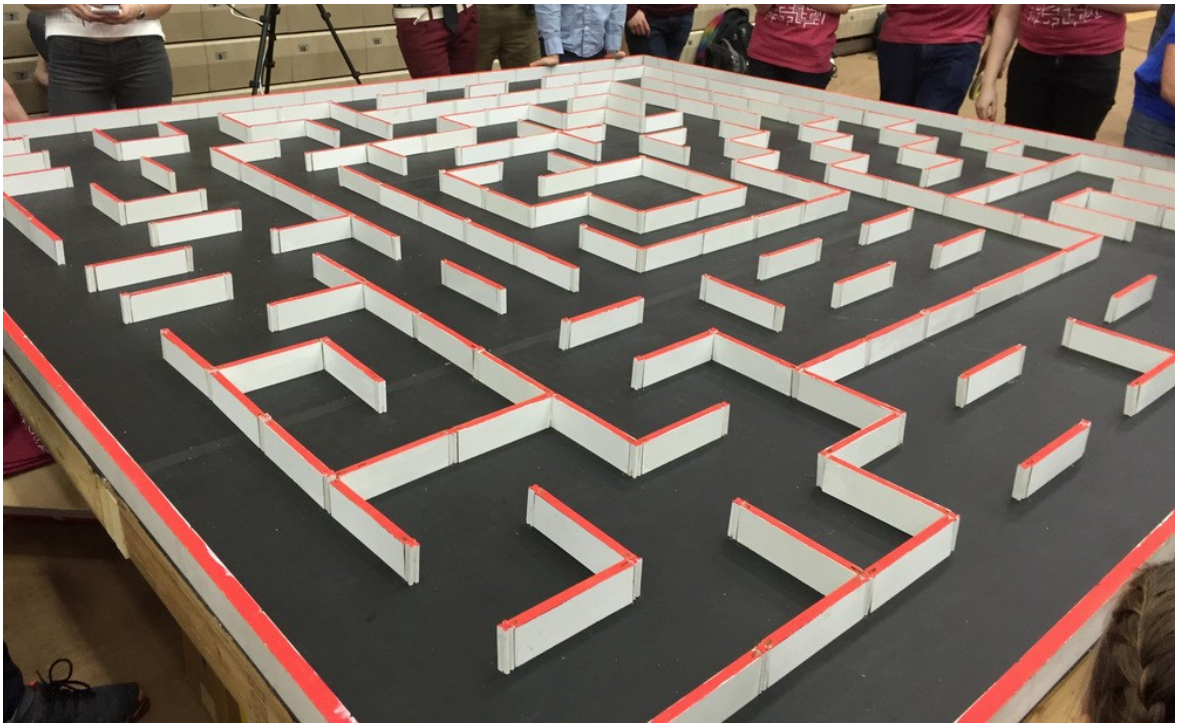
Priechod bludiskom, keď už je vo veľkej miere odhalené a umožňuje si robotom vypočítať ako sa majú správať na jednotlivých úsekoch, znižuje nároky na senzoriku a tým pádom je pravidlo číslo 2) veľmi dôležité.

1.3 Bludisko

Pravidlá popisujúce konštrukciu bludiska [4] a s tým súvislosti spojené:

1) Bludisko je zložené z plôch o rozmeroch 18 cm x 18 cm, ktoré sú 5 cm vysoké a 1.2 cm široké, čo znamená, že robot sa pohybuje po 16.8 cm širokých uličkách. Toto je popis 16 x 16 štvorcového bludiska (**Full size Micromouse**).
   Okrem toho existuje aj bludisko o rozmeroch 32 x 32 polí (**Half size Micromouse**), ktoré má ale rovnaké parametre ako Full size. Tým pádom je šírka uličiek iba 7,8 cm. Po prvý krát bola predstavená v Japonsku (Tsukuba) v roku 2009 na *30th All Japan Micromouse Competition* [5].
2) Steny bludiska sú biele (najväčšia odrazivosť pre detekciu), vrch stien červený a podlaha čierna (pohltenie rušivých svetelných lúčov pre senzoriku). Bludisko je vyrobené z dreva a podlaha je natretá nelesknúcou sa čiernou farbou. Ďalej je tu niekoľko rád ohľadom kvality materiálu (najmä čo sa týka odrazivosti a pohltivosti) a farieb v bludisku, pričom je dôraz kladený na odchýlky, ktoré môžu v parametroch konštrukcie bludiska existovať.
3) Štart je umiestnený do jedného zo 4 rohov bludiska. Štart je ohraničený z 3 strán stenami. Štartovacia čiara je umiestnená medzi štartovacím poľom a jeho jediným susediacim poľom. V momente prejdenia tejto čiary robotom za spustí časomiera. Koncové pole o rozmeroch 2 x 2 polí je umiestnené zvyčajne v strede bludiska a taktiež má iba 1 vstupné pole.
4) Na okraje každého poľa sú umiestnené stĺpiky z rozmermi 1.2 x 1.2 cm, pričom v priestore bludiska v ktorom je umožnené sa myší hýbať je vždy, pri každom stĺpiku aspoň 1 stena.
5) Predpokladá sa, že do cieľa vedie viacero ciest, pričom cieľ bude tak v bludisku umiestnený aby ho robot s jednoduchým algoritmom sledovania steny nenašiel.

Na základe všetkých týchto pravidiel by sa mohlo zdať, že sa organizátori snažili zamedziť nejakým nekalím praktikám pri súťaži, avšak pravda je taká, že žiadne početné podvody neboli v histórii súťaže zaznamenané. Jedná sa iba o snahu vyšpecifikovať pravidlá tak aby každý súťažiaci mal predstavu o tom akú úlohu má presne riešiť a nemusel zbytočne dopytovať alebo hádať nejaké detaily v súvislosti s napr. technickými parametrami bludiska.



Obrázok 2 Bludisko o rozmeroch 16 x 16 [7]

2 KONŠTRUKCIA ROBOTA

Na to aby bol robot schopný samostatného pohybu bez vonkajšieho zásahu človekom je nutné, aby dokázal vnímať svoje okolie prostredníctvom svojho senzorického aparátu a následne získané obrazové, zvukové, silové vnemy vyhodnocovať prostredníctvom nejakého naprogramovaného správania.

2.1 Mikropočítač

Je zvyčajne o integrovaný na jeden čip s RAM o veľkosti 32K bytov v kombinácii s pamäťou napr. typu EPROM alebo FLASH, typicky o veľkosti 32K bytov. CPU s taktovacou frekvenciou 32 MHZ sa stará o vykonávanie inštrukcií programu (v našom prípade správanie robota), ktorý je typicky uložený na nevolatilnej pamäti EPROM alebo FLASH. Medzivýsledky jednotlivých aritmetických operácií si ukladá do registrov, ktoré sú umiestnené priamo na procesore. RAM pamäť je určená primárne na ukladanie dát už bežiaceho programu. Jej prednosť voči nevolatilným pamätiam je jej prístupová rýchlosť k dátam, ktorá je sa ráta v desiatkach nanosekúnd, na rozdiel od FLASH v stovkách mikrosekúnd [10]. Komunikácia prebieha pomocou zbernice, ktorá sa delí na adresnú, dátovú a riadiacu. Každý komponent MPC má svoju adresu, ktorá si vymieňa dáta cez dátovú zbernicu a prostredníctvom riadiacej sa zisťuje akú operáciu má z vymieňanými dátami robiť (čítanie, zápis).

Vzhľadom k týmto parametrom je vhodné použiť algoritmus, ktorý si vytvorí všetky potrebné dátové štruktúry a vyhradí si maximum pamäte, ktorú bude využívať, hneď na začiatku. Takto môžeme jednoducho predpokladať akú veľkosť pamäte FLASH a RAM pre uchovanie programu máme zvoliť. Jeden z takýchto algoritmov je tzv. záplavový algoritmus (flood-fill), ktorý si uchováva pre každé navštívené pole veľkosť dráhy, ktorú je zo štartu potrebné k nemu prejsť. Iné grafové algoritmy vyžadujú pamäť pre dátové štruktúry typu front alebo zásobník, ktorých implementácia môže byť oveľa zložitejšia [11].

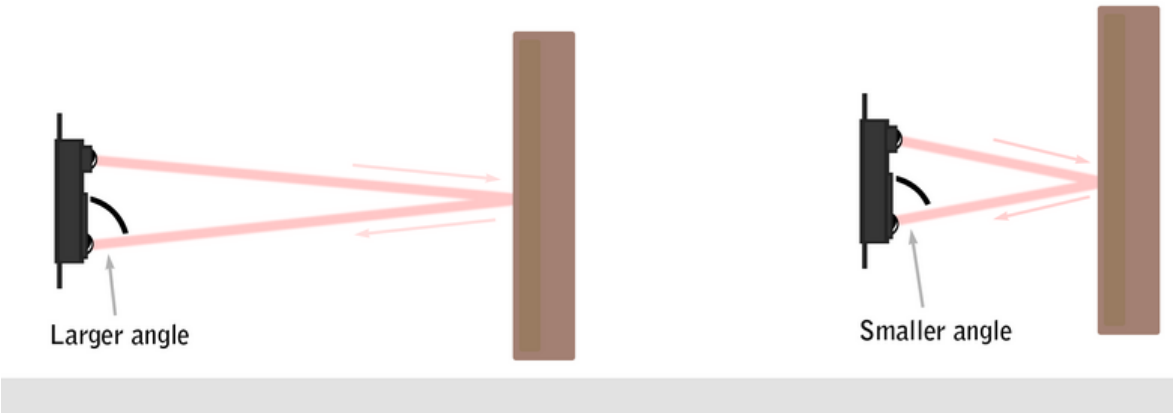
2.2 Gyroskop

Gyroskop je zariadenie na udržovanie a meranie rovnakého smeru. Robotovi toto zariadenie umožňuje meranie uhlovej rýchlosti, čo je nutné pri zatáčkach. Pokiaľ sa jedná o samostatný modul tak môžeme jeho výstupy vyvádzať pomocou pinov.

2.3 Senzory

Senzory stien slúžia nielen k tomu aby zisťovali či na danej pozícii je alebo nie je stena, ale aj k udržiavaniu konštantnej pozície v strede cesty. Najčastejšie sa používa kombinácia IR LED Diód a IR fototranzistoru. V závislosti akým spôsobom sa myš pohybuje je zvolený počet a rozmiestnenie jednotlivých senzorov.

Dôležité je si uvedomiť aký vyžarovací uhol ma daná LED dióda aby nezasahoval a nerušil iný IR fototranzistor než má. Druhým parametrom môže byť aj intenzita žiarenia danej IR diódy, ktorá môže zohrať úlohu pri rýchlosti odhalenia prekážky [12]. Kritéria pre fototranzistor sú aby boli s diódou na jednej vlnovej frekvencii a aby mal čo najväčšiu odolnosť voči rušeniu z okolitých svetelných zdrojov.



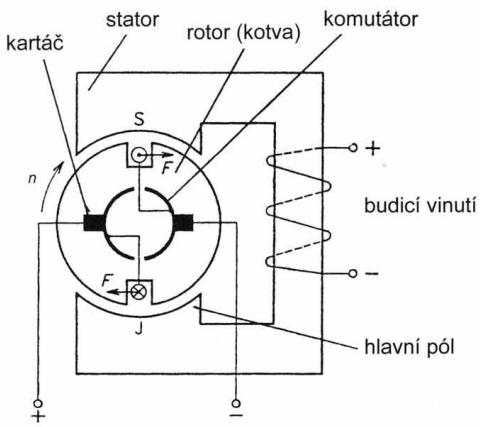
Obrázok 3 Princíp spolupráce diódy a fototranzistoru pri detekcii prekážky [13]

2.4 Motory

2.4.1 Jednosmerné motory

Princíp spočíva v tom, že cez cievku, ktorá je umiestnená na rotore (kotve) preteká elektrický prúd a tým pádom generuje magnetické pole. To pole je rovnaké ako pole statoru, pretože ho vždy komutátor v danej chvíli preklopí. Rovnaké póly sa začnú odpudzovať a na rotor začne pôsobiť magnetický moment, ktorý sa ho pokúsi otočiť, čím vznikne krútiaci moment. Otáčky js motoru rastú priamo úmerne s rastúcim napájacím napätím na cievkach. Jedna z nevýhod js motorov je, že je nutná údržba klzných kontaktov v komutátore (uhlíkové kartáče) pre opotrebovanie ku ktorému dochádza pri rotácii kotvy [14]. V spolupráci s enkodé-

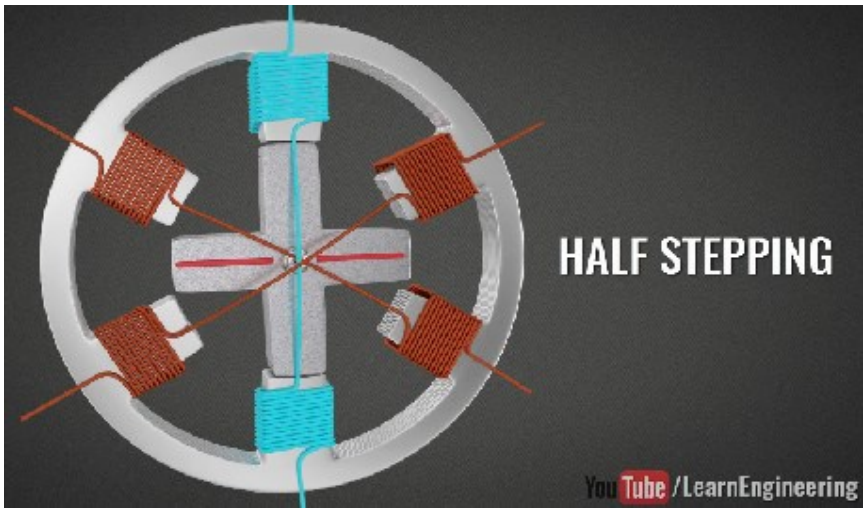
rom [17], ktorý býva zväčša umiestnený priamo na rotačných častiach alebo na hriadeli motoru, môžeme určiť ujdenú vzdialenosť. Regulácia rýchlosti prebieha cez PWM a smer otáčania kolies je riadený pomocou H-mostíku.



Obrázok 4 Schéma jednosmerného motoru [14]

2.4.2 Krokový motor

Najčastejšie vybavený permanentným magnetom v rotore, reaguje na zmenu magnetické poľa jednotlivých pólových dvojíc umiestnených na statore [6], resp. sa rotor priťahuje k opačne magneticky orientovaným dvojiciam na statore. Rýchlosť zmeny mag. poľa na týchto dvojiciach je daná frekvenciou impulzov od ovládača, ktorý budí jednotlivé vinutia na statore. Jeho výhodou je, že nemusíme použiť enkodér pre sledovanie otáčok kolies, pretože vieme ako sú rozmiestnené pólové dvojice po vnútornom obvode statora a koľko ich je. Ďalšia jeho bezo sporná výhoda spočíva v jeho údržbe, keďže iba jediná časť fyzicky spojená (a tým pádom opotrebovávaná) s rotorom sú ložiská.



Obrázok 5 Princíp krokového motoru [16]

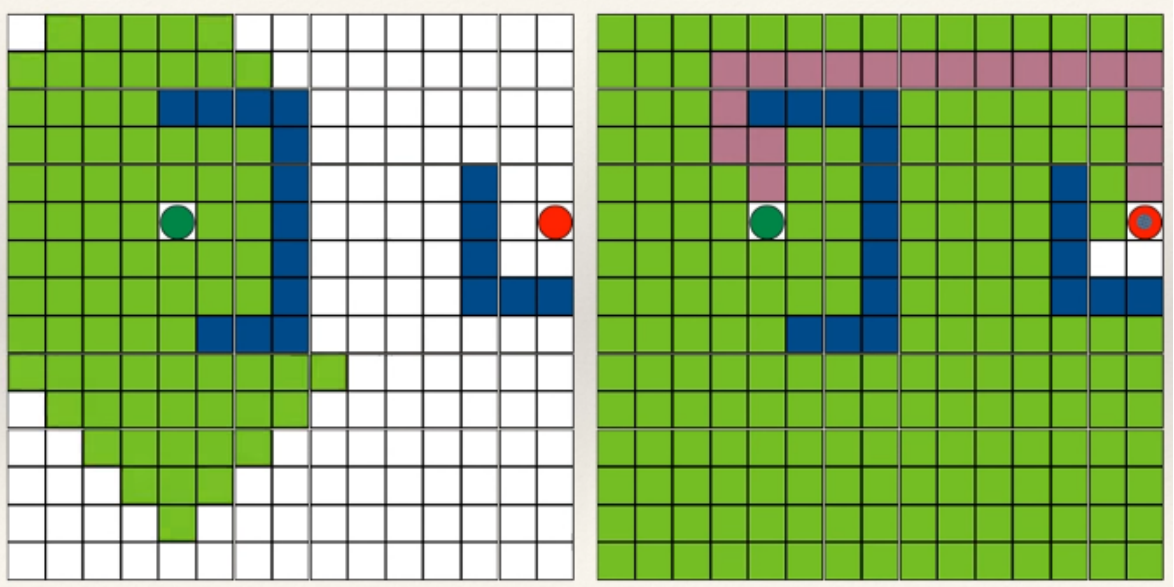
3 GRAFOVÉ ALGORITMY

Okrem obslužných metód, ktoré vyhodnocujú vstupy zo senzorického a pohybového aparátu robota, musia existovať aj algoritmy, podľa ktorých sa vie robot v danej situácii rozhodnúť, keď mu prídu nové informácie o okolí. V našom prípade musí robot najprv prehľadať bludisko a následne sa rozhodnúť, či existujú ešte aj iné cesty do cieľa, prípadne už skúsiť prejsť tú trasu, ktorá je časovo optimálna. Výhodnosť stratégií prehľadávania grafu sa v závislosti na ich implementácii, môžu zmeniť

3.1 **BFS** (Breadth-first search)

V preklade, prehľadávanie do šírky. Hlavná výhoda tohto algoritmu spočíva v tom, že vždy nájde najkratšiu cestu k cieľu, čo sa do počtu polí týka. Dokáže byť aj opakovane využitý na nájdenie časovo optimálnej trasy. Radí sa medzi záplavové algoritmy (flood-fill), pretože napodobňuje šírenie sa kvapaliny (záplavovej vlny) v nejakom uzavretom priestore.

Algoritmus využíva dátovú štruktúru front (angl. queue), ktorý na začiatku vloží štartovacie pole do fronty (inicializačný krok). Následné vyberie prvý vložený prvok z fronty a vloží do nej susedné ešte nenavštívené polia (cyklická operácia). V našom prípade (Obr. č.6) sú susedné polia bielej farby a sú umiestené nahor, nadol, vpravo, vľavo od vybraného poľa z fronty. Tento proces sa opakuje až kým nenarazíme na cieľ (červená bodka) alebo dokým nie je front prázdny (cesta k cieľu neexistuje – môže byť zatarasená alebo cieľ na mape nie je). Pre to či bolo pole už navštívené alebo nie potrebuje nejakú extra pamäť.



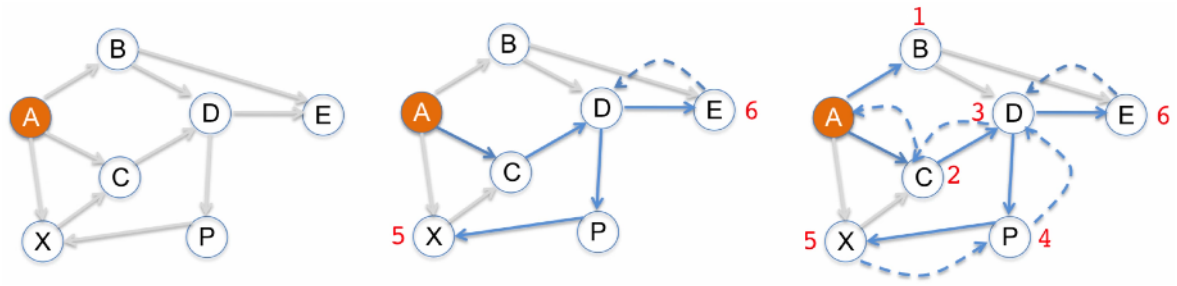
Obrázok 6 Základný princíp BFS [18]

Z obrázka č.6 vidíme, že „voda“ (zelená farba) sa nemohla dostať cez prekážku (modrá) a tak ju bočnými cestami obišla. V tejto vizualizácii bol autorom, zakázaný diagonálny pohyb, ktorý by mohol spôsobiť pri rozmernejších bludiskách hromadiacu sa chybu. V dôsledku toho, že sú stredy políčok od seba vzdialené rovnakú celočíselnú vzdialenosť (napr. 1 meter) vyplýva z Pytagorovej vety to, že by diagonálna vzdialenosť bola číslo s nekonečným desatinným rozvojom (napr. $\sqrt{2}$). Tú by sme samozrejme nemohli uložiť, do žiadnej premennej tak aby sme ne-aproximovali jej hodnotu. Preto je aj výsledná ružová najkratšia cesta vedená iba diagonálne a horizontálne.

3.2 **DFS** (Depth-First Search)

V preklade prehľadávanie do hĺbky. Algoritmus by sa dal slovne v bodoch vyložiť nasledovne:

1) Choď ďaleko (hlboko) v jednom smere, kým nenarazíš na slepú uličku alebo cieľ
2) V prípade nájdenia cieľa ukončí program
3) V prípade nájdenia slepej uličky (nemožný žiadny iný pohyb okrem smere z ktorého som prišiel), sa vracaj po ceste ktorou si sa do slepej uličky dostal a skúšaj či nie sú v ceste iné odbočky (cesty, ktorými som ešte ani raz nešiel) a vráť sa na krok 1)
4) Pokiaľ sa vrátiš až na samý začiatok (štart) a nie sú iné cesty ktorými by si sa vydal, tak cieľ na mape buď neexistuje alebo je neprístupný (zatarasený).



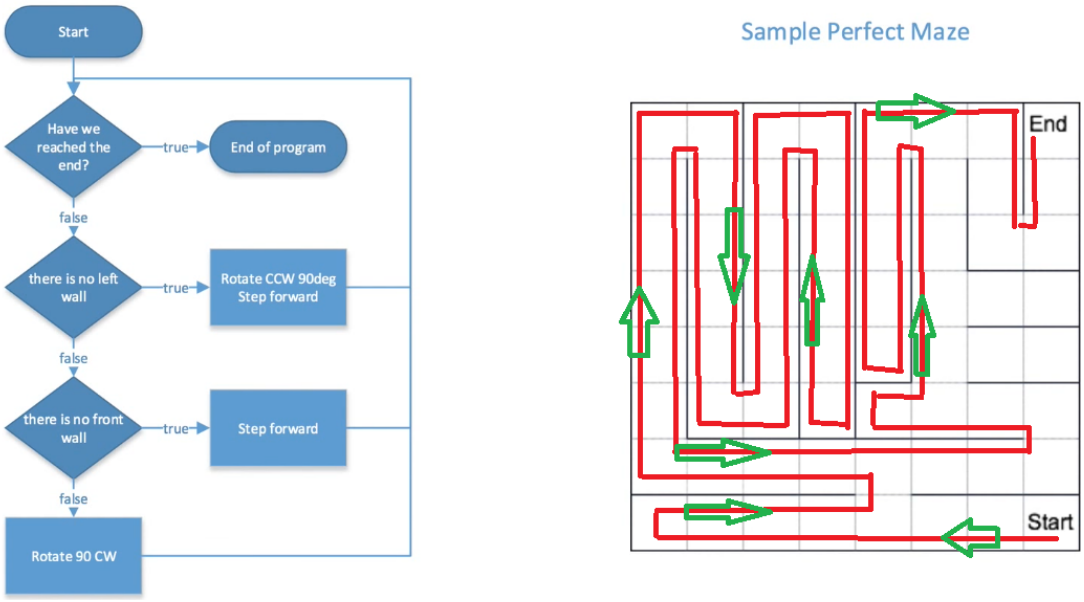
Obrázok 7 Základný princíp DFS [19]

Algoritmus využíva dátovú štruktúru zásobník (stack), do ktorej si vloží najprv štartovacie pole (inicializačný krok). Následne v cykle vyberie posledný vložený prvok zo zásobníka a skontroluje či ešte má nejakého nenavštíveného suseda a proces opakuje (cyklická operácia). Pre to či bolo pole už navštívené alebo nie potrebuje nejakú extra pamäť. Na obrázku č.7 vidíme graf s orientovanými hranami (z A sa dostanem do C, ale z C sa nedostanem do A). Zo štartovacieho poľa A, sme sa vydali po susedoch jedným smerom (A→C→D→E) a narazili sme na slepú uličku (E nemá žiadnych ďalších susedov). Vraciame sa do D, kde

vidíme, že ešte existuje jeden nenavštívený sused a to P. Z P ideme do X a dostávame sa znova do slepej uličky a vraciame sa bodmi (X→P→D→C) až naspäť do A, kde jediný nenavštívený sused je B. Slepá ulička, ideme naspäť do A a algoritmus končí pretože zásobník je prázdny (vytiahli sme posledný prvok zo zásobníku a to A). Našou úlohou v tomto prípade nebolo nájsť cieľ, ale iba prehľadať celú mapu. Výhodou tohto algoritmu napr. pre Micromouse je to, že dokážeme sa vracať naspäť po cestách v prípade narazenia na slepú uličku a nemusíme využívať nijaký iný sekundárny algoritmus hľadania cesty, ktorým by sme prešli na ďalším nenavštívený bod (na rozdiel od BFS, ktorý by musel implementovať napr. Dijkstrov algoritmus hľadania najkratšej cesty). Nevýhodou je to, že algoritmus je citlivý na poradie, akým susedov prechádzame (akým sú vkladaný do zásobníku). Algoritmus nezaručuje nájdenie najkratšej cesty, ale iba to či existuje alebo nie. Môžeme mať totižto 2 susedov, ktorých cesty vedú k cieľu (jedného z dlhšou cestou k cieľu a jedného z kratšou).

3.3 Left (Right) hand algorithm

Tento algoritmus sa radí optimistické, pretože predpokladá, že steny okolo cieľa sú spojené (neprerušené) zo stenami naľavo (prípadne napravo) od cieľa. Celý algoritmus by sa dal laicky prirovnať, k situácii kedy by človek s páskou na očiach blúdil v nejakej, pre neho neznámej, budove a vedel by sa pevne držať iba steny naľavo od neho a v prípadne steny pred ním zabočiť doprava. Pre svoju hlavnú nevýhodu sa už v Micromouse súťaži nevyužíva, pretože ciele sú kladné do stredu a žiadna zo stien naľavo (príp. napravo) nie je fyzicky spojená zo stenami okolo cieľa. Bludisko, ktoré dokáže tento algoritmus úspešne vyriešiť, je nazývaná aj ako perfektné (perfect maze).



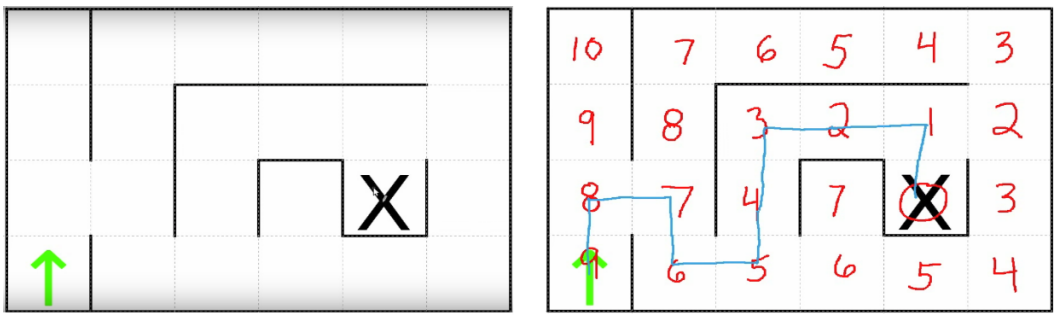
Obrázok 8 Stavový diagram naľavo a priebeh riešenia bludiska left hand algoritmom [20]

3.4 Flood-fill Algorithm

Ako už samotný názov napovedá, jedná sa, taktiež ako v prípade BFS, o záplavový algoritmus. Tento variant konkrétne rieši situáciu, keď už sú všetky vrcholy grafu (polia), známe (resp. odhalené).

Prvá fáza algoritmu spočíva v tom, že z cieľa (X) vyplňujeme susediace polia číslami, ktoré následne inkrementujeme o 1 (cyklicky opakujeme), pričom cieľu X sme priradili číslo 0. Číslo prakticky udáva vzdialenosť daného poľa od cieľa.

Druhá fáza spočíva v samotnom prejdení najkratšej cesty zo štartu (zelená šipka hore). Postupujeme (modrá čiara) vždy iba po poliach, ktoré majú priradené číslo menšie ako to na ktoré vstupujú.



Obrázok 9 Počiatočná odhalená mapa (naľavo) a riešenie (napravo) pomocou FFA [21]

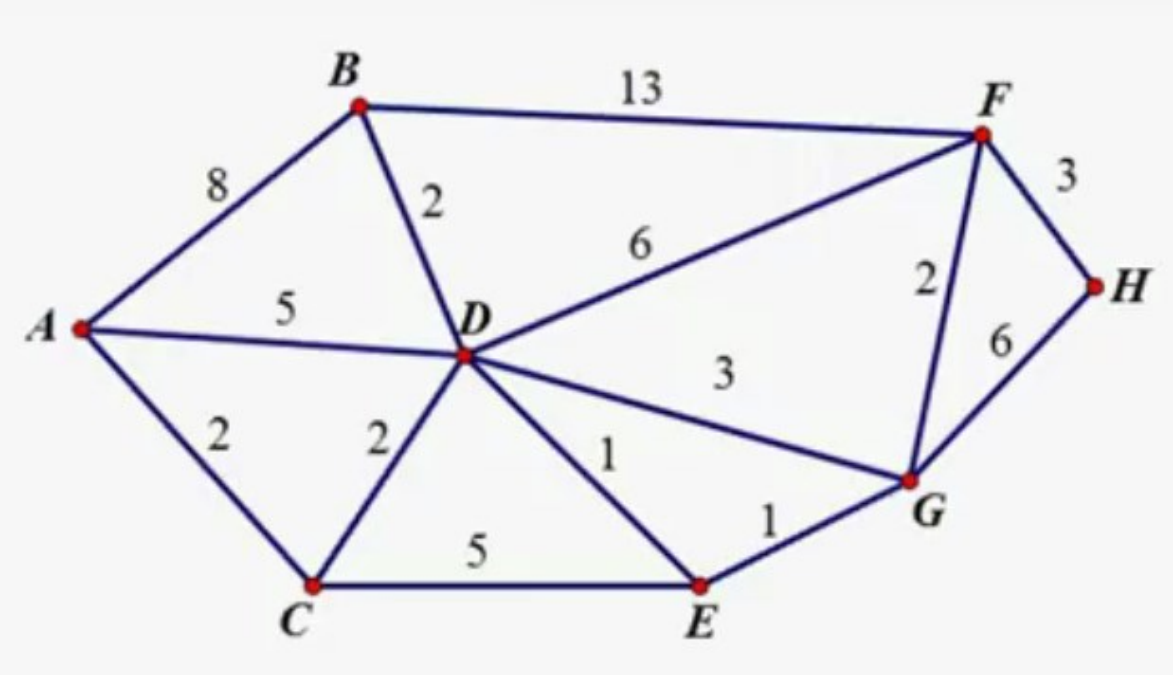
Modifikovaný variant tohto algoritmu (*Modified-Flood-fill algorithm*), už nepočíta s odkrytou (známou) mapou, ale potrebuje vedieť rozmery bludiska a presnú pozíciu štartu a cieľa.

3.5 Dijkstra's algorithm

Vyznačuje sa tým, že dokáže nájsť najkratšiu cestu a to v čase $O(|V|^2)$ s použitím minimálnej prioritnej fronty, kde V je počet vrcholov. V prípade použitia dátovej štruktúry minimálne (maximálne) haldy, sa čas vyhľadávania môže zlepšiť na $O\big((|E| + |V|) \cdot \log|V|\big)$, kde V je počet vrcholov a E počet hrán.

Pravidlá prehľadávania grafu sú nasledujúce:

0) Vložíme štartovací vrchol (A) do dátovej štruktúry, kde budú prvky zoradené podľa nejakej priority (v našom prípade vzdialenosti od štartu), pričom štart má prioritu nula (inicializačný krok)
1) Vytiahneme prvok s najnižšou prioritou von zo štruktúry (keďže hľadáme najkratšiu cestu mohla by tou štruktúrou byť napr. minimálna halda) a pozrieme sa na susedov tohto vrcholu (B, C, D) a prepíšeme im prioritu takú, akú majú ich hrany, ktorými sa do nich dostaneme + prioritu prvku z ktorého sme sa do nich dostali (taktiež si musíme pamätať meno toho vrcholu).
2) Vytiahnutý prvok si zapamätáme, že už bol v štruktúre a vložíme do nich iba vrcholy (resp. susedov), ktorý v nej ešte neboli
3) 1) a 2) krok sa cyklicky opakujú, pričom musíme navštíviť všetkých susedov a to aj v prípade, že sme našli už cieľ, ale zostávajú nejaké vrcholy nenavštívené.



Obrázok 10 Neorientovaný graf [22]

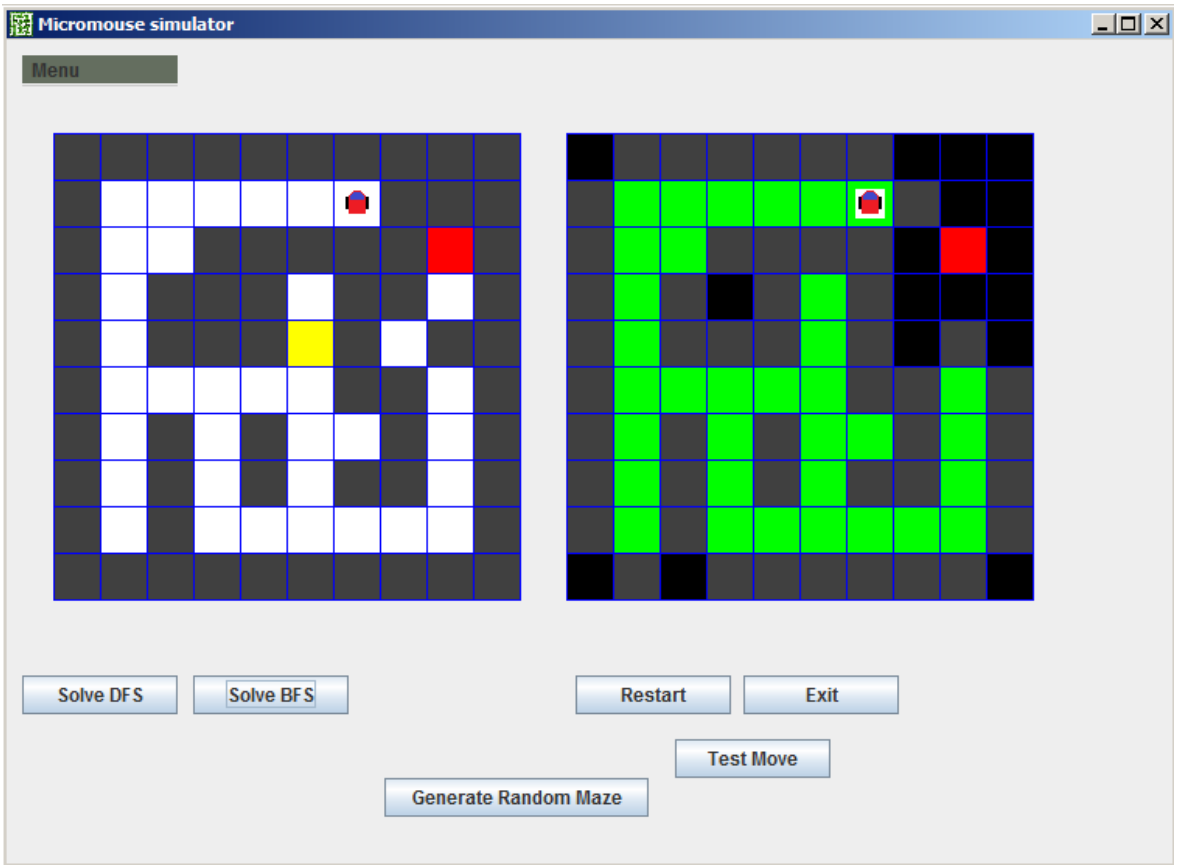
Vrcholy	A		B		C		D		E		F		G		H	
A	**0**	**A**	8	A	2	A	5	A	inf		inf		inf		inf	
C	**0**	**A**	8	A	**2**	**A**	4	C	7	C	inf		inf		inf	
D	**0**	**A**	6	D	**2**	**A**	**4**	**C**	5	D	10	D	7	D	inf	
E	**0**	**A**	6	D	**2**	**A**	**4**	**C**	**5**	**D**	10	D	6	E	inf	
B	**0**	**A**	**6**	**D**	**2**	**A**	**4**	**C**	**5**	**D**	10	D	6	E	inf	
G	**0**	**A**	**6**	**D**	**2**	**A**	**4**	**C**	**5**	**D**	8	G	**6**	**E**	12	G
F	**0**	**A**	**6**	**D**	**2**	**A**	**4**	**C**	**5**	**D**	**8**	**G**	**6**	**E**	11	F
H	**0**	**A**	**6**	**D**	**2**	**A**	**4**	**C**	**5**	**D**	**8**	**G**	**6**	**E**	**11**	**F**

Tabuľka 1 Riešenie grafu na obrázku 10

Z tabuľky č.1 vidíme, že začíname v bode A a vkladáme do neho jeho susedné vrcholy (B, C, D) . Aktuálne vybraný vrchol zo štruktúry, je v stĺpci „Vrcholy“ a hrubým písmom sú vyznačené body, ktoré sme už navštívili. V druhom kroku (výber C vrcholu) už vidíme, že sme našli kratšiu cestu do vrcholu D o dĺžke 4.

II. PRAKTICKÁ ČÁST

4 ARCHITEKTÚRA APLIKÁCIE



Obrázok 11 Hlavné okno aplikácie

4.1 Hlavné okno

Predstavuje platformu na ktorej sú umiestnené interaktívne a vizualizačné prvky. Je typu JFrame a spúšťa sa metódou run.

4.1.1 Mapa s robotom

Jedná sa o 2x mapy, pričom jedna slúži pre používateľa iba na vykreslenie mapy a druhá má za úlohu priblížiť určitý stav vedomostí myši o bludisku. Po stlačení tlačítka reset sa nastaví mapa aj myš do počiatočného stavu.

4.1.2 Tlačítka

Solve DFS – spúšťa metódu (algoritmus) prehľadávania do hĺbky, pokiaľ myš nenájde cieľ vracia sa naspäť na štart.

Solve BFS - spúšťa metódu (algoritmus) prehľadávania do šírky, pokiaľ myš nenájde cieľ zostáva na poslednom navštívenom poli.

Restart – vráti mapu aj myš do pôvodného stavu, pri načítaní mapy

Generate random maze – vygeneruje náhodné bludisko o rozmeroch 10 x 10

Exit – ukončí aplikáciu

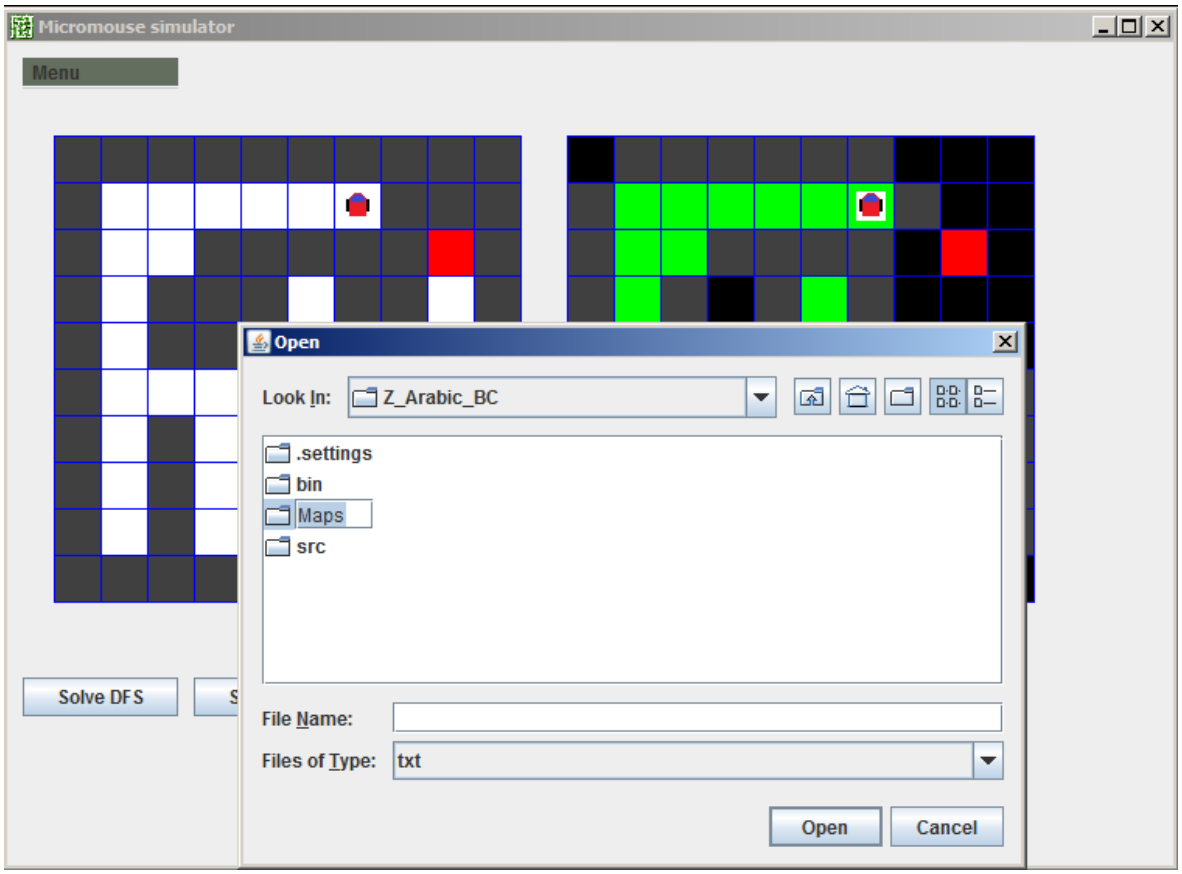
Test Move – slúži iba ako pomocné tlačítko v prípade otestovania novej metódy, ktorá sa vloží do jeho metódy actionPerformed

4.1.3 Menu

Menu obsahuje možnosť načítania vlastnej mapy z textového súboru. Nejaké ukážkové mapy sú pripravené v zložke Maps.

4.2 Okno načítania mapy

Slúži k samotnému výberu .txt súboru z ktorého sa má mapa vytvoriť.



Obrázok 12 Otvorené okno načítania mapy

5 IMPLEMENTÁCIA

Použitie tejto aplikácie vidím najmä v rámci výuky základov programovania alebo v prípade krúžkov zaoberajúcich sa robotikou. Simulácia umožňuje používateľovi vidieť priebeh algoritmu a tým pádom mu pomôcť pochopiť jeho základný princíp. Jednoduchou editáciou mapy v textovom súbore, si môže vyskúšať správanie algoritmu na svojich mapách.

V prípade, že používateľ je už programátorsky zdatnejší, môže si vyskúšať implementáciu vlastných algoritmov alebo vizualizačných metód. Preto má aj k dispozícii, v triede Algoritmy, ukážkové typy algoritmov, pričom sú organizované do blokov (Timerov) a vytvárajú tak jednoduchý stavový automat. V rámci jednotlivých automatov, si programátor môže pridať nejaké vlastné stavy správanie myši alebo vlastné vizualizačné metódy.

5.1 Výber platformy

Ako programovaciu platformu bola zvolená Java (verzia Java SE 8). Sada JDK 8 obsahuje už technológiu Java FX, ale ja som si pre tvorbu GUI zvolil pre mňa známejší Swing, poskytuje všetky potrebné komponenty GUI, ako sú napríklad tlačítka, panely, menu, pre konštrukciu solídnej oknovej aplikácie. Všetky tieto komponenty sa dajú vytvárať a editovať aj špeciálnom grafickom designeri. Najväčšou prednosťou u swingu je to, že programátor, keď chce vykonávať zmeny v GUI, tak ich musí spúšťať z takzvanej EDT (Event Dispatch Thread), čo je jediné vlákno, ktoré je schopné aktualizovať stavy jednotlivých grafický komponentov. To núti programátora, štruktúrovať kód podľa tohto obmedzenia, čo v konečnom dôsledku ho ušetrí od rôznych problémov s napr. bijúcimi sa vláknami o nejaký prostriedok alebo vláknami prepisujúcimi si nejakú jednu spoločnú premennú, čo môže vyústiť k nepredvídateľnému chovaniu GUI. Balíček `javax.swing.Timer`, obsahuje časovač, ktorý je primárne určený na vytváranie oneskorených odoziev, pomocou ktorého som vytváral animáciu rotácie. Obsahuje metódu *actionPerformed*, ktorá má prístup priamo na EDT. Časovač má aj niekoľko parametrov, ktoré si užívateľ môže nastaviť pred jeho samotnou inicializáciou napr. ako nastavenie dĺžky odozvy alebo zakázanie opakovaného spúšťania sa.

5.2 Popis implementácie

Aplikácia je členená do jednotlivých balíčkoch, tak aby ich jednotlivé triedy korešpondovali s názvom balíčku, čo sa týka do účelu činnosti. Navyše toto členenie, zabraňuje mennej kolízii jednotlivých tried, keďže majú vlastný menný priestor.

5.2.1 Balíček gui

Definuje používateľské rozhranie, jeho komponenty a obsahuje globálny Timer, ktorý slúži pre účely vykresľovania mapy na panel v nejakých periódach (animácie atď...).

5.2.2 Balíček maze

Obsahuje sadu tried, slúžiacich pre tvorbu samotného bludiska. Pričom je tu zahrnutý aj generátor náhodných máp.

5.2.3 Balíček micromouse

Sú v ňom triedy, slúžiace pre tvorbu myši a jej správania. Trieda „Algorithms“ je zároveň určená aj na implementáciu ďalších používateľských algoritmov a vizualizačných metód.

5.2.4 Balíček dataStructures

Tu sa nachádzajú pomocné dátové štruktúry, ktoré môže algoritmy využívať.

5.2.5 Balíček files

Slúži k načítaniu vlastnej mapy z textového súboru. Bludisko môže mať maximálny formát 10x10 polí. V samotnom projekte je aj zložka Maps, s nejakými už testovanými mapami.

5.3 Ukážkové implementácie grafových algoritmov

V tejto kapitole popíšem ako presne jednotlivé už vstavané algoritmy pracujú.

5.3.1 DFS

Jedná sa o algoritmus, ktorý prehľadáva mapu do hĺbky, takže musí mať dátovú štruktúru zabezpečujúcu pamätania si cesty, ktorou v prípade narazenia na slepú uličku „vycúvame“ naspäť.

```
public boolean solveByDFS(Maze maze, Micromouse mmouse, ContentPanel
BtnPnlReference)          // DFS correspond to Stack (LIFO)
    {
      destCoord = new MazeCoord(mmouse.getHor()/30, mmouse.getVer()/30);
      Stack<MazeCoord> stack = new Stack<MazeCoord>();
                                    // create stack of MazeCoord
      stack.push(destCoord);          // insert the start node

      BFS_Tree dfsTree = new BFS_Tree();
      BFS_Node rootNode = new BFS_Node(destCoord);
      dfsTree.setRoot(rootNode);
```

Táto časť kódu obsahuje počiatočnú inicializáciu zásobníku (v ktorom si pamätáme spiatočnú cestu) ako aj vloženie počiatočnej polohy myši do zásobníku.

```
        startStep(timerTaskScheduler, mmouse);
        return false;
```

Na konci metódy je zahájený prvý krok algoritmu a to spustením časovača timerTaskSchedule.

```
            timerTaskScheduler = new Timer(0, new ActionListener()
            {
                  public void actionPerformed(ActionEvent evt)
                  {
                        System.out.println("timerTaskScheduler.exec");

                        if (mmouse.getStatus() == 0)
                        {
                          if (!stack.empty())
                          {
                               destCoord = stack.pop();
                               RotationPlanning(mmouse);
                              nextStep(timerTaskScheduler, timerMouseRotate, mmouse);
                          }
                          else
                          {
                               System.out.println("Worker stopped - no way");
                               stopStep(timerTaskScheduler);
                               return;
                          }
                        }
                        else
                               System.out.println("Stack waiting for mouse");
                  }
            });
```

Tu už začína prvý blok stavového automatu, kde sa kontroluje či zásobník už nie je prázdny (cesta neexistuje). Ak existuje, tak vyjmi posledný vložený prvok zo zásobníku, naplánuj rotáciu v smere, kam pôjdeš a prejdi na ďalší krok.

```
timerMouseRotate = new Timer(50, new ActionListener()
{
  public void actionPerformed(ActionEvent evt)
  {
       System.out.println("Rotating " + mmouse.getRotation() + " == " + beta);

       if (mmouse.getStatus() == 1)
       {
             if (BtnPnlReference.getTick() == false)
             {
                   if (mmouse.getRotation() == beta)
                   {
                       System.out.println("Rotating END");
                       nextStep(timerMouseRotate, timerMouseMoving, mmouse);
                   }
```

```
                    else
                    {
                        mmouse.setRotation(mmouse.getRotation() + stepRotation);
                        if (mmouse.getRotation() >= 360)
                            mmouse.setRotation(mmouse.getRotation() - 360);
                        else if (mmouse.getRotation() < 0)
                            mmouse.setRotation(360 + mmouse.getRotation());
                    }
                    BtnPnlReference.repaint();
                }
            }
        }
});
```

Skontrolujeme:

- či stav algoritmu myši je 1 (sme v rotačnom režime).
- máme povolenie vykresľovať od globálneho časovača

Pokiaľ je sú hore uvedené splnené, začneme s rotáciou okolo vlastnej osi.

```
timerMouseMoving = new Timer(100, new ActionListener()
{
       public void actionPerformed(ActionEvent evt)
       {
              if (mmouse.getStatus() == 2)
              {
                    if (BtnPnlReference.getTick() == false)
                    {
                          mmouse.moveTo(mmouse, destCoord, BtnPnlReference, maze);
                          nextStep(timerMouseMoving, timerMousePlaning, mmouse);
                    }
              }
        }
});
```

Skontrolujeme:

- či stav algoritmu myši je 2 (sme v translačnom režime).
- máme povolenie vykresľovať od globálneho časovača

Pohneme sa vpred na danú destináciu (destCoord).

```
timerMousePlaning = new Timer(0, new ActionListener()
{
       public void actionPerformed(ActionEvent evt)
       {
              if (mmouse.getStatus() == 3)
              {
                   if (mmouse.getMazeVisited().isFinal(destCoord))
                   {
                         System.out.println("Final Timers stopped\n");
                         stopStep(timerMousePlaning);
                         return;
                    }

                    if      (addToTreeStructure_DFS(mmouse, dfsTree, stack, new MazeCo-
ord( destCoord.getHor()  , destCoord.getVer()-1 ), BtnPnlReference, maze, 1));      //
go up    from the current node
                          else if (addToTreeStructure_DFS(mmouse, dfsTree, stack, new
MazeCoord( destCoord.getHor()+1, destCoord.getVer()   ), BtnPnlReference, maze, 2));
       // go right from the current node
```

```
                        else if (addToTreeStructure_DFS(mmouse, dfsTree, stack, new
MazeCoord( destCoord.getHor()-1, destCoord.getVer()   ), BtnPnlReference, maze, 3));
      // go left  from the current node
                        else if (addToTreeStructure_DFS(mmouse, dfsTree, stack, new
MazeCoord( destCoord.getHor()  , destCoord.getVer()+1 ), BtnPnlReference, maze, 4));
      // go down  from the current node

                        nextStep(timerMousePlaning, timerTaskScheduler, mmouse);
                    }
                    }
             });
```

Skontrolujeme:

- či stav algoritmu myši je 3 (sme vo vkladacom režime).
- sme v cieli (ak áno, program končí)

Pokiaľ nie sme v cieli a stav algoritmu je 3, tak:

if - choď hore (vlož horný prvok, od aktuálnej pozície myši, do zásobníku ak je ešte nenavštívený)

else if – choď vpravo

else if – choď vľavo

else if – choď dole

Vrátime sa na stav 0 (nextStep).

5.3.2 BFS

Jedná sa o algoritmus, ktorý prehľadáva mapu do šírky, takže musí mať dátovú štruktúru zabezpečujúcu si pamätanie všetkých susedných polí, od aktuálneho vybraného zo štruktúry.

V tejto implementácii nás zaujíma v akom poradí sú jednotlivé polia na mape navštevované.

```
public boolean solveBFS(Maze maze, Micromouse mmouse, ContentPanel BtnPnlReference)
//BFS correspond to Queue (FIFO)
    {
      destCoord = new MazeCoord(mmouse.getHor()/30, mmouse.getVer()/30);

      LinkedList<MazeCoord> list = new LinkedList<MazeCoord>();
      list.add(destCoord);

      BFS_Tree bfsTree = new BFS_Tree();
      BFS_Node rootNode = new BFS_Node(destCoord);
      bfsTree.setRoot(rootNode);
```

Táto časť kódu obsahuje počiatočnú inicializáciu listu (všetkých susedné polia od aktuálneho poľa) ako aj vloženie počiatočnej polohy myši do listu.

```
startStep(timerTaskScheduler, mmouse);
return false;
```

Na konci metódy je zahájený prvý krok algoritmu a to spustením časovača timerTaskSchedule.

```
timerTaskScheduler = new Timer(0, new ActionListener()
{
      public void actionPerformed(ActionEvent evt)
      {
            if (mmouse.getStatus() == 0)
            {
                  if (!list.isEmpty())
                  {
                      destCoord = list.removeFirst();
                      //RotationPlanning(mmouse);
                      nextStep(timerTaskScheduler, timerMouseRotate, mmouse);
                  }
                  else
                  {
                      System.out.println("Worker stopped - no way\n");
                      stopStep(timerTaskScheduler);
                      return;
                  }
             }
             else
                  System.out.println("Stack waiting for mouse\n");
      }
});
```

Tu už začína prvý blok stavového automatu, kde sa kontroluje či list už nie je prázdny (cesta neexistuje). Ak existuje, tak vyjmi posledný vložený prvok z listu. Volanie rotácie „`RotationPlanning(mmouse);`“ je zakomentované pretože, na demonštráciu priebehu algoritmu nie je nutná.

```
timerMouseRotate = new Timer(50, new ActionListener()
{
    public void actionPerformed(ActionEvent evt)
    {
       if (mmouse.getStatus() == 1)
       {
              if (BtnPnlReference.getTick() == false)
              {
                    nextStep(timerMouseRotate, timerMouseMoving, mmouse);
              }}}});
```

Rotáciu myši nebudeme simulovať a prejdeme k ďalšiemu kroku.

```
timerMouseMoving = new Timer(100, new ActionListener()
{
      public void actionPerformed(ActionEvent evt)
      {
            if (mmouse.getStatus() == 2)
            {
               if (BtnPnlReference.getTick() == false)
               {
                 mmouse.moveTo(mmouse, destCoord, BtnPnlReference, maze);
                 nextStep(timerMouseMoving, timerMousePlaning, mmouse);
               }}}});
```

Skontrolujeme:

- či stav algoritmu myši je 2 (sme v translačnom režime).
- máme povolenie vykresľovať od globálneho časovača

Pohneme sa na danú destináciu (destCoord).

```
timerMousePlaning = new Timer(0, new ActionListener()
{
      public void actionPerformed(ActionEvent evt)
      {
            if (mmouse.getStatus() == 3)
            {
                  if (BtnPnlReference.getTick() == false)
                  {
                      if (mmouse.getMazeVisited().isFinal(destCoord))
                      {
                           System.out.println("Final Timers stopped\n");
printPath(bfsTree.getTreeNode(new BFS_Node(destCoord)), BtnPnlReference,
maze.getGridsize(), maze.getHorDrawPadding(), maze.getVerDrawPadding());
                           stopStep(timerMousePlaning);
                           return;
                      }

                            addToTreeStructure_BFS(mmouse, bfsTree, list,
new MazeCoord( destCoord.getHor()  , destCoord.getVer()-1 ), 1);       // go
up    from the current node
                           addToTreeStructure_BFS(mmouse, bfsTree, list,
new MazeCoord( destCoord.getHor()+1, destCoord.getVer()   ), 2);       // go
right from the current node
                           addToTreeStructure_BFS(mmouse, bfsTree, list,
new MazeCoord( destCoord.getHor()-1, destCoord.getVer()   ), 3);       // go
left  from the current node
                           addToTreeStructure_BFS(mmouse, bfsTree, list,
new MazeCoord( destCoord.getHor()  , destCoord.getVer()+1 ), 4);       // go
down  from the current node

                           nextStep(timerMousePlaning, timerTaskSchedu-
ler, mmouse);
                  }
            }
      }
});
```

Skontrolujeme:

- či stav algoritmu myši je 3 (sme vo vkladacom režime).
- sme v cieli (ak áno, `printPath k cieľu` a program končí)

Pokiaľ nie sme v cieli a stav algoritmu je 3, tak:
if - choď hore (vlož horný prvok, od aktuálnej pozície myši, do listu ak je ešte nenavštívený)
if – choď vpravo
if – choď vľavo
if – choď dole

Vrátime sa na stav 0 (nextStep).

ZÁVĚR

Hlavným zámerom tejto prace bolo vytvoriť okennú aplikáciu, ktorá by simulovala jednotlivé aspekty chovania chobota v súvislosti so súťažou Micromouse. Prvoradé bolo si zistiť ako súťaž vlastne prebieha. Nejaké pravidlá boli síce sformulované na stránke robogames, ale v žiadnom prípade sa nejednalo o celosvetovo záväzné, pretože súťaže typu Micromouse, nezastrešuje iba jedna organizácia. Štúdium konštrukcie robotov mi pomohlo lepšie si zorganizovať logiku svojej aplikácie, pretože som osvojil vedomosti o funkcii jednotlivých komponentov a až následne tvoril jednotlivé triedy a ich metódy.

V samotnom programe som kládol dôraz na to akým spôsobom bude užívateľovi umožnené pracovať rozumne s jeho jednotlivými funkciami, prípadne ich spôsobom rozšírenia. To vyústilo v to, že algoritmy vyhľadávania, sú písane do navzájom zreťazených blokov, ktoré pripomínajú jednoduchý stavový automat. Ako blok stavového automatu som použil časovač pre jeho jednoduchú implementáciu a pre jednoduchý spôsob jeho používania. Sledovanie samotného fungovania algoritmu prehľadávania mapy je prehľadné, pretože užívateľ presne vidí, čo sa na mape zmenilo a aký bol je počiatočný stav.

V neposlednej rade, je v triede „Algorithms“ načrtnutý základný princíp tvorenia algoritmov, ktorý by mal pomôcť používateľovi pochopiť proces ich štrukturovania a tvorby. Tento princíp je načrtnutý vo forme implementácii vybraných grafových algoritmov.

SEZNAM POUŽITÉ LITERATURY

[1] DREAMS, Robots. *31st All Japan Micromouse Robot Event: Ng Beng Kiat https://www.youtube.com/watch?v=76blllun09Q* [online]. In: . 26.11.2010 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=76blllun09Q
[2] ACKERMAN, Evan. *Meet the New World's Fastest Micromouse Robot* [online]. 21.nov.2011 [cit. 2018-05-29]. Dostupné z: https://spectrum.ieee.org/automaton/robotics/diy/meet-the-new-worlds-fastest-micromouse
[3] ROBERT ANDERSON, Mark. *After 75 years, Isaac Asimov's Three Laws of Robotics need updating* [online]. 17.mar.2017 [cit. 2018-05-29]. Dostupné z: http://theconversation.com/after-75-years-isaac-asimovs-three-laws-of-robotics-need-updating-74501
[4] , Micromouse. *Maze Solving / Micromouse Rules* [online]. [cit. 2018-05-29]. Dostupné z: http://robogames.net/rules/maze.php
[5] HARRISON, Peter. *Japan 2009 Micromouse Rules* [online]. 16.jul.2009 [cit. 2018-05-29]. Dostupné z: http://www.micromouseonline.com/2009/07/16/japan-2009-micromouse-rules/
[6] HARRISON, Peter. *Stepper Motors* [online]. [cit. 2018-05-31]. Dostupné z: http://www.micromouseonline.com/micromouse-book/motors/stepper-motors/
[7] DATSIKAS, Chris. *Micromouse* [online]. In: . 2015 [cit. 2018-05-30]. Dostupné z: http://www.chrisdatsikas.com/micromouse/
[8] *Micromouse History* [online]. [cit. 2018-05-30]. Dostupné z: http://www.micromouseonline.com/micromouse-book/history/
[9] *1977-79 – "Moonlight Special" Battelle Inst. (American)* [online]. 8.máj.2010 [cit. 2018-05-30]. Dostupné z: http://cyberneticzoo.com/mazesolvers/1977-79-moonlight-special-battelle-inst-american/
[10] ROUSE, Margaret. *Fast Guide to RAM* [online]. 2011 [cit. 2018-05-30]. Dostupné z: https://whatis.techtarget.com/reference/Fast-Guide-to-RAM
[11] HARRISON, Peter. *Memory* [online]. [cit. 2018-05-30]. Dostupné z: http://www.micromouseonline.com/micromouse-book/command-and-control/memory/
[12] SINGH, Harjit. *Micromouse Sensor Design* [online]. 3.nov.2015 [cit. 2018-05-30]. Dostupné z: http://www.micromouseonline.com/2015/11/03/zeetah-sensor-design/
[13] *What is different about the Sharp IR Sensor?* [online]. [cit. 2018-05-30]. Dostupné z: http://education.rec.ri.cmu.edu/content/electronics/boe/ir_sensor/4.html
[14] ROUBÍČEK, Ota. *Princip stejnosměrných motorů* [online]. 13.8.2008 [cit. 2018-05-30]. Dostupné z: https://elektrika.cz/data/clanky/princip-stejnosmernych-motoru
[15] YE, Green. *Micromouse USA* [online]. [cit. 2018-05-31]. Dostupné z: http://micromouseusa.com/
[16] , Learn Engineering. *How does a Stepper Motor work ?* [online]. In: . 19.10.2016 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=eyqwLiowZiU
[17] ROYER, Dan. *Micromouse encoders to measure distance* [online]. 19.máj.2016 [cit. 2018-05-31]. Dostupné z: https://www.marginallyclever.com/2016/05/measuring-micromouse-distance-travelled-encoders/
[18] MARKOVITCH, Shaul. *Breadth first search* [online]. 9.11.2014 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=n1Cm1XiGd48
[19] VENUGOPAL, Sesh. *Graph Topological Sort Using Depth-First Search* [online]. 4.8.2016 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=n_yl2a6n7nM
[20] , professorrobertsolis. *CS225 Algorithms - Solving a maze using the left hand algorithm* [online]. 10.9.2015 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=yz0EOsUkwD4

[21] BACKUS, Michael. *Dynamic Programming / Flood Fill Algorithm* [online]. 18.5.2015 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=Zwh-QNlsurI
[22] *Dijkstra's Algorithm: Another example* [online]. 14.11.2014 [cit. 2018-05-31]. Dostupné z: https://www.youtube.com/watch?v=5GT5hYzjNoo

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

LED	Luminiscenčná dióda
IR	Infrared - infračervený
CPU	Central processing unit - Procesorová jednotka
RAM	Random Access Memory - Pamäť s priamym prístupom
ROM	Read-Only Memory - Permanentná pamäť určená len pre čítanie
EPROM	Erasable Programmable Read-Only Memory -Pamäť typu ROM
MPC	Microprocessor - Mikropočítač
GUI	Graphical user interface – Grafické používateľské rozhranie

SEZNAM OBRÁZKŮ

Obrázok 1 Víťaz súťaže Moonlight Flash z roku 1979 [9] 11
Obrázok 2 Bludisko o rozmeroch 16 x 16 [7] 14
Obrázok 3 Princíp spolupráce diódy a fototranzistoru pri detekcii prekážky [13].....16
Obrázok 4 Schéma jednosmerného motoru [14] 17
Obrázok 5 Princíp krokového motoru [16].....17
Obrázok 6 Základný princíp BFS [18] 18
Obrázok 7 Základný princíp DFS [19] 19
Obrázok 8 Stavový diagram naľavo a priebeh riešenia bludiska left hand algoritmom [20] 21
Obrázok 9 Počiatočná odhalená mapa (naľavo) a riešenie (napravo) pomocou FFA [21] 21
Obrázok 10 Neorientovaný graf [22].....23
Obrázok 11 Hlavné okno aplikácie.....25
Obrázok 12 Otvorené okno načítania mapy 26

SEZNAM TABULEK

Tabuľka 1 Riešenie grafu na obrázku 10 23

SEZNAM PŘÍLOH

P I 1x CD

PŘÍLOHA P I: NÁZEV PŘÍLOHY

Obsah priloženého CD:

- Bakalárska práce v elektronickej podobe

- Zdrojový kód programu

- Spustiteľná aplikácia