VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DEMONSTRAČNÍ PROGRAM PRO PŘEDMĚT IZU

DEMONSTRATIONAL PROGRAM FOR IZU COURSE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MIROSLAVA MÍŠOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **FRANTIŠEK V. ZBOŘIL, doc. Ing., CSc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Studentka: **Míšová Miroslava**
Program: Informační technologie
Název: **Demonstrační program pro předmět IZU**
Demonstrational Program for IZU Course
Kategorie: Umělá inteligence
Zadání:

1. Prostudujte zdrojové programy apletů zveřejněných na privátních stránkách předmětu IZU.
2. Navrhněte program pro demonstraci principů metod prezentovaných v předmětu IZU, který bude pokrývat alespoň rozsah současných apletů.
3. Při návrhu předpokládejte možnost pozdějšího doplňování/nahrazování novými metodami.
4. Navržený program implementujte s cílem názorné demonstrace (grafický výstup, snadné nastavování parametrů, krokování).
5. Ověřte funkčnost programu.

Literatura:

- Zdrojové programy k jednotlivým apletům (dodá vedoucí práce).
- Studijní opora k předmětu IZU

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František V., doc. Ing., CSc.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato bakalářská práce se věnuje vypracování studijních aplikací pro předmět Základy umělé inteligence. Tyto aplikace mají za vzor starší applety, které využívají nástroje, pro které skončila podpora. Pro jednotlivé aplikace byl vytvořen objektově orientovaný návrh a následně byly implementovány. Při vytváření byl dbán důraz na jednoduchost uživatelského rozhraní a na možné další rozšiřování aplikací.

Abstract

This bachelor's thesis deals with development of new study aplications for course Fundamentals of Artificial Intelligence. These aplications are based on the older version of JavaApplet, which use features, that are no longer supported. Each applicatoin was made acording to an object-oriented paradigm and than implemented. Special care was taken in order for the UI to be intuitive and easy to use and also for the aplication to be able to be further developed.

Klíčová slova

Základy umělé inteligence, IZU, stojové učení, demonstrační programy, C++, Qt, K-means, lineární klasifikátor, MiniMax, AlphaBeta, N-queens, nelineární klasifikátor, perceptron, pocket, posilované učení, prohledávání stavového prostoru

Keywords

Fundamentals of Artificial Intelligence, IZU, machine learning, demonstration programs, C++, Qt, K-means, linear classifier, MiniMax, AlphaBeta, N-queens, nonlinear classifier, perceptron, pocket, reinforcement learning, states space search

Citace

MÍŠOVÁ, Miroslava. *Demonstrační program pro předmět IZU.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce František V. Zbořil, doc. Ing., CSc.

Demonstrační program pro předmět IZU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Františka V. Zbořila, doc. Ing., CSc. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

. . . .
Miroslava Míšová
30. července 2019

Poděkování

Touto cestou bych chtěla poděkovat svému vedoucímu bakalářské práce, panu Františku V. Zbořilovi, doc. Ing., CSc., za cenné rady, trpělivost a pomoc při vzniku této práce.

Dále bych chtěla poděkovat rodině, přátelům a svému příteli za morální podporu, díky které jsem tuto práci ve zdraví dokončila.

Obsah

1 Úvod **3**

2 Teorie **4**
2.1 Prohledávání stavového prostoru 4
2.1.1 Neinformované metody 5
2.1.2 Informované metody 8
2.1.3 Metody lokálního prohledávání 10
2.2 Metody řešení úloh s omezujícími podmínkami (CSP) 11
2.2.1 Forward Checking 11
2.2.2 Min Conflict 12
2.2.3 Hopfield Neural Network 12
2.3 Metody hraní her 13
2.3.1 MiniMax 13
2.3.2 AlphaBeta 14
2.4 Rozpoznávání 15
2.4.1 Lineární klasifikátor 15
2.4.2 Nelineární klasifikátor 15
2.4.3 Perceptron 16
2.4.4 Pocket 17
2.5 Strojové učení 17
2.5.1 K-means clustering 18
2.5.2 TD learning 18
2.5.3 Q learning 19

3 Analýza a návrh řešení **20**
3.1 Prostudování zadání a původní aplikace 20
3.2 Obecná aplikace 21
3.3 Jednotlivé aplikace 22
3.3.1 Prohledávání stavového prostoru 22
3.3.2 MiniMax a AlphaBeta 23
3.3.3 N-queens 24
3.3.4 Lineární klasifikátor, Perceptron a Pocket 24
3.3.5 Nelineární klasifikátor 25
3.3.6 K-means 26
3.3.7 Posilované učení 27

4 Implementace a vyhodnocení **29**
4.1 Použité technologie a cílová platforma 29

4.2 Implementace aplikací 29
4.2.1 Aplikace state_space_search 29
4.2.2 Aplikace minimax_alphabeta 30
4.2.3 Aplikace n_queens 31
4.2.4 Aplikace lin_klas a nelin_klas 31
4.2.5 Aplikace perceptron 31
4.2.6 Aplikace k_means 32
4.2.7 Aplikace reinforcement_learning 32

5 Porovnání s původní verzí **33**

6 Shrnutí **40**

Literatura **41**

Kapitola 1

Úvod

Jeden z povinných předmětů vyučovaných ve 2. ročníku bakalářského studijního programu Informační technologie se jmenuje Základy umělé inteligence. Studenti se v rámci něj seznamují například s oblastí metod řešení úloh prohledávání stavového prostoru, hraní her, základními principy strojového učení a zpracovávání přirozeného jazyka.

> "Co slyším, to zapomenu. Co vidím, si pamatuji. Co si vyzkouším, tomu rozumím."
> Konfucius

K pochopení látky probírané na přednáškách slouží především studijní opora, jednotlivé prezentace a applety. Tyto applety byly vytvořeny za účelem vizualizace probíraných metod a algoritmů. Bohužel využívají aplikaci Java applet, jejíž podpora webovými prohlížeči byla z bezpečnostních důvodů ukončena. Je tedy možné je spustit pouze pomocí starších verzí webových prohlížečů a pluginů. Nastavování a spouštění může být pro studenty náročné, což může vést k tomu, že si nemohou vyzkoušet probíranou látku před hodnoceným počítačovým cvičením a k nedostatečnému pochopení metod.

Tato bakalářská práce se zabývá nahrazením původních appletů aplikacemi vytvořenými současnými technologiemi, které by bylo možné v budoucnu rozšířit o nové metody a algoritmy. Při jejich vytváření je využito třídního návrhu a vybraných návrhových vzorů. Teoretická část technické zprávy se zabývá vysvětlením jednotlivých metod a potřebného základu, za kterou se nachází návrh obecné aplikace a jednotlivých dílčích aplikací, kapitola implementace a vyhodnocení. V závěru práce je věnován prostor pro porovnání s původní verzí appletů, experimentování a konečnému shrnutí výsledků.

Kapitola 2

Teorie

Pro pochopení činnosti aplikace, která je jedním z výstupů této práce, je třeba nejprve popsat základní algoritmy, které se běžně používají nejen v oboru umělé inteligence. Tyto algoritmy jsou rozděleny do několika skupin – Prohledávání stavového prostoru, Metody řešení úloh s omezujícími podmínkami, Metody hraní her, Strojové učení a Rozpoznávání. Dále lze jmenovat Metody řešení úloh založené na rozkladu úloh na podproblémy, například rozklad na AND a OR problémy, a algoritmy založené na genetických algoritmech, které již nejsou základem této práce. Algoritmy a informační popisy vycházejí z textů studijní opory předmětu Základy umělé inteligence [12], přednášek a knihy Umělá inteligence [1].

2.1 Prohledávání stavového prostoru

Stavový prostor je definován dvojicí (S, O), kde symbol S (States) označuje množinu všech možných stavů úlohy $S = \{s_1, s_2, s_3, ...\}$ a symbol O (Operators) množinu všech operátorů $O = \{o_1, ..., o_j\}$, kterými lze stavy úlohy měnit. Úloha je definovaná dvojicí (s_0, G), kde symbol $s_0 \in S$ značí počáteční stav a symbol $G \subset S$ množinu cílových stavů této úlohy (Goals). Řešením úlohy je posloupnost operátorů $s_1 = o_1(s_0), s_2 = o_2(s_1), ..., s_n = o_n(s_n - 1), s_n \in G$. [12] [7]

Stavový prostor si lze představit jako orientovaný graf, který je složený z uzlů a hran. Uzly budou odpovídat stavům z množiny všech možných stavů úlohy S a hrany operátorům z množiny všech operátorů O. Řešením úlohy je pak cesta z počátečního uzlu po cílový. To, zda je cesta nejkratší možná, tj. minimální, či se dá získat za nejkratší možný čas, je ovlivněno volbou metody.

Pokud má být strategie pro prohledávání úspěšnou, musí splňovat následující základní vlastnosti:

1. musí vést k prohledávání, tj. způsobit pohyb a zabraňovat cyklům
2. musí být systematická

Při prohledávání malého stavového prostoru nám systematické prohledávání obvykle nevadí. Při složitějších úlohách s rozsáhlým prohledávacím prostorem už může být systematické prohledávání značně neefektivní. Dochází zde ke zpracování značně velkého stavového prostoru, který ovšem nevede k cíli. Pokud bychom věděli dodatečné informace o stavech, kterými procházíme (například směr nebo vzdálenost k cílovému stavu), můžeme využít algoritmů, které tyto informace aplikují při takzvaných heuristikách pro ohodnocování stavů. Těmto

algoritmům potom říkáme informované. Čím lepší heuristiky tyto algoritmy využívají, tím menší část stavového prostoru musí projít a tím efektivnější jsou při hledání cesty k cíly. [1]

2.1.1 Neinformované metody

Neinformované metody, někdy zvané Slepé metody, nevyužívají žádné informace, které by jim mohly usnadnit řešení problémů. Nemohou například hodnotit, jak daleko se cílový uzel nachází ani jakým směrem. Níže jsou uvedeny verze metod, které byly použity v implementaci. [12] [8]

Breadth First Search

BFS (Slepé prohledávání do šířky) je založena na expandování uzlu s minimální hloubkou zanoření. Využívá při tom seznamu (přesněji fronty) OPEN pro ukládání všech bezprostředních následníků. V upravené variantě, která byla použita v implementaci, se přidává ještě seznam CLOSED, na ukládání všech expandovaných uzlů, a do fronty OPEN se ukládají pouze bezprostřední následníci, kteří nejsou v OPEN ani CLOSED. Tato metoda je úplná a optimální. Všechny stavy ovšem zůstávají v paměti, proto je u tohoto algoritmu vysoká prostorová složitost.

Algoritmus 2.1.1. BFS (Breadth First Search – Slepé prohledávání do šířky) s využitím seznamu CLOSED

1. Sestroj frontu OPEN (bude obsahovat všechny uzly určené k expanzi) a seznam CLOSED (bude obsahovat všechny expandované uzly). Do fronty OPEN umísti počáteční uzel.

2. Je-li fronta OPEN prázdná, pak úloha nemá řešení, a ukonči prohledávání jako neúspěšné. Jinak pokračuj.

3. Vyber z čela fronty OPEN první uzel a umísti tento uzel do seznamu CLOSED.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.

5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky, kteří nejsou ve frontě OPEN, ani v seznamu CLOSED, umísti do fronty OPEN a vrať se na bod 2.

Depth First Search

DFS (Slepé prohledávání do hloubky) je metoda, kde se upřednostňuje expandování uzlu s maximální hloubkou zanoření. Základní verze DFS ovšem nezjišťuje, zda daný uzel, který má aktuálně generovat, již v minulosti zpracoval. Tím může dojít ke vzniku smyčky, ze které se algoritmus již nedostane. Proto původní verze není úplná, natož optimální. Já ve své bakalářské práci vycházím z upravené verze, kde se eliminují stejné stavy a kontrolují se předci v OPEN. Tím se metoda stává úplnou, avšak ne optimální.

Algoritmus 2.1.2. DFS (Depth First Search – Slepé prohledávání do hloubky)

1. Sestroj zásobník OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel.

2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber z vrcholu zásobníku OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.
5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky, kteří nejsou v zásobníku OPEN a nejsou ani předky generovaného uzlu, umísti do zásobníku OPEN a vrať se na bod 2.

Iterative Deepening Search

Metoda IDS (Slepé prohledávání iterativním zanořováním) vychází z algoritmu DFS. Využívá procedury DLS, která provádí prohledávání do omezené hloubky, kterou při neúspěšném prohledání inkrementuje. Tato metoda je úplná i optimální.

Algoritmus 2.1.3. IDS (Iterative Deepening Search – Slepé prohledávání iterativním zanořováním)

1. Zadej maximální hloubku prohledávání.
2. Nastav aktuální hloubku prohledávání na hodnotu 1.
3. Zavolej proceduru DLS s počátečním a koncovým stavem (seznamem koncových stavů) a s omezením na aktuální hloubku. Skončí-li tato procedura úspěchem (tj. vrátí cestu od počátečního uzlu k uzlu cílovému), ukonči prohledávání jako úspěšné (vyznač cestu nalezenou procedurou DLS). Jinak pokračuj.
4. Inkrementuj aktuální hloubku.
5. Je-li nová aktuální hloubka menší než zadaná maximální hloubka, vrať se na bod 3. Jinak ukonči prohledávání jako neúspěšné.

Algoritmus 2.1.4. Procedura DLS (Depth Limited Search – Slepé prohledávání do omezené hloubky)

1. Sestroj zásobník OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel.
2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber z vrcholu zásobníku OPEN první uzel.
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.
5. Je-li hloubka vybraného uzlu menší než zadaná maximální hloubka, tak tento uzel expanduj, všechny jeho bezprostřední následníky, kteří nejsou jeho předky a nejsou dosud v zásobníku OPEN, umísti do zásobníku OPEN. Vrať se na bod 2.

Bidirectional Search

Metodu BS (Obousměrné prohledání) lze použít při řešení úloh, kde je možnost snadno získat i reverzní operátory ke všem operátorům. Prohledávání probíhá podobně jako tomu bylo u BFS, avšak vychází se z obou směrů, tj. od počátečního i od koncového stavu. Metoda je úplná a optimální.

Algoritmus 2.1.5. BS (Bidirectional Search – Obousměrné prohledávání)

1. Sestroj fronty OPEN1 a OPEN2 (budou obsahovat všechny uzly určené k expanzi) a seznamy CLOSED1 a CLOSED2 (budou obsahovat všechny expandované uzly). Do fronty OPEN1 umísti počáteční uzel a do fronty OPEN2 cílový uzel.
2. Je-li fronta OPEN1 prázdná, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber z čela fronty OPEN1 první uzel a umísti tento uzel do seznamu CLOSED1.
4. Vybraný uzel expanduj. Pokud některý bezprostřední následník je prvkem fronty OPEN2 (je tzv. „můstek“), ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému (v CLOSED1 jsou uzly od počátečního stavu k můstku, v CLOSED2 jsou uzly od můstku k cílovému stavu), jinak ulož tohoto následníka do fronty OPEN1.
5. Vyber z čela fronty OPEN2 první uzel a umísti tento uzel do seznamu CLOSED2.
6. Vybraný uzel expanduj. Pokud některý bezprostřední následník je prvkem fronty OPEN1 („můstek“), ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému (v CLOSED1 jsou uzly od počátečního stavu k můstku, v CLOSED2 jsou uzly od můstku k cílovému stavu), jinak ulož tohoto následníka do fronty OPEN2 a jdi na bod 2.

Uniform Cost Search

Metoda UCS (Slepé prohledávání do šířky s respektováním stejných cest, či Metoda stejných cest) vychází z algoritmu BFS. Na rozdíl od něj však zohledňuje skutečné ceny přechodu a z nich vypočítaných cen cest. (Metoda BFS vycházela pouze z počtu expandování.) Ze seznamu OPEN proto k expanzi vybírá stav s nejmenším ohodnocením daným nejnižší cenou cesty.

Algoritmus 2.1.6. UCS (Uniform Cost Search – Slepé prohledávání do šířky s respektováním stejných cen)

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi) a seznam CLOSED (bude obsahovat všechny expandované uzly). Do seznamu OPEN umísti počáteční uzel včetně jeho (nulového) ohodnocení.
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s nejnižším ohodnocením a umísti tento uzel do seznamu CLOSED.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.

5. Vybraný uzel expanduj a ohodnoť všechny jeho bezprostřední následníky (cena generovaného uzlu = cena expandovaného uzlu + vzdálenost mezi oběma uzly v pixelech), kteří nejsou v seznamu CLOSED a ulož je do seznamu OPEN. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechej v tomto seznamu pouze uzel s nejnižším ohodnocením a vrať se na bod 2.

Backtracking

Metoda Backtracking vychází z prohledávání DFS. Při samotné expanzi však nedochází ke generování všech bezprostředních následníků, nýbrž jen jednoho. Při neúspěšném průchodu se metoda vrací k těmto stavům, a rozgenerovává dalšího jeho bezprostředního následníka. Pokud již dalšího rozgenerovat nemůže, pošle zprávu svému přímému předchůdci o neúspěchu. Metoda je však neúplná a není optimální, tak jako tomu bylo u DFS. Má však výhodu ve velmi nízké paměťové náročnosti, kvůli generování právě jednoho následníka a tudíž v zásobníku OPEN se nachází jen aktuální cesta.

Algoritmus 2.1.7. Backtracking (Slepé prohledávání se zpětným navracením)

1. Sestroj zásobník OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel.

2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.

3. Jde-li na uzel na vrcholu zásobníku aplikovat první, resp. jiný dosud neaplikovaný operátor, tak tento operátor aplikuj a pokračuj bodem 4, v opačném případě testovaný uzel z vrcholu zásobníku odstraň a vrať se na bod 2.

4. Je-li vygenerovaný uzel (uzel vzniklý aplikací operátoru) uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak ulož nový uzel na vrchol zásobníku (pouze však, pokud se tento uzel již v zásobníku OPEN nenachází) a vrať se na bod 2.

2.1.2 Informované metody

Informované metody využíváme při řešení úloh, kde jsou k dispozici informace o cílových stavech a aktuální stavy prohledávání je možné určitým způsobem hodnotit. Takovými informacemi může být například přibližný směr a vzdálenost výskytu cílového stavu. Při jejich využití dochází k usnadnění prohledávání, případně umožnění řešení této úlohy. [12]

Zde byly vybrány metody typu Best First Search, jejichž podstatou je nalezení stavu s nejlepším ohodnocením. To lze získat pomocí tzv. ohodnocující funkce:

$$f(n) = g(n) + h(n)$$

Cena cesty z počátečního uzlu po uzel n udává složka $g(n)$ a odhad ceny cesty z uzlu n do cílového uzlu udává složka $h(n)$. Výsledný součet $f(n)$ udává hodnotu ohodnocující funkce. [7]

V případě neinformované metody UCS se využívalo pouze hodnot $g(n)$, informovaná metoda GS (Greedy Search) bude využívat pouze hodnot $h(n)$ a informovaná metoda A* bude využívat obou těchto hodnot.

Greedy Search

Metoda Greedy Search (překládané jako Metoda lačného/hltavého prohledávání) ohodnocuje každý stav pouze na základě heuristické funkce. Tato funkce se snaží vyjádřit očekávanou cenu cesty z aktuálního uzlu do cílového. Kvalitní heuristická funkce dokáže výrazně snížit časovou náročnost prohledávání, ale metoda není úplná ani optimální. Nevýhodou tohoto algoritmu je uvíznutí v lokálním minimu. Pokud se totiž algoritmus dostane do lokálního minima, není schopen přejít do dalšího stavu a tím se ve výsledku dostat do cílového stavu.

Algoritmus 2.1.8. GS (Greedy Search – Informované „hltavé“ prohledávání)

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi) a seznam CLOSED (bude obsahovat všechny expandované uzly). Do seznamu OPEN umísti počáteční uzel včetně jeho ohodnocení (cena uzlu = přímá vzdálenost k cílovému uzlu v pixelech).
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s nejnižším ohodnocením.
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.
5. Vybraný uzel expanduj, ohodnoť všechny jeho bezprostřední následníky, kteří nejsou jeho předky a ulož je do seznamu OPEN (cena uzlu = přímá vzdálenost k cílovému uzlu v pixelech). Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechej v tomto seznamu pouze uzel s nejnižším ohodnocením a vrať se na bod 2.

A Star Search

Metoda A Star Search patří pod metody Best First Search. Hodnocení jednotlivých stavů se skládá z ceny cesty z počátečního uzlu po aktuální uzel a výsledku heuristické funkce udávající cenu cesty z aktuálního uzlu po cílový. Podmínkou pro heuristickou funkci je, že její odhad musí být stejný nebo menší (spodní odhad) než je skutečnost. Jedná se o tzv. přípustnou heuristiku. Výpočet heuristické funkce však může být velmi komplikovaný a doba zpracování může výrazně ovlivnit i celkovou dobu metody. Někdy se proto volí heuristiky jednodušší, které ovšem zapříčiní větší počet expandovaných stavů, avšak výsledná doba výpočtu je nižší. Metoda je úplná a optimální.

Algoritmus 2.1.9. A* (A Star Search – Informované prohledávání „A s hvězdičkou“)

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi). Do seznamu OPEN umísti počáteční uzel spolu s jeho ohodnocením (ohodnocení počátečního uzlu $f_0 = g_0 + h_0$, kde $g_0 = 0$ a h_0 = přímá vzdálenost počátečního uzlu k cílovému uzlu v pixelech).
2. Je-li seznam OPEN prázdný, pak úloha nemá řešení, a proto ukonči prohledávání jako neúspěšné. Jinak pokračuj.
3. Vyber ze seznamu OPEN uzel s nejnižším ohodnocením.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vyznač cestu od počátečního uzlu k uzlu cílovému. Jinak pokračuj.

5. Vybraný uzel expanduj, ohodnoť všechny jeho bezprostřední následníky, kteří nejsou jeho předky a ulož je do seznamu OPEN (cena uzlu je dána vzorcem $f = g + h$). Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponechej v tomto seznamu pouze uzel s nejnižším ohodnocením a vrať se na bod 2.

2.1.3 Metody lokálního prohledávání

V některých případech úloh není důležité nalezení ideální cesty, ale pouze nalezení cílového či optimálního stavu. Pro tyto případy se využívá metod lokálního prohledávání, které neprohledávají systematicky celý stavový prostor, ale zaměřují se na určité podoblasti a extrémy. Jsou proto schopny nalézt přijatelné řešení v rozsáhlých, teoreticky až nekonečných, stavových prostorech za použití zanedbatelné paměťové náročnosti. [12] [8]

Hill-Climbing

Hill-Climbing je metodou velmi podobnou metodě Greedy Search. Během výběru nejvhodnějšího uzlu se rozhoduje pouze na základě heuristické funkce. Tou se však snaží najít nejlepší uzel z hlediska kvality. Nemusí tedy vždy využívat směru a vzdálenosti k cílovému stavu. Tato metoda ovšem trpí stejnými nedostatky jako tomu bylo u Greedy Search, tj. problémy s lokálními extrémy. Proto není úplná ani optimální.

Algoritmus 2.1.10. Hill-Climbing (Horolezecký algoritmus – lokální prohledávání)

1. Vytvoř uzel *Current* a ulož do něj počáteční uzel spolu s jeho ohodnocením (ohodnocení = přímá vzdálenost k cílovému uzlu v pixelech).

2. Expanduj uzel *Current*, ohodnoť jeho bezprostřední následníky a vyber z nich nejlépe ohodnoceného (nazvěme jej *Next*).

3. Je-li ohodnocení uzlu *Current* lepší než ohodnocení uzlu Next, ukonči řešení a vrať jako výsledek uzel *Current* (vyznač cestu od počátečního uzlu k uzlu *Current*). Jinak pokračuj.

4. Ulož uzel *Next* do uzlu *Current* a vrať se na bod 2.

Simulated Annealing

Metoda Simulated Annealing (Simulované žíhání) vychází z algoritmu Hill-Climbing avšak snaží se zamezit jeho nedostatkům. Nejvhodnější uzel hledá stejným způsobem, avšak přechází do něj pouze s určitou pravděpodobností. V některých případech je proto schopna vyváznout z lokálních extrémů. Tato pravděpodobnost je závislá na teplotě, která se v každém kroku algoritmu snižuje až na teplotu s hodnotou nula.

Algoritmus 2.1.11. Simulated Annealing (Simulované žíhání – pravděpodobnostní prohledávání)

1. Vytvoř předpis pro klesání teploty T v závislosti na kroku výpočtu k (v implementovaném algoritmu $T = max(0, 200 - k)$).

2. Vytvoř pracovní uzel *Current* a ulož do něj počáteční uzel spolu s jeho ohodnocením (ohodnocení = přímá vzdálenost k cílovému uzlu v pixelech). Nastav krok výpočtu na nulu ($k = 0$).

3. Je-li uzel *Current* uzlem cílovým, ukonči řešení jako úspěšné a vyznač cestu od počátečního uzlu k uzlu Current.

4. Je-li $k > k_{max}$ ukonči řešení, ukonči řešení jako neúspěšné a vyznač cestu od počátečního uzlu k uzlu *Current*.

5. Expanduj uzel *Current* a z jeho bezprostředních následníků vyber náhodně jednoho z nich (nazvěme jej *Next*).

6. Vypočítej rozdíl ohodnocení uzlů *Current* a *Next*: $\Delta E = value(Next) - value(Current)$.

7. Jestliže $\Delta E > 0$, tak ulož uzel *Next* do uzlu *Current*, jinak ulož uzel *Next* do uzlu *Current* s pravděpodobností $e^{\Delta E/T}$.

8. Inkrementuj krok výpočtu k a vrať se na bod 3.

2.2 Metody řešení úloh s omezujícími podmínkami (CSP)

Existují úlohy, při jejichž řešení se musí brát ohled i na vnitřní strukturu jednotlivých stavů. Tyto stavy mohou být totiž ovlivněny omezujícími podmínkami, které mohou mít za následek změnu počtu použitelných operátorů. Hledané řešení úlohy je potom řešením, které není s těmito podmínkami v rozporu. Pro demonstraci a vysvětlení algoritmů zde bude využito problému N dam na NxN poli. [4] [2] [12] [8]

2.2.1 Forward Checking

Metoda Forward Checking je založena na principu přiřazení hodnoty proměnné a kontroly, jaké toto přiřazení mělo vliv na množinu přípustných hodnot pro zbývající proměnné. Po přiřazení totiž odebere z množin pro jednotlivé proměnné všechny hodnoty, které jsou v konfliktu s aktuálním přiřazením. Pokud některá proměnná bude mít svoji množinu prázdnou, prohlásí se aktuální stav za neúspěšný a hledá se další možné přiřazení, které by se mohlo provést místo posledních přiřazení. Metoda je úplná a optimální.

Algoritmus 2.2.1. Forward Checking (Dopředná kontrola)

1. Přiřaď ke každé proměnné Q_i (dámě ve sloupci i, i = 1...N) množinu přípustných hodnot S_i (tj. množinu čísel $\{1, 2, ..., N\}$).

2. Volej proceduru $Forward_Checking(1, X)$, kde X je vrácený konečný stav – řešení úlohy.

Algoritmus 2.2.2. Procedura Forward_Checking(i,X)

1. Odstraň první hodnotu z množiny S_i a přiřaď tuto hodnotu proměnné Q_i. Nechť Y je nový stav.

2. Je-li Y cílovým stavem, ukonči proceduru úspěchem (vrať stav Y), jinak pokračuj.

3. Odstraň z množin S_j ($j = i+1, ..., N$) všechny hodnoty, které jsou v konfliktu s dosud přiřazenými hodnotami.
4. Jestliže je nějaká množina S_j ($j = i + 1, ..., N$) prázdná, obnov původní stav množin S_j (tj. stav před bodem 3) a jdi na bod 7. Jinak pokračuj.
5. Volej proceduru $Forward_Checking(i + 1, X)$.
6. Skončí-li procedura $Forward_Checking(i + 1, X)$ úspěchem, skonči také úspěchem (vrať vrácený stav X). Jinak pokračuj.
7. Není-li množina S_i prázdná, vrať se na bod 1. Jinak skonči proceduru neúspěchem.

2.2.2 Min Conflict

Min Conflict je metoda, která na začátku vygeneruje libovolný stav, kde všechny proměnné mají přiřazenou hodnotu a následně kontroluje počet konfliktů v tomto stavu pro jednotlivé proměnné. Tyto proměnné postupně mění za účelem snížení počtu konfliktů pro daný stav. Cílem této metody je získat stav, kde se nenachází žádné konflikty. Metoda je efektivní, avšak není známo, zda je úplná a optimální.

Algoritmus 2.2.3. Min Conflict (Minimalizace konfliktů)

1. Postav dámy Q_i, tj. dámy ve sloupcích i ($i = 1, ..., N$) na libovolné pozice ($Q_i = j$, j je řádek, na kterém je postavena dáma Q_i, $j = 1...N$)) a nastav proměnnou i na hodnotu 1 (první sloupec).
2. Je-li úloha vyřešena (žádná dáma neohrožuje jinou) algoritmus úspěšně končí. Jinak pokračuj.
3. Pro každý řádek $j(j = 1...N)$ sloupce i urči počet teoreticky možných ohrožení (fiktivní) dámy na tomto sloupci dámami z ostatních sloupců, a to včetně teoreticky možných ohrožení dámami, které jsou postavené za jinými dámami!
4. Pokud ve sloupci i existuje řádek j s nižším počtem ohrožení, než je počet ohrožení na řádku, na kterém se dáma Q_i právě nachází, přesuň tuto dámu na řádek j a jdi na bod 6. Jinak pokračuj.
5. Pokud ve sloupci i existuje jiný řádek j se stejným počtem ohrožení, jako je počet ohrožení na řádku, na kterém se dáma Q_i právě nachází, přesuň tuto dámu na řádek j a pokračuj. Pokud je takových řádků více, přesuň dámu na první následující řádek se stejným počtem ohrožení, přičemž za řádkem N pokračuj řádkem 1.
6. Inkrementuj hodnotu proměnné i ($i = ((i - 1) mod N) + 1$) a vrať se na bod 2.

2.2.3 Hopfield Neural Network

Hopfieldova síť je plně propojená rekurentní neuronová síť, ve které jsou vzájemné vazby symetrické a přímé zpětné vazby nulové. Na začátku tohoto algoritmu se každému políčku šachovnice přiřadí náhodná hodnota v rozmezí 0 a 1. Následně se přepočítávají hodnoty neuronů s využití znalostí, jak dané postavení dámy ovlivní postavení dalších dam. Metoda přepočítává hodnoty, dokud v každém řádku a každém sloupci není právě jeden neuron

s uloženou hodnotou větší než 0.95. Poté se označí tyto neurony jako pozice královen a algoritmus končí. Tato metoda je součástí látky probírané v magisterském předmětu Soft Computing. [4] [1] [5]

Algoritmus 2.2.4. Hopfield (Hopfield Neural Network)

1. Vytvoř spojitou Hopfieldovu neuronovou síť, topologicky shodnou s aktuální šachovnicí (tj. $N * N$ neuronů). Každý neuron spoj s neurony ve stejném sloupci, s neurony na stejném řádku a s neurony na stejných úhlopříčkách. Doplň tuto síť pomocnými neurony a nastav váhy všech neuronů ad hoc tak, aby vynucovaly excitaci právě jednoho neuronu v každém řádku, právě jednoho neuronu v každém sloupci a maximálně jednoho neuronu v každé úhlopříčce.

2. Inicializuj všechny neurony náhodnými počátečními hodnotami z intervalu $< 0, 1)$.

3. Nastav počítadlo kroků výpočtu $k = 1$.

4. Má-li v každém sloupci a v každém řádku právě jeden neuron výstupní hodnotu větší než zvolenou hodnotu (0.95) je úloha vyřešena - zobraz tyto neurony a zobraz i hlášení o úspěšném vyřešení úlohy. Jinak pokračuj.

5. Přepočítej výstupní hodnoty všech neuronů a inkrementuj počítadlo kroků výpočtu k.

6. Je-li počet kroků výpočtu k menší než přednastavený počet (1000), vrať se na bod 4. Jinak skonči neúspěchem a zobraz chybové hlášení.

2.3 Metody hraní her

Tato část se zabývá hrou dvou hráčů, kteří se pravidelně střídají po jednotlivých tazích a snaží se čestně vyhrát nad druhým z hráčů. Oba tito hráči mají úplnou informaci o stavu hry a každý z nich ve svém tahu hledá nejlepší tah, který může provést, aby na konci hry vyhrál nad protivníkem.

V rámci této bakalářské práce je kladen důraz na složitější hry, kde se nevyužívá úplného prohledávání AND/OR grafu. Graf znázorňující stavy hry se prochází pouze do předem dané hloubky. Uzly si nesou své ohodnocení, které je dáno celočíselnou hodnotou. Hráč A, který začíná, si pak vybírá tah, který je pro něj nejvýhodnější. [12] [9]

2.3.1 MiniMax

Metoda MiniMax je založena na rekurzivním prohledávání stavového prostoru. Každý stav zde má hodnocení dané jednou číselnou hodnotou. Stavy s kladným hodnocením jsou výhodnější pro hráče A a stavy záporně hodnocené jsou výhodnější pro hráče B. Algoritmus začíná v počátečním stavu a ohodnocuje nejbližší přímé potomky tohoto stavu. Z těchto potomků pak vybírá tah vedoucí ke stavu s největším ohodnocením. Aby ovšem získal ohodnocení potomků, musí pro každého zavolat stejný algoritmus (rekurze) s hledáním potomků s nejmenším ohodnocením. Tomu totiž odpovídá tah hráče B. Zanořování probíhá do chvíle dosažení listových uzlů, či dosažením maximální hloubky udané na počátku algoritmu. Nevýhodou této metody je nutnost procházet i stavy, které jsou z pohledu hráče A zbytečné, protože například volbou tahu hráče B, jsou pro hráče A nevýhodné.

Algoritmus 2.3.1. MiniMax

1. Zavolej proceduru MiniMax, jejímž vstupním parametrem je počáteční (kořenový) uzel a výstupními parametry jsou jeho ohodnocení a optimální tah.

2. Vyznač optimální tah.

Algoritmus 2.3.2. Procedura MiniMax

1. Nazvěme předaný vstupní uzel uzlem X.

2. Je-li uzel X listem (konečným stavem hry, nebo uzlem v maximální hloubce), vrať ohodnocení tohoto uzlu. Jinak pokračuj.

3. Je-li na tahu hráč A, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X) volej proceduru MiniMax a vrať maximální z navrácených hodnot. Je-li X kořenovým uzlem vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.

4. Je-li na tahu hráč B, tak postupně pro všechny jeho možné tahy (bezprostřední následníky uzlu X) volej proceduru MiniMax a vrať minimální z navrácených hodnot.

2.3.2 AlphaBeta

Metoda AlphaBeta se do značné míry podobá algoritmu MiniMax. Pro hodnocení stavů však používá dvou proměnných tj. alpha a beta. Na počátku je alpha naplněna pro všechny uzly procházeného grafu minimální hodnotou a proměnná beta maximální možnou hodnotou. Postupným průchodem se hodnoty proměnných mění v závislosti na maximální (mění alpha) a minimální (mění beta) hodnotě jednotlivých procházených uzlů. V průběhu algoritmu se testuje, zda je hodnota alpha menší než beta. Pokud by to neplatilo, potomci se nemusejí dále zpracovávat. Přechází se do vyšší úrovně grafu (menšího zanoření).

Algoritmus 2.3.3. AlphaBeta

1. Zavolej proceduru AlphaBeta, jejímž vstupním parametrem je počáteční (kořenový) uzel, výstupním parametrem optimální tah a vstup/výstupními parametry s počátečními hodnotami $\alpha = -128, \beta = 127$ (teoreticky pak $\alpha = -\inf, \beta = \inf$).

2. Vyznač optimální tah.

Algoritmus 2.3.4. Procedura AlphaBeta

1. Nazvěme předaný vstupní uzel uzlem X.

2. Je-li uzel X listem (konečným stavem hry, nebo uzlem v maximální hloubce), vrať ohodnocení tohoto uzlu.

3. Je-li uzel typu AND (na tahu je hráč B) jdi na bod 4, jinak pokračuj (uzel je typu OR, na tahu je hráč A):

   3.1. Dokud platí nerovnost $\alpha < \beta$, tak postupně pro první/další tah (bezprostředního následníka uzlu X) volej proceduru AlphaBeta s aktuálními hodnotami proměnných α a β. Po každém vyšetřeném tahu nastav hodnotu proměnné α na maximum z aktuální a navrácené hodnoty.

3.2. Vrať aktuální hodnotu proměnné α a pro kořenový uzel vrať i tah, který vede k nejlépe ohodnocenému bezprostřednímu následníkovi.

4. Uzel je typu AND (na tahu je hráč B):

   4.1. Dokud platí nerovnost $\alpha < \beta$, tak postupně pro první/další tah (bezprostředního následníka uzlu X) volej proceduru AlfaBeta s aktuálními hodnotami proměnných α a β. Po každém vyšetřeném tahu nastav hodnotu proměnné β na minimum z aktuální a navrácené hodnoty.

   4.2. Vrať aktuální hodnotu proměnné β.

2.4 Rozpoznávání

Úlohy zabývající se rozpoznáváním se zaměřují na zařazování objektů, jevů a situací z reálného světa do tříd. Každý z těchto objektů je jedinečný a právě třída představuje jistou generalizaci. V následujících případech se bude pracovat s dvourozměrným obrazovým prostorem a právě každému obrazu, tj. reprezentaci objektu, se algoritmy budou snažit přiřadit vhodnou třídu. Tyto algoritmy můžeme označit názvem klasifikátory, ačkoliv pojem jako takový má širší význam. [12] [10] [6]

2.4.1 Lineární klasifikátor

Lineární klasifikátor pro dichotomii, který je v této práci prezentován, se pokouší rozdělit vektory $\overrightarrow{x}_i$ do dvou tříd. Těmto dvěma třídám přiřazuje opačné hodnoty (pro první hodnotu $d_i = -1$, a pro druhou $d_i = 1$). Na základě následujícího algoritmu pak vypočítá vektor vah a následně pozici dělící přímky.

Algoritmus 2.4.1. Lineární klasifikátor

1. Vynuluj vektor vah $\overrightarrow{q} = \overrightarrow{0}$.

2. Nastav indikátor změny $modify = false$

3. Proveď pro každý vektor $\overrightarrow{x}(i)(i = 1...P)$:

   Pokud vektor $\overrightarrow{x}(i)$ není správně klasifikován, tj. je-li $sign\Big(g(\overrightarrow{x_i})\Big) \neq d_i$, uprav vektor vah podle vztahu:

   $\overrightarrow{q} \leftarrow \overrightarrow{q} + \frac{\mu d_i \overrightarrow{x}_i}{\|\overrightarrow{x}_i\|}$

   $\|\overrightarrow{x}\|^2 = x_0^2 + x_1^2 + ... + x_n^2$

   (symbolem μ je označen koeficient učení) a nastav indikátor změny $modify = true$.

4. Pokud došlo k úpravě vektoru vah ($modify == true$) vrať se na bod 2.

2.4.2 Nelineární klasifikátor

Nelineární klasifikátor, který je zde uveden, má za úkol rozdělit vektory $\overrightarrow{x}_i$ do tří (obecně N) tříd. K učení tohoto klasifikátoru byla použita tzv. metoda odhadu. K rozdělení již nevyužívá dělící přímky, ale křivky, které rozdělují celkovou obrazovou plochu do tří podprostorů.

Algoritmus 2.4.2. Nelineární klasifikátor

1. Pro všechny třídy nastav: $\overrightarrow{\mu_r} = \overrightarrow{0}$; $\overrightarrow{\sigma_r} = \overrightarrow{0}$; $P_r = 0$;

2. Pro všechny vzorky $\left(\overrightarrow{x}(i), d(i)\right), i = 1...P$, počítej:

   $\overrightarrow{\mu} = \overrightarrow{\mu_{d_i}}$;

   $\overrightarrow{\mu_{d_i}} = \left(P_{d_i} * \overrightarrow{\mu_{d_i}} + \overrightarrow{x_i}\right) / \left(P_{d_i} + 1\right)$

   Jestliže $P_{d_i} > 0$ počítej:

   $\overrightarrow{v} = (\overrightarrow{x_i} - \overrightarrow{\mu_{d_i}})$;

   $\overrightarrow{w} = (\overrightarrow{\mu} - \overrightarrow{\mu_{d_i}})$;

   $\overrightarrow{\sigma_{d_i}} = \left((P_{d_i} - 1)\overrightarrow{\sigma_{d_i}} + \overrightarrow{v}^T * \overrightarrow{v} + P_{d_i} * \overrightarrow{w}^T * \overrightarrow{w}\right) / P_{d_i}$;

   $P_{d_i}++$;

3. Pro všechny třídy počítej: $p(r) = P_r / P$;

2.4.3 Perceptron

Jedná se o nejjednodušší neuronovou síť modelu zpětného propagování, která neobsahuje žádné skryté vrstvy, pouze vrstvu vstupní a vrstvu výstupní. Zde je uveden příklad neuronu s bipolární skokovou aktivační funkcí. Na základě vstupních dat, se snaží algoritmus vypočítat vektor vah $\overrightarrow{w}$, který slouží ke klasifikaci dat do dvou tříd. V každé iteraci probíhá nové přepočítání těchto vah. Po provedení série iterací se vypočítá poloha přímky a ta se následně vykreslí. [2]

Algoritmus 2.4.3. Perceptron

1. Inicializuj váhy neuronu malými náhodnými čísly (v tomto appletu jsou počáteční váhy nastaveny na nulu, kvůli možnostem opakování/krokování stejného řešení).

2. Nastav $modify = true$, $max = 1000000$, $k = 1$.

3. Dokud $modify$ and $(k < max)$ opakuj:

   3.1. Nastav $modify = false$.

   3.2. Vyber vzorek $\overrightarrow{z}(kk)$, $kk = \left[(k-1) mod P\right] + 1$ (jde o opakovaný sekvenční výběr z P bodů na pracovní ploše).

   3.3. Proveď jeden krok učení (pokud některý vzorek je chybně klasifikován, změň váhy neuronu; symbolem μ je označen koeficient učení):

   $$\begin{array}{ll} \overrightarrow{w}(k+1) = \overrightarrow{w}(k) + 2 \cdot \mu \cdot \overrightarrow{z}(kk) & pro\left(\overrightarrow{w}(k) \cdot z(kk)\right) \leq 0 \\ \overrightarrow{w}(k+1) = \overrightarrow{w}(k) & jinak \end{array}$$

   3.4. Jestliže došlo ke změně vektoru vah, nastav $modify = true$.

   3.5. Inkrementuj krok učení ($k = k + 1$).

4. Při případném dalším stisku tlačítka Perceptron nastav $max = max + 1000000$, $modify = true$ a vrať se na bod 3.

2.4.4 Pocket

Pocket je rozšířením neuronové sítě Perceptron. Při učení této sítě se hledá váhový vektor $\overrightarrow{w}^*$, pro který je počet chybně klasifikovaných vstupních vektorů minimální. Algoritmus končí v momentě, kdy lepší rozdělení vstupních prvků nemůže nalézt. Po každé sérii iterací se vykreslí dělící přímka na vypočítanou pozici. [2]

Algoritmus 2.4.4. Pocket

1. Inicializuj váhy neuronu malými náhodnými čísly z intervalu $\langle 0, 1)$.
2. Nastav $max = 1000000$, $k = 1$, $best_length = 0$, $current_length = 0$, $pocket = \overrightarrow{w}(1)$.
3. Dokud $k < max$ opakuj:
   - 3.1. Náhodně vyber vzorek $\overrightarrow{z}$ z P bodů na pracovní ploše.
   - 3.2. Pokud je vzorek správně klasifikován $\overrightarrow{w}(k)^T \cdot \overrightarrow{z} > 0$ tak inkrementuj hodnotu $current_length$ a nastav $\overrightarrow{w}(k+1) = \overrightarrow{w}(k)$, jinak:
     - 3.2.1. Jestliže $best_length < current_length$ a jestliže současně neuron s aktuálním váhovým vektorem chybně klasifikuje méně vzorků (ze všech P vzorků), než s váhovým vektorem uloženým v pocket, tak $best_length = current_length$ a $pocket = \overrightarrow{w}(k)$.
     - 3.2.2. Uprav váhový vektor (symbolem μ je opět označen koeficient učení):

       $$\overrightarrow{w}(k+1) = \overrightarrow{w}(k) + 2 \cdot \mu \cdot \overrightarrow{z}(kk)$$

     - 3.2.3. Nastav $current_length = 0$.
   - 3.3. Inkrementuj krok učení $k = k + 1$.
4. Jestliže $best_length < current_length$ a jestliže současně neuron s aktuálním váhovým vektorem chybně klasifikuje méně vzorků (ze všech P vzorků), než s váhovým vektorem uloženým v pocket, tak $pocket = \overrightarrow{w}(max)$. $w = pocket$
5. Při případném dalším stisku tlačítka *Pocket* nastav $max = max + 1000000$, $current_length = 0$ a vrať se na bod 3.

2.5 Strojové učení

Strojové učení je schopnost inteligentních systémů zlepšovat své rozhodovací schopnosti a znalosti, aby danou úlohu či úlohu podobnou bylo možné vyřešit efektivněji. Takovéto trénování může probíhat různým způsobem. Učení může provádět s učitelem, který po každé akci informuje systém o hodnocení posledního kroku. Opačným příkladem je učení bez učitele, kde se algoritmy snaží obrazce přiřadit do shluků s podobnými vlastnostmi. Dále existuje takzvané posilované učení, které je založené na přidělování různě velkých odměn po dokončení všech akcí nebo po jednotlivých akcích. Tyto odměny mohou být v případě úspěšné série akcí kladné (pozitivní), či v případě neúspěchu záporné. [12] [11]

2.5.1 K-means clustering

Metoda K-means clustering se řadí do metod učení bez učitele, tudíž systém nedostává žádné informace o správnosti klasifikace. Metoda pouze přiděluje body vstupní množiny do shluků (tzv. clusterů), které tvoří jednotlivé třídy. Každý vstupní bod přiřadí na základě nejkratší vzdálenosti k těžišti příslušného shluku. Po přiřazení všech bodů vstupní množiny, přepočítá u každého shluku jeho těžiště a opakuje celé přiřazování. Metoda končí, až se složení jednotlivých shluků nemění. [1]

Algoritmus 2.5.1. K-means clustering

1. Inicializuj k prototypů (použij náhodně vybrané, ale různé vektory/vzorky/body:

$$\overrightarrow{w_j} = \overrightarrow{x_p}, j \in \langle 1, k \rangle, p \in \langle 1, P \rangle$$

2. Přiřaď každý vektor $\overrightarrow{x_p}$ do shluku $C_j, j \in \langle 1, k \rangle$, jehož prototyp má od tohoto vektoru nejmenší vzdálenost:

$$|\overrightarrow{x_p} - \overrightarrow{w_j}| \leq |\overrightarrow{x_p} - \overrightarrow{w_i}|, j \in \langle 1, k \rangle$$

3. Pro každý shluk $C_j, j \in \langle 1, k \rangle$ přepočítej prototyp $\overrightarrow{w_j}$ tak, aby byl těžištěm všech vektorů, které jsou k tomuto shluku právě přiřazené (nechť n_j je počet těchto vektorů):

$$\overrightarrow{w_j} = \frac{\sum_{\overrightarrow{x_i} \in C_j} \overrightarrow{x_i}}{n_j}$$

4. Vypočítej chybu aktuálního stavu shlukování:

$$E = \sum_{j=1}^{k} \sum_{\overrightarrow{x_i} \in C_j} |\overrightarrow{x_i} - \overrightarrow{w_j}|^2$$

5. Pokud chyba E klesla, nebo pokud byl některý vektor přiřazen k jinému shluku, vrať se na bod 2.

Pro příklad řešení posilovaného učení bude v následujících algoritmech využito procházení bludiště. Postava, reprezentující učící se systém, bude procházet jednotlivými místy a hledat cestu na políčka s odměnou a snažit se vyhýbat místům s penalizací. Pro pochopení metod jsem vycházela ze slidů předmětu a knihy Reinforcement Learning [3]

2.5.2 TD learning

Metoda TD learning prochází náhodně bludiště, přičemž po každé takovéto procházce hodnotí stavy poslední cesty. Jedna procházka skončí, až postava dojde do políčka s odměnou (kladnou či zápornou). Postupně se tak snaží ohodnotit všechny dosažitelná místa bludiště a tím získat ideální cestu ke kladně hodnoceným místům.

Algoritmus 2.5.2. TD learning

1. Zvolte hodnoty koeficientů α a γ ($0 < \alpha \leq 1; 0 < \gamma \leq 1$) a vynulujte ohodnocení $U^{\pi}(s)$ všech stavů. Dále vynulujte počítadlo procházek $p = 0$ a nastavte jejich maximální počet p_{max}. Nastavte $start \rightarrow s$.

2. Generujte nový stav s' s použitím strategie π
3. Je-li stav s' cílovým stavem stavem, pak mu přiřaďte hodnotu $U^\pi(s') = r(s')$.
4. Vypočítejte novou hodnotu stavu pomocí vztahu:

$$U^\pi(s) = U^\pi(s) + \alpha\Big(r(s) + \gamma U^\pi(s') - U^\pi(s)\Big)$$

5. Je-li stav s' cílovým stavem, pak $p + 1 \to p$, $start \to s$, jinak $s' \to s$.
6. Je-li $p < p_{max}$, pak se vraťte na bod 2.

2.5.3 Q learning

Metoda Q learning je založena na hodnocení akcí v jednotlivých stavech. Systém pak podle tohoto hodnocení rozhoduje, kterou akci zvolí jako následující. Do značné míry se podobá metodě TD learning, avšak patří mezi aktivní přístupy k učení.

Algoritmus 2.5.3. Q learning

1. Zvolte hodnoty koeficientů α a γ ($0 < \alpha \leq 1$; $0 < \gamma \leq 1$) a vynulujte ohodnocení $Q(s, a)$ všech akcí a ve všech stavech s. Dále vynulujte počítadlo procházek $p = 0$ a nastavte jejich maximální počet p_{max}. Nastavte $start \to s$.
2. Vyberte akci a, která povede k přechodu ze stavu s do stavu s'.
3. Je-li stav s' cílovým stavem stavem, pak $Q(s, a) = r(s')$.
4. Vypočítejte novou hodnotu akce a ve stavu s pomocí vztahu:

$$Q(s, a) = U(s, a) + \alpha\Big(r(s) + \gamma \underset{a'}{max} Q(s', a') - Q(s, a)\Big)$$

5. Je-li stav s' cílovým stavem, pak $p + 1 \to p$, $start \to s$, jinak $s' \to s$.
6. Je-li $p < p_{max}$, pak se vraťte na bod 2.

Kapitola 3

Analýza a návrh řešení

Úvodní etapou vývoje této bakalářské práce bylo analyzování zadání a jeho konzultování s vedoucím práce. Na základě úvodních konzultací byl vytvořen návrh aplikace a postup, jakým bude návrh později implementován. Dále mi byly poskytnuty zdrojové kódy původní verze aplikací, se kterými jsem se podrobně seznámila. Na jejich základě jsem vytvořila návrh nového softwaru, který se snažil předejít nedostatkům minulého řešení a umožnit jeho jednodušší rozšíření.

3.1 Prostudování zadání a původní aplikace

Původní aplikace byly vytvořeny jako projekt Podpora výuky předmětů Základy umělé inteligence a Soft Computing pro Fond rozvoje vysokých škol MŠMT roku 2010. Řešiteli byli Radim Dvořák, Martina Drozdová a František Vítězslav Zbořil. Jednalo se o sadu Java appletů, jejichž cílem bylo demonstrovat vybrané algoritmy z předmětu Základy umělé inteligence bakalářského studijního programu a Soft Computing magisterského studijního programu.

Tyto programy byly napsány pomocí aplikace Java applet, které bylo možné spouštět ve webových prohlížečích pomocí virtuálního stroje Javy (Java virtual machine, zkráceně JVM). Applety bylo možné vytvářet již od roku 1995. Byly velmi rychlé, proto se často využívaly i pro netriviální výpočty pro zobrazování, a jelikož se překládaly do instrukční sady Javy, bylo možné je velmi snadno přenášet mezi různými platformami. V roce 2017 však skončila jejich podpora z důvodu bezpečnosti a na podzim roku 2018 byly ze standardu Javy zcela odstraněny. Proto již nelze s využitím nejnovějších webových prohlížečů a jejich pluginů tyto aplikace spouštět a využívat při výuce.

Spuštění těchto appletů se mi podařilo nastavit v operačním systému Windows 7 ve webovém prohlížeči Internet Explorer verze 11 s využitím Java Plug-in 11.191.2 verze 8.0.1910.12. Před tímto úspěšným spuštěním jsem se o to snažila v operačním systému Windows 10 v různých verzích webového prohlížeče Firefox, kde bylo možné dočasně spustit ve verzích starších než 52, avšak od aktualizace Java Runtime Environment (JRE) na podzim 2018, se mi to již znovu nepodařilo v žádné verzi tohoto prohlížeče, a to i přes reinstalaci starší verze JRE. Totéž platilo u prohlížeče Palemoon, který vychází ze základů Firefoxu. Prohlížeč Chrome podporoval applety do verze 42, avšak tam se mi je nepodařilo spustit ani před podzimem roku 2018. Safari podporovala applety ve verzích starších než 12 a Opera ve verzích nižších než 4.x. Tyto dva prohlížeče jsem však dále netestovala.

Jak ale vypadají původní aplikace? Z pohledu zdrojového kódu je každá aplikace uložena do jednoho samostatného souboru, kde se nachází jedna třída, která dědí od třídy java.applet.Applet. Tato třída obsahuje grafické prvky a různé proměnné, které nejsou nijak od sebe třídně nebo významově oddělené. V některých algoritmech se využívá abstraktních datových typů zásobníku a fronty, které jsou definované jako pole pevné délky. Při práci se zásobníkem se pak před přidáním jednoho prvku posunou jednotlivé položky pole o jednu pozici dozadu a následně se uloží nová položka. Pokud by se ovšem celé toto pole otočilo, tj. dno zásobníku by leželo na počátku pole a vrchol zásobníku by byl pohyblivý podle toho, kolik položek by bylo uloženo, zrychlilo by se zpracovávání jednotlivých kroků algoritmu a i samotný zdrojový kód by se zpřehlednil. S využitím vhodných kontejnerů, které některé programovací jazyky nabízejí, by se algoritmy zpřehlednily ještě o stupeň více, a odpadla by nutnost využití velkého množství počitadel (velikost a zaplněnost polí, některé iterátory apod.) na globální úrovni, protože tyto informace si objektově navržené kontejnery nesou jako své atributy. Dále by bylo vhodné poukázat na porovnávání reálných čísel ekvivalencí, kde může docházet ke špatným výsledkům, protože dané podmínky se mohou na některých architekturách vyhodnotit jako nepravdivé. Čísla mohou totiž být nepřesně uložena, kde odchylka od čísla, které mělo být uloženo, může být na nejnižším významovém bitu.

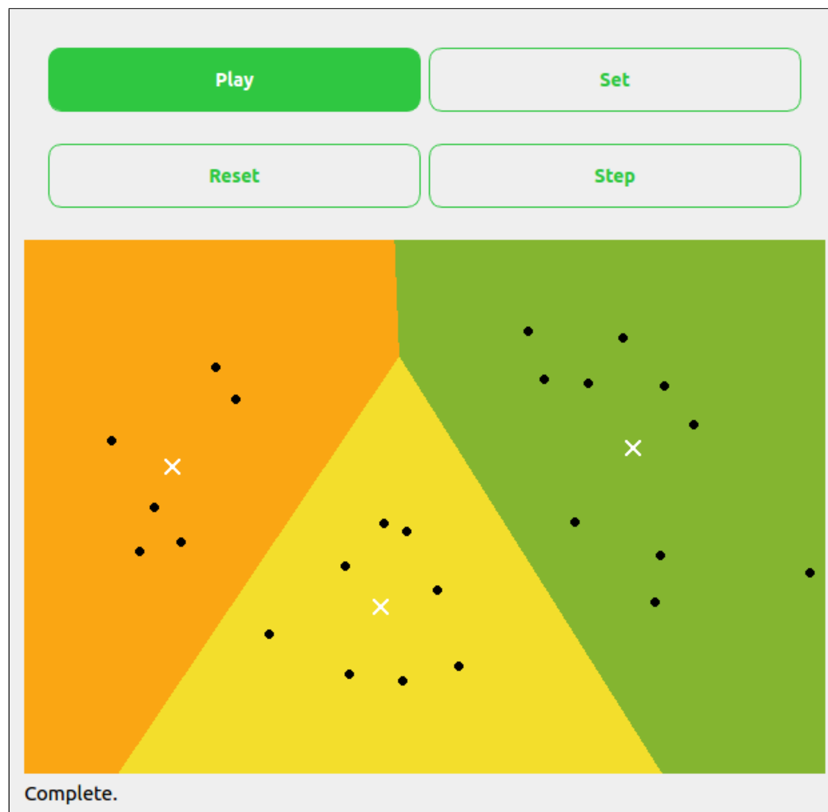
Vzhled appletů je jednoduchý, což je pozitivum. Okno je rozděleno na dvě části, kde horní část je ovládací panel a spodní samotné zobrazovací pole. Nevýhodou však může být velké množství ovládacích a nastavovacích prvků v ovládacím panelu. Dle mého názoru by bylo vhodnější a přehlednější tyto prvky od sebe oddělit. Další negativem může být nemožnost ovlivnit velikost zobrazovacího pole. Applety zaměřující se na zobrazování grafů (například AlphaBeta a procházení stavového prostoru) mohou pak trpět nedostatkem prostoru při větším počtu uzlů.

3.2 Obecná aplikace

Při tvorbě aplikací by měl být dbán důraz na lehkou rozšiřitelnost, intuitivnost jejího používání a názorného znázornění, jak jednotlivé algoritmy pracují. Pokud se oddělí ve zdrojových kódech logická část od grafického rozhraní aplikace, zjednoduší se tím rozšiřitelnost a udržovatelnost kódu. Další výhodou je skutečnost, že logická část implementace není závislá na použité knihovně pro tvorbu GUI.

V bakalářské práci Michala Klašky, který se zabýval vlivem barvy uživatelského rozhraní na výkon uživatele, lze zjistit následující: Na základě psychologických výzkumů se odborníci přiklánějí k názoru, že barvy o kratší vlnové délce uklidňují psychickou činnost a barvy o delší vlnové délce ji povzbuzují. Dále se autor zaměřuje též na to, jaký vliv mají některé barvy na únavu. Dle svého testování zjistil, že sytější barvy uživatele více unavují, ale například červené rozhraní nemělo příliš výrazné rozdíly od modrého v počtu vytvořených chyb, délce času či hodnocení spokojenosti.

Protože jsem se s podobnými informacemi setkávala jak na základní tak i střední škole, jsem se rozhodla využít méně sytých barev, kde výrazné budou využity pro důležité nebo nové informace. Pro méně významné informace jsem pak zvolila jemné odstíny, které neupoutají zvýšenou pozornost uživatele.



Obrázek 3.1: Příklad aplikace

V horní části každé aplikace se nachází navigační panel, pomocí kterého se dá ovládat zobrazovací pole ležící ve spodní části okna. Navigační panel obsahuje dva řádky, kde v prvním se nachází přepínání režimu programu. Režimy jsou dva, kdy první se zabývá nastavováním a druhý samotnou vizualizací algoritmů. V druhém řádku navigačního panelu se zobrazí možnost editace obsahu, lze si zde zvolit mezi použitými algoritmy, změnit velikost zobrazovacího pole a další různé parametry důležité pro běh aplikace. V režimu vizualizace se v druhém řádku objeví ovládací prvky pro pohyb jednotlivých kroků algoritmu.

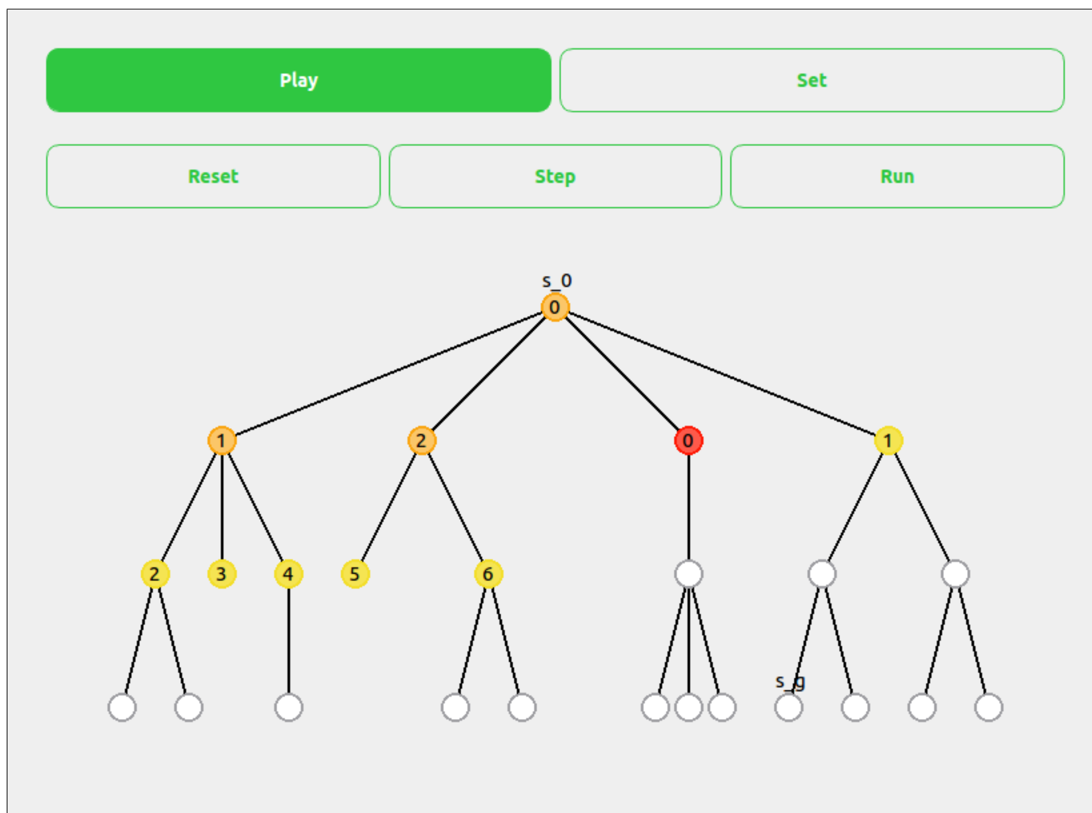
3.3 Jednotlivé aplikace

3.3.1 Prohledávání stavového prostoru

Tato aplikace by měla zobrazovat prohledávaný graf. Tento graf by mělo být možné vytvořit a upravovat, a následně procházet jednotlivými metodami. Během každého průchodu musí být odlišitelné, kde se uzly grafu nacházejí, tj. zda jsou volné, zda leží uložené v OPEN či CLOSED. V editačním režimu by také mělo jít zvětšit zobrazovací pole, kde leží graf, aby bylo možné uložit větší množství uzlů. Pro každý uzel by také mělo jít vypsat podrobnější informace, které jsou pro něj uloženy.

Každý uzel má pevně dáno, kde leží, tj. svoji pozici na zobrazovacím poli a především algoritmickou pozici. Tu lze chápat jako umístění v rámci jednotlivých kontejnerů (fronta, seznam či zásobník). Každá tato pozice by měla být pro uživatele jinak zvýrazněna. Nej-

důležitější pro uživatele je pozice aktuálního zpracovávaného uzlu. Ten musí být zobrazen výrazně a především musí být mezi ostatními ihned vidět.



Obrázek 3.2: Příklad prohledávání stavového prostoru

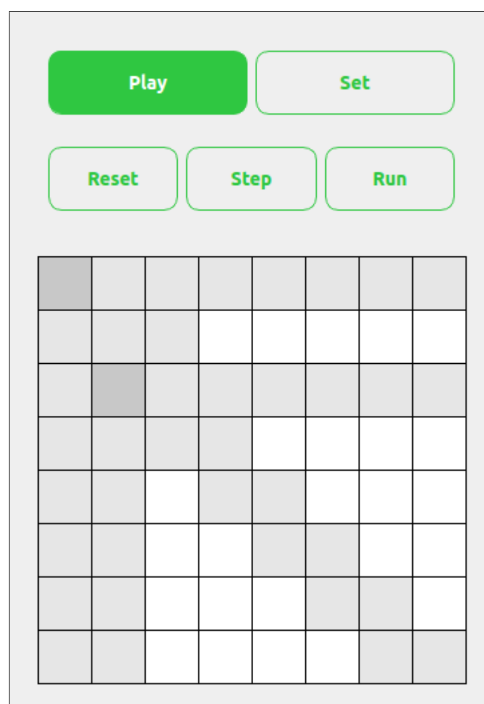
Na obrázku lze vidět, že aktuálně zpracovávaný uzel je zvýrazněn sytou červenou barvou. Nezpracované uzly jsou bílé se světle šedým okrajem a uzly v kontejneru OPEN jsou světle žluté. Uzly, které jsou uloženy v CLOSED, jsou vykreslené světlým odstínem oranžové barvy. Výsledná cesta po nalezení řešení je zvýrazněna zelenou barvou.

3.3.2 MiniMax a AlphaBeta

Základem této aplikace je vyobrazení jednoduchého grafu typu strom. Tento graf může uživatel upravovat a následně nachystat na průchod algoritmem. Při odstraňování jednotlivých uzlů, dochází k mazání i jejich podstromů. Tím jsem se snažila zabránit existenci samostatných uzlů, které nejsou na hlavní graf napojeny. Ze stejného důvodu může uživatel při tvorbě nových uzlů přidávat potomky k již existujícím uzlům. Algoritmus vnímá uložené hodnoty pouze u uzlů listových, či uzlů v nastavené maximální hloubce. Ostatní hodnoty zanedbává a pracuje s nimi pouze při změně této maximální hloubky. Během průchodu metod je zvýrazněn aktuální procházený uzel a uživatel má možnost zobrazit hodnoty proměnných pro jakýkoliv vybraný uzel. Na konci průchodu algoritmu je obarven nejlépe hodnocený tah.

3.3.3 N-queens

V této aplikaci se zobrazuje řešení metod s omezujícími podmínkami. Příkladem pro ně byl vybrán problém N dam (problém 8 dam na šachovnici zobecnění na N dam na NxN poli). Při metodě Backtracking stačí zobrazovat jen pozice jednotlivých dam, jak se na šachovnici umisťují. V metodě Forward Checking lze ovšem vykreslovat i pozice, které jsou již umístěnými dámami omezovány. Pro tato ohrožovaná políčka jsem volila světlou barvu, protože jich je na šachovnici postupem algoritmu hodně a působila by rušivým dojmem. Další metodou je pak využití Hopfieldové neuronové sítě. Pro každou pozici jsem se rozhodla vypisovat uložené hodnoty zaokrouhlené na 2 desetinná místa. Po vypočítání konečné pozice dam, jsou tato místa zvýrazněna. Poslední metodou je Min Conflict, pro který se na jednotlivé pozice plochy zobrazuje počet konfliktů s daným místem. Zvýrazněný je aktuální sloupec výpočtu a pozice dam tomu odpovídající.

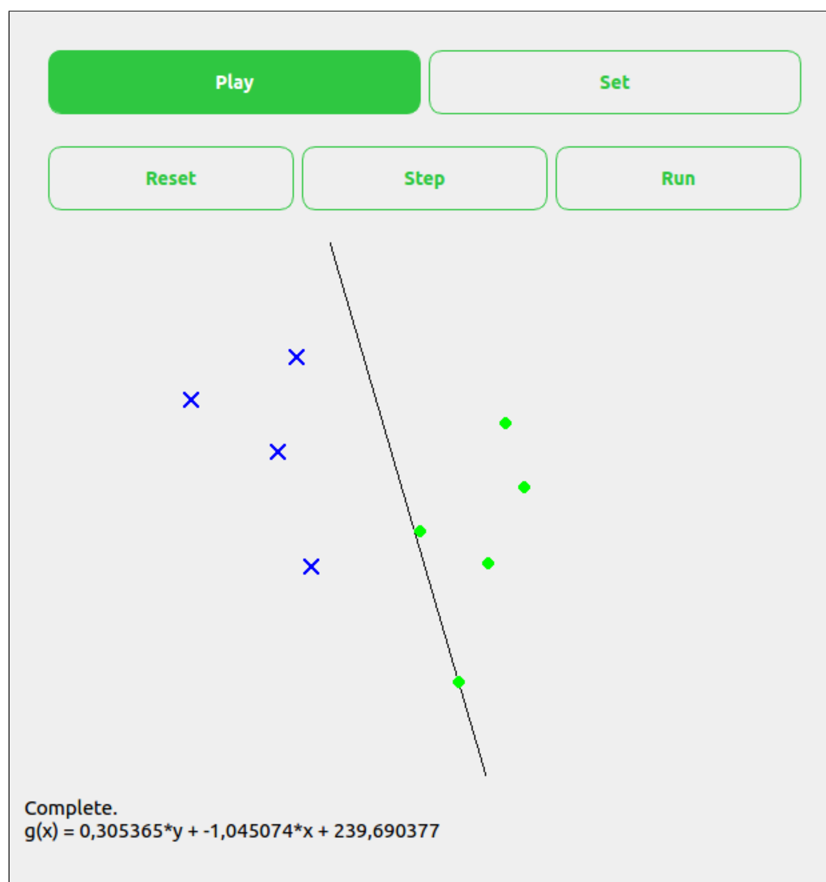


Obrázek 3.3: Příklad řešení problému N-queens pomocí metody Forward Checking

3.3.4 Lineární klasifikátor, Perceptron a Pocket

Tyto dvě aplikace jsou si velmi podobné. Mají společný cíl, a to pokud možno oddělit body dvou různých tříd od sebe. Uživatel nejprve vytvoří body a ke každému určí třídu, ke které náleží. Následně si může vybrat, zda bude postupovat po krocích nebo nechá aplikaci hledat řešení, dokud ho nenajde (případně do maximálního počtu kroků). Po ukončení každého jednotlivého kroku se vypočítá pozice dělící přímky, která se následně vykreslí na zobrazovací pole. Při výběru pouhého hledání řešení (a nezobrazování jednotlivých kroků), algoritmus provede sérii výpočtů, na jejichž konci vykreslí dělící přímku na pozici získanou posledním výpočtem. Zobrazovací plocha je doplněna o kartézskou soustavu souřadnic. Oddělení bodů pomocí různobarevného pozadí mi pro pouhé dvě třídy přišlo zbytečné. Přímka je v těchto případech dostatečně názorná. Pro lepší přehlednost, jsem se rozhodla

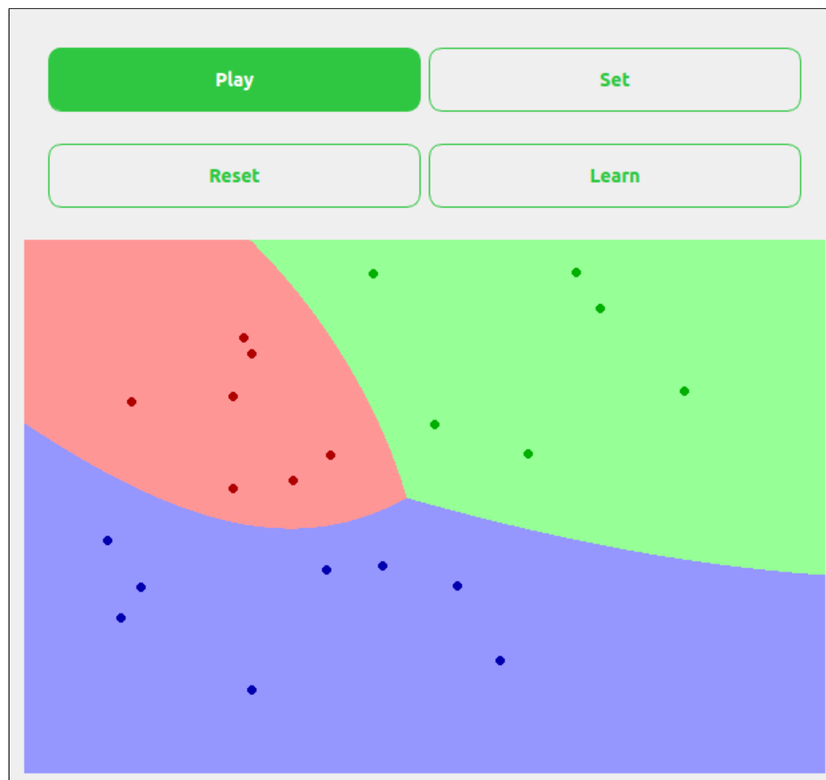
body tříd vyobrazit různými tvary, aby důvodu z větší vzdálenosti a případně špatnému rozlišení barev monitoru či plátna, šly od sebe jednoduše odlišit.



Obrázek 3.4: Příklad lineárního klasifikátoru

3.3.5 Nelineární klasifikátor

V této aplikaci lze vytvořit body různých tříd, které lze umisťovat na bílé zobrazovací pole. Poté lze spustit rozpoznávání. To obarví plochu zobrazovacího pole světlými odstíny barev z důvodu rozlišení tříd bodů vyhodnocených klasifikátorem. Plocha jedné třídy nemusí být bezpodmínečně spojitá, v některých případech může být rozdělena plochou jiné třídy a tak tvořit více ploch definujících jednu třídu.



Obrázek 3.5: Příklad nelineárního klasifikátoru

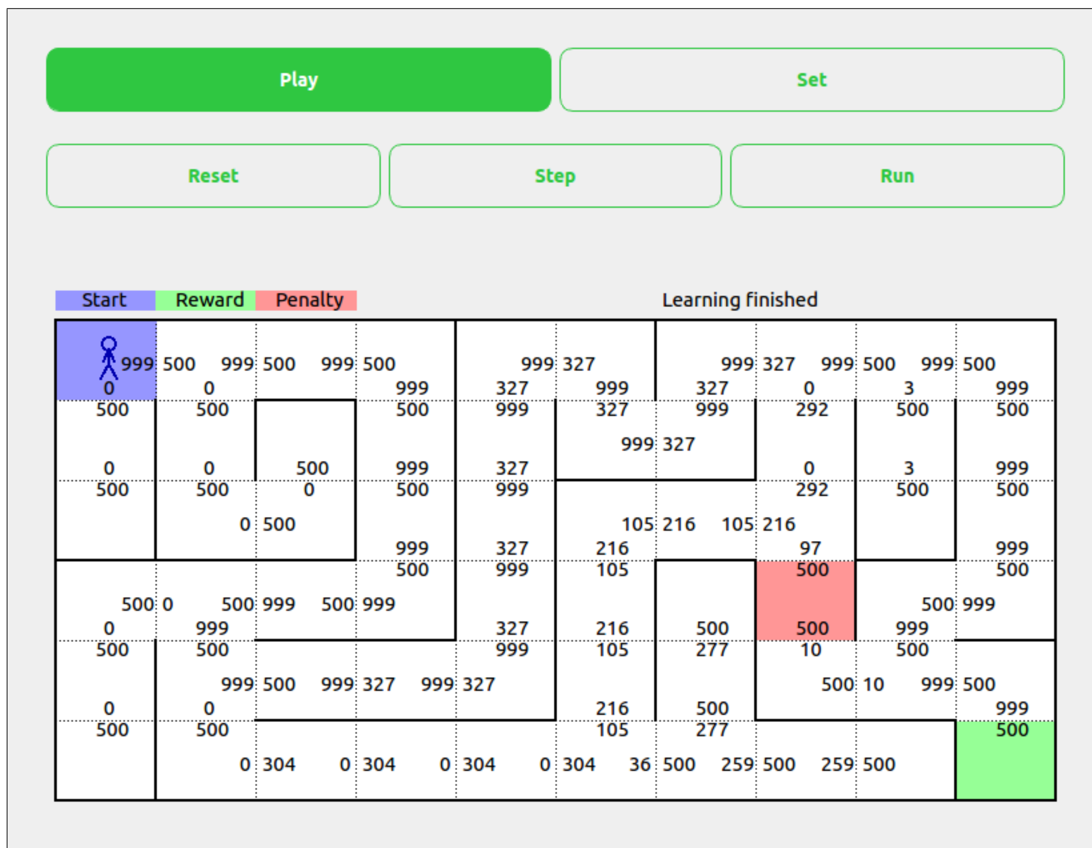
3.3.6 K-means

Uživatel na zobrazovací pole vytvoří několik bodů. Před zahájením výpočtu tyto body nejsou zatím přiřazeny třídám a jsou všechny označeny stejně. Při spuštění výpočtu se označí těžiště jednotlivých shluků a podle polohy tohoto těžiště se určí rozřazení bodů. Následně se vykreslí na zobrazovací pole území náležící jednotlivým clusterům. Každým krokem dochází ke změně barvy plochy podle změn poloh těžišť shluků. Zde jsem váhala, zda ponechat pozadí zobrazovací plochy jednolitou barvou a měnit pouze barvy bodů. Změna barev jednotlivých bodů u maximálního počtu možných shluků mi však přišla nedostatečně výrazná, a z větší vzdálenosti od monitoru nerozpoznatelná. Po několika pokusech jsem se rozhodla pro variantu, kdy vypočítávám plochy shluků, kterým poté přiřazuji odlišné barvy.

Obrázek 3.6: Příklad řešení k-means

3.3.7 Posilované učení

V této aplikaci je vyobrazeno bludiště, které se skládá z jednotlivých míst (políček) a zdí. Postavička, která bludištěm prochází, se snaží dostat na místa s odměnou a vyhýbat se místům s penaltou (záporné ohodnocení místa). Při svých cestách nemůže procházet skrze zdi. Zdi jsou vyobrazeny celistvými čarami a volné průchody čerchovaně. Počáteční stav a ohodnocená místa jsou barevně zvýrazněna světlými barvami. U každého průchozího místa je vypsáno jeho ohodnocení (v některých případech zaokrouhlené), aby uživatel mohl každým průchodem bludiště kontrolovat změny v hodnotách použitých při výpočtech.



Obrázek 3.7: Příklad posilovaného učení

Kapitola 4

Implementace a vyhodnocení

4.1 Použité technologie a cílová platforma

Všechny aplikace jsou napsané v programovacím jazyce C++ s využitím frameworku Qt. Při výběru použitých technologií byl kladen důraz, aby aplikace bylo možné spustit v prostředí serveru merlin.fit.vutbr.cz, který je přístupný všem studentům a zaměstnancům fakulty informatiky VUT v Brně, pro které je tento software určen.

Jazyk C++ je imperativní strukturální programovací jazyk podporující objektově orientovaný návrh a v současné době patří mezi nejrozšířenější programovací jazyky. Rozhodla jsem se ho ve své práci využít, protože studenti VUT FIT v Brně se s ním spolu s jazykem C setkávají v některých povinných předmětech.

Qt je framework obsahující knihovny, aplikace a nástroje určené nejen pro tvorbu grafických uživatelských prostředí. Ve své práci jsem se pro něj rozhodla zejména pro jeho kvalitní dokumentaci a velké množství návodů, ať už oficiálních nebo uživatelských.

4.2 Implementace aplikací

Logická část implementace je uložena v třídě Board. Tato třída zapouzdřuje jednotlivé metody algoritmů, kontejnery (vektory) a proměnné k těmto metodám náležejících. V těchto kontejnerech bývají obvykle uložené odkazy na objekty tříd Point (abstrakce nad bodem), Node (třída pro práci s uzly) a Edge (reprezentace hran mezi dvěma uzly).

Grafické uživatelské rozhraní je implementované v souborech main_window a g_board. Třída G_Board dědí od třídy QWidget (z knihovny frameworku Qt) a reprezentuje zobrazovací pole aplikace. Jsou zde vyobrazeny grafy, šachovnice, bludiště či kartézská soustava souřadnic. V režimu nastavování lze měnit zadání úlohy, tj. měnit graf, přidávat nebo odstraňovat body a měnit vzhled bludiště. Main_Window je třídou, která zapouzdřuje ovládací panel, výpisový panel (u některých aplikací) a objekt třídy G_Board. Ovládací panel se skládá ze 2 částí. První část obsahuje rozhraní pro změnu režimu mezi editací a vizualizací algoritmů. Druhá část se mění podle režimu a zobrazuje možnosti editace či ovládací prvky pro průchod algoritmem.

4.2.1 Aplikace state_space_search

Tato aplikace se zabývá průchodem stavového prostoru zobrazovaného jako graf. Graf se skládá z jednotlivých uzlů reprezentovaných objekty třídy Node, které nesou informace o své pozici v dvoudimenzionální kartézské soustavě souřadnic, kde počátek leží v levém horním

rohu zobrazovacího okna G_Board. Hodnota pozice na ose x roste směrem doprava a na ose y směrem dolů. Dále si každý uzel nese informaci o svém typu, tj. zda se jedná o uzel koncový, počáteční nebo obecný, a pozici, v kterém z kontejnerů (OPEN, CLOSED...) se nachází. To je uloženo ve formě výčtů možných hodnot. Každý uzel si udržuje ukazatel na svého předka a hodnoty proměnných pro informované metody průchodu.

V grafu je dále uložen kontejner obsahující hrany mezi uzly. Tyto hrany jsou objekty třídy Edge, drží si ukazatele svého počátku a konce hrany, a to zda má být hrana při vykreslování zvýrazněná. V této aplikaci uvažujeme o případu neorientovaného grafu, proto při porovnávání hran nezáleží na pořadí počátečního a koncového uzlu.

Třída Board obsahuje informace o aktuálně vybrané prohledávací metodě a stavu prohledávání. Prohledávací metody byly již popsány v teoretické části bakalářské práce a zde je lze chápat jako metody třídy, které mají za úkol vykonat jednu sérii kroků algoritmu (jedno generování a ukládání). Hrany a ukazatelé na uzly jsou ukládány do kontejneru typu vektor, aby bylo docíleno jednoduché práce s pamětí, efektivního ukládání a vybírání prvků. Lze nad nimi využít funkce na prohledávání ze standardní knihovny, které jsou pro práci s kontejnery optimalizované. Algoritmy pro průchod stavovým prostorem byly pro práci s vektory upraveny, aby nedocházelo ke zbytečnému přesunu a kopírování položek. Třída Board si udržuje informaci o velikostech zobrazovacího pole, aby nové uzly, které se mají uložit, nebyly mimo něj. Dále obsahuje pomocné proměnné pro některé metody, jako je například simulované žíhání a rozšířené zprávy uživateli, pokud byl při režimu editace nesprávně vytvořen graf.

Grafické znázornění objektu třídy Board je implementováno třídou G_Board. Tato třída zajišťuje také zpracování událostí mezi které patří mimo jiné i události vyvolané myší. Objekt této třídy dále nese informaci o tom, jaká akce a režim byly zvoleny uživatelem na ovládacím panelu a na základě nich mění své chování. Při vytváření a odstraňování uzlů zajišťuje kontrolu, aby uzly navzájem udržovaly minimální vzdálenost a nedocházelo k jejich překrytí. Vytvářet uzel lze volbou Add point pouhým kliknutím na zobrazovací pole. Pro vytvoření hrany je nutné označit jeden uzel a po kliknutí na druhý se hrana objeví.

4.2.2 Aplikace minimax_alphabeta

Tato aplikace je do značné míry podobná předešlé, tj. state_space_search. Jejím úkolem je vyobrazit graf, přesněji strom. Po dokončení průchodu algoritmů, se zvýrazní nejlépe hodnocený krok pro hráče A, jehož aktuální stav je zobrazen jako kořenový uzel. Každý uzel jakožto objekt třídy Node si nese informace o své pozici, hloubce vůči kořenovému uzlu a hodnotě. Dále je uloženo, jakého typu uzel je, to jest, zda se pro daný uzel hledá minimální či maximální hodnota ohodnocení. Nejdůležitější informací jsou ovšem samotné hodnoty alpha/maximum a beta/minimum pro daný uzel. Z jejich znalosti vycházejí algoritmy vyobrazené v této aplikaci.

Třída Board zapouzdřuje algoritmy AlphaBeta a MiniMax. Dále obsahuje kontejnery na ukládání jednotlivých uzlů, hran a OPEN. Drží si informaci o velikosti zobrazovacího pole, aktuálně vybrané metodě a stavu v jakém se nachází. Hrany jsou vyjádřeny stejnou formou, jako tomu bylo u předešlé aplikace.

Úlohou třídy G_Board je vytvořit a vykreslit již zmiňovaný strom. Při tvorbě nového uzlu lze využít metody pro přidání potomka uzlu, který je uživatelem vybrán jako rodičovský. Naopak při odstraňování uzlu uživatelem, se odebere spolu s uzlem i jeho podstrom. Tím se zamezí vzniku volných uzlů, neboli uzlů, které nejsou připojeny na strom. Pokud není

vybrána žádná editační metoda, po vybrání uzlu jsou vypsány uložené informace o tomto uzlu.

4.2.3 Aplikace n_queens

V této aplikaci je znázorněno prohledávání stavového prostoru s omezujícími podmínkami na příkladu řešení Problému osmi dam zobecněného na Problém N dam. Zde nebylo potřeba žádné pomocné třídy pro uložení pozice figurky dámy, protože stačilo ukládat pouze jednu pozici, a to pozici v daném sloupci šachovnice.

Třída Board zapouzdřuje jednotlivé metody CSP, aktuálně vybranou metodu a její stav. Velikost zobrazovacího pole zde není důležitá a objekt třídy si ukládá pouze velikost šachovnice (počet sloupců čtvercové šachovnice). Dále lze v třídě Board nalézt proměnné a pole proměnných pro ukládání pomocných informací pro jednotlivé metody.

Vykreslení šachovnice na zobrazovací pole je úkolem třídy G_Board. Její políčka mají stejnou barvu, ale jsou zvýrazněné pozice, na které nelze v algoritmech další dámu již položit. Jedinou možnou úpravou šachovnice byla změna její velikosti, protože jiná editační činnost od uživatele postrádala z pohledu algoritmů význam.

4.2.4 Aplikace lin_klas a nelin_klas

Aplikace lineárního a nelineárního klasifikátoru se snaží vyobrazit objekty reálného světa formou bodů různého typu (reprezentovaných různými symboly). Tyto body se následně snaží od sebe oddělit pomocí přímky (u lineárního klasifikátoru) či pomocí křivek (u nelineárního klasifikátoru) do tříd bodů stejného typu.

Jednotlivé body jsou implementovány třídou Point, jejichž objekty jsou definované svojí pozicí a typem. Objekty této třídy jsou pak uloženy v kontejneru jakožto atribut třídy Board. Dále si třída Board udržuje informaci o velikosti zobrazovacího pole kvůli kontrole, aby pozice žádného z bodů nebyla mimo toto zobrazovací pole. Pokud si uživatel vyžádá jeho zmenšení, může o některé body, které by byly za hranicí pole, přijít.

V zobrazovacím režimu lze lineární klasifikátor spouštět po jednotlivých krocích, kde na konci každého kroku se vykreslí dělící přímka na novou pozici. Také lze spustit výpočet, kde se projdou všechny tyto kroky, a vykreslí se pouze přímka na výsledné pozici. Nelineární klasifikátor umožňuje projít pouze všechny body, přičemž při vykreslování dochází u každého pixelu zobrazovacího pole k výpočtu, ke které třídě daný pixel náleží. Na základě této třídy je pixelu přidělena barva.

4.2.5 Aplikace perceptron

Úlohou aplikace perceptron je realizace a zobrazení algoritmů Perceptron a Pocket. Ty se snaží množinu vstupních vektorů oddělit přímkou do dvou tříd bodů stejného typu. K tomu využívají postupného učení a změny hodnot váhového vektoru.

Třída Point byla implementována podobným způsobem jako tomu bylo u lineárního klasifikátoru s tím rozdílem, že zde je nutné vyjadřovat pozici bodů i v záporných hodnotách, protože počátek kartézské soustavy souřadnic se nachází ve středu zobrazovacího pole. Třída Board si kromě kontejneru bodů nese především váhové vektory, informace o množství iterací, které má algoritmus během jednoho uživatelského kroku provést, a koeficient učení.

Na zobrazovací pole se vykreslí kartézská soustava souřadnic. Uživatel v editačním režimu může přidávat či odebírat body, dle svého uvážení. V režimu vizualizace prochází

vybraný algoritmus sérii iterací, po jejichž proběhnutí se na zobrazovací pole vykreslí dělící přímka na pozici udanou výsledkem posledního kroku.

4.2.6 Aplikace k_means

Aplikace k_means pracuje oproti ostatním výše zmíněným klasifikátorům výrazně jiným způsobem. Na počátku má k dispozici pouze pozice jednotlivých bodů, avšak nezná jejich typ. Tento typ se pouze snaží odhadnout na základě shlukování bodů do skupin, tzv. clusterů nebo shluků.

Byla vytvořena třída Node, podobná jako u lineárního klasifikátoru, avšak objektům této třídy se typ během chodu algoritmu měnil. Dále se tu objevuje nová třída Cluster, která nese informaci o pozici středu/těžiště clusteru.

Algoritmus na počátku (pseudo)náhodně vybere pozice středů jednotlivých shluků z hodnot vstupních vektorů (pozic bodů) a rozřadí všechny vstupní vektory ke clusterům, vždy k nejbližšímu podle vzdálenosti v kartézské soustavě souřadnic. Přepočítají se středy clusterů a začne se přiřazovat stejným způsobem, dokud se nezačnou ke clusterům přiřazovat stale stejné vektory. Tento algoritmus je implementován ve třídě Board.

Pro proběhnutí každého kroku algoritmu se pro každý bod zobrazovacího pole třídy G_Board najde střed nejbližšího clusteru a barvou určenou pro tento shluk se vykreslí. Dále se na zobrazovacím poli vykreslí i střed každého clusteru.

4.2.7 Aplikace reinforcement_learning

Aplikace reinforcement_learning se snaží vyobrazit jednoduché bludiště, na kterém by bylo zjevné, jak fungují metody posilovaného učení. Bludiště je složeno z míst (uzlů), které mohou mít kolem sebe stěny. Těmito místy postava v bludišti prochází a ohodnocuje cestou průchody mezi místy (uzly). Uživatel ještě před spuštěním metod označí některá místa odměnou či penalizací (zde se nebude počítat s různě velkou hodnotou, pouze s jejich existencí). Postava se pak bude snažit dostat na místa s odměnou a místům s penalizací se vyhýbat.

V třídě Board je uloženo celé bludiště jakožto pole míst. Každé toto místo může mít kolem sebe až 4 stěny (horní, pravá, spodní, levá). Pokud nějaká stěna u daného místa chybí, znamená to, že tudy vede průchod do vedlejšího místa, a tento průchod může být ohodnocen. Některé algoritmy, jako je například TD learning, si neudržují ohodnocení přechodu, ale je nutné ukládat si hodnocení stavu. Každé místo si také s sebou nese informaci, jak je uživatelem hodnoceno (odměnou či penaltou). Dále je důležité znát cestu, kterou postava z počátečního stavu již vykonala.

Třída G_Board má za úkol ze všech získaných informací vytvořit vizualizaci celého bludiště, ohodnocení míst dle dané metody a samotný posuv postavy. V editačním režimu reaguje na žádosti uživatele na úpravu průchodů (vytváření a ničení zdí), přidělování či odebírání hodnocení (odměn a penalt) a změnu počátečního místa.

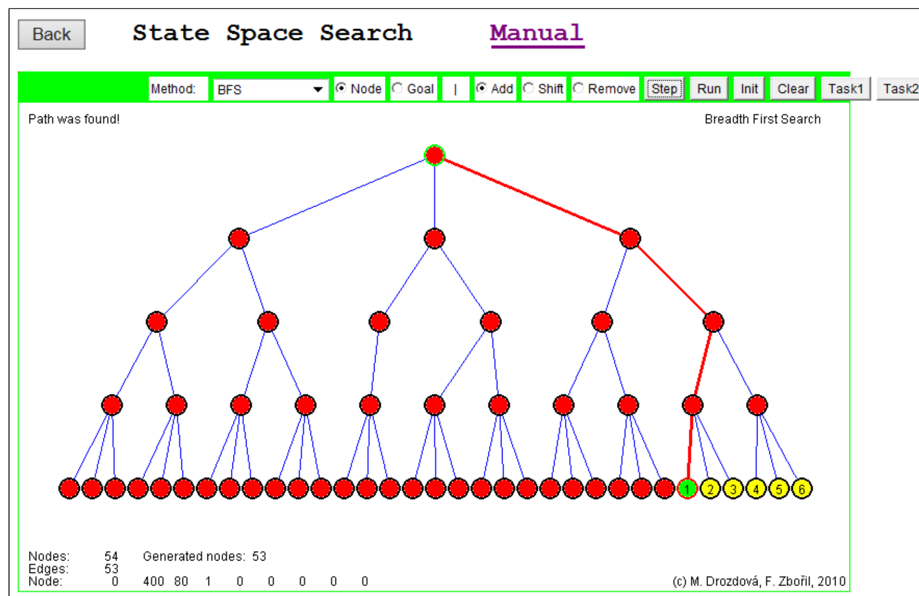
Kapitola 5

Porovnání s původní verzí

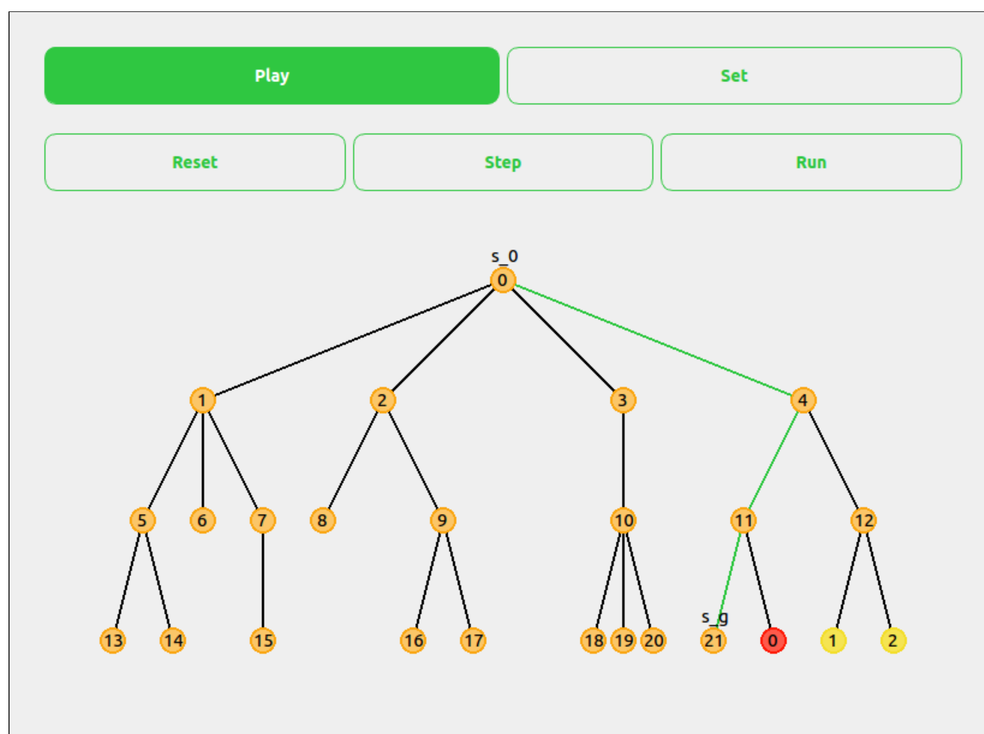
Původní verze byla vytvořena jako objekty typu Java Applet. V této bakalářské práci byly nahrazeny aplikacemi vytvořenými pomocí programovacího jazyku C++ a frameworku Qt. Toto zázemí má velmi bohatou uživatelskou základnu, a v blízké budoucnosti nehrozí, že by přišlo o podporu. I kdyby došlo k ukončení podpory frameworku Qt, zdrojové kódy logické části aplikací lze využít s jinými prostředími či grafickými knihovnami pro jazyk C++. Grafická část aplikace pak lze snadno přepsat pro zvolené prostředí. Pokud by k tomu mělo dojít, doporučila bych nejdříve zkontrolovat pozici počátku soustavy souřadnic, která by mohla mít vliv na fungování některých algoritmů (například klasifikátorů).

Základním rozdílem původní verze s novými aplikacemi z pohledu uživatele je oddělení editace zobrazovacího pole a výběru metod od samotného spouštění algoritmu. V některých aplikacích byla pozměněna vizuální stránka vykreslování zobrazovacího pole, aby probírané metody byly lépe pochopitelné. Z pohledu programátora došlo k rozdělení proměnných a významových částí do tříd, čímž se zvýšila čitelnost kódu. Lze měnit velikost zobrazovacího pole, což umožnilo přidávání většího množství prvků (například body u klasifikátorů či uzly u grafů) bez ztráty na přehlednosti.

Pro aplikace state_space_search a minimax_alphabeta byly voleny decentnější odstíny barev, aby nedocházelo k upoutání pozornosti uživatele již neměnnými či prozatím nedůležitými částmi procházeného grafu. Bylo umožněno po vybrání uzlu uživatelem vypsání uložených hodnot uzlu, především pak hodnot alpha a beta, minimum a maximum pro aplikaci minimax_alphabeta. V původní verzi nebylo uživateli umožněno sledovat tyto hodnoty, což mohlo vést ke zhoršení pochopení metody.



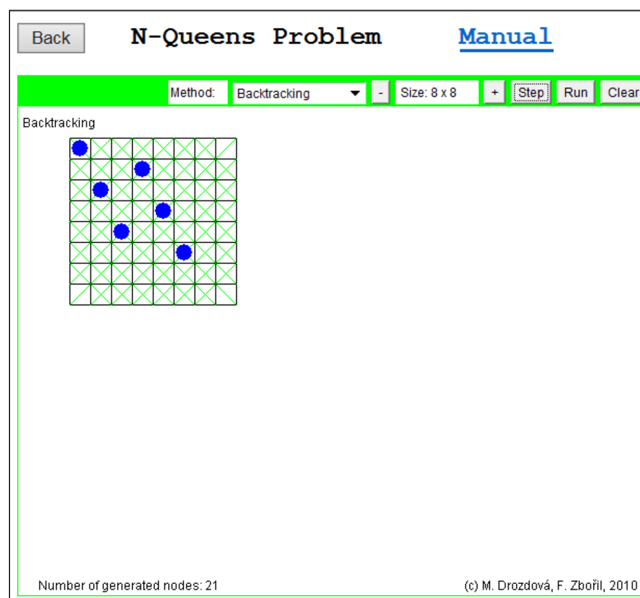
Obrázek 5.1: Příklad appletu pro řešení prohledání stavového prostoru



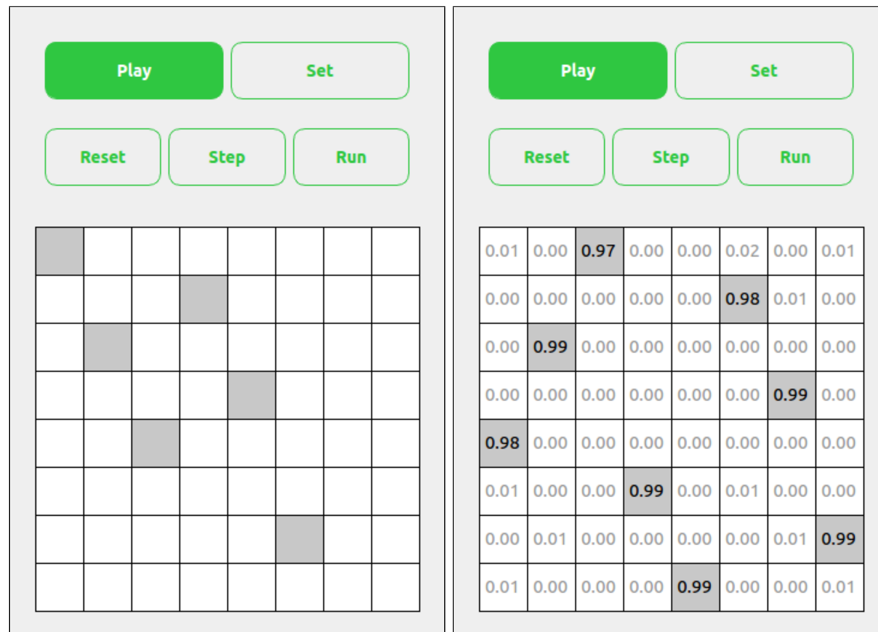
Obrázek 5.2: Příklad nové aplikace pro řešení prohledání stavového prostoru

U aplikace n_queens došlo výraznému odlišení jednotlivých metod pro získání řešení. Na pozadí všech políček není zelený kříž, který by na první pohled říkal, že toto pole je nějakou dámou ohrožováno. Tato informace byla totiž klamná, protože označeny byly všechny,

i když ohrožena některá políčka nebyla. Tyto křížky tedy byly zcela vypuštěny. V metodě Backtracking byly označeny pouze pozice položených dam. V metodě Forward Checking pak bylo "hrací"pole doplněno o zvýraznění již zmiňovaných ohrožených pozic. Min Conflict pak pro každou pozici vypisuje počet konfliktů vypočítaných algoritmem. Zvýrazněn je pak aktuálně procházený sloupec. Pro metodu s využitím Hopfieldové neuronové sítě je pak vypisována aktuální hodnota pozice a při nalezení řešení je pak toto řešení zvýrazněno.

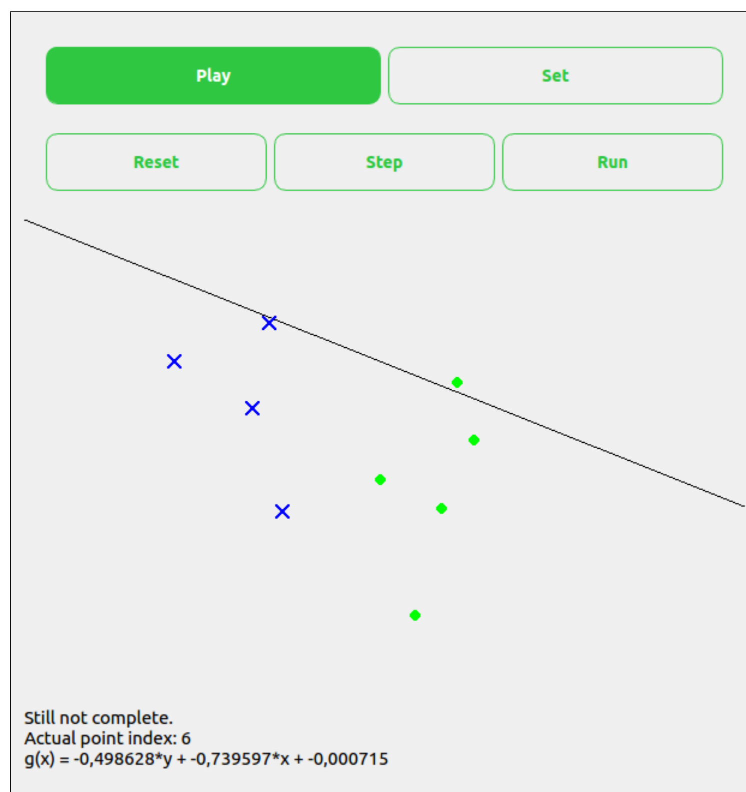


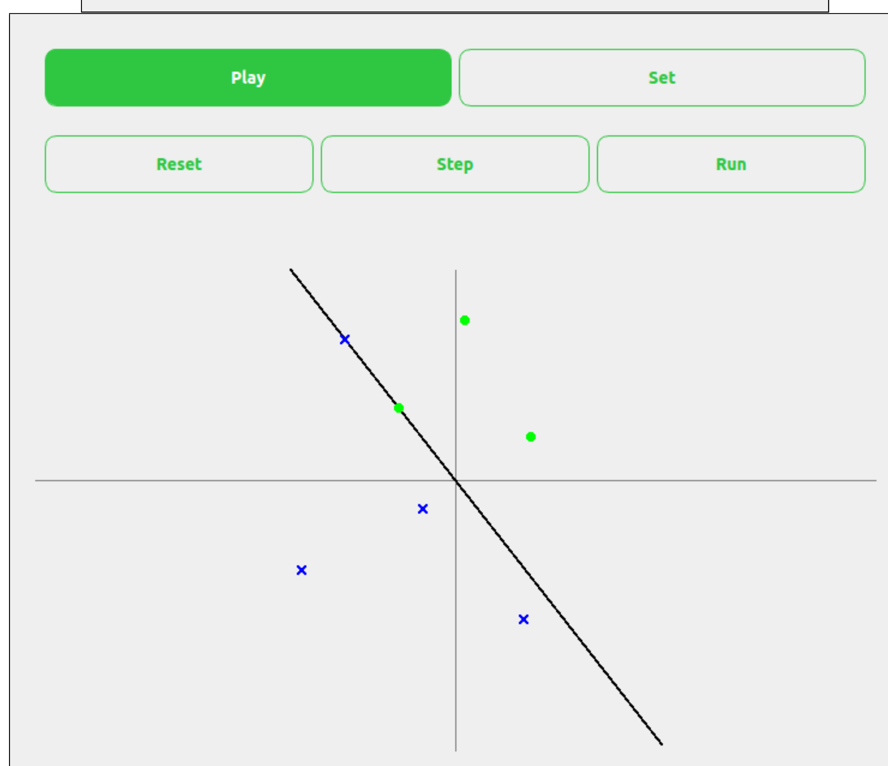
Obrázek 5.3: Applet pro řešení problému N-queens pomocí metody Backtracking



Obrázek 5.4: Nová aplikace pro řešení problému N-queens pomocí metody Backtracking a využití Hopfieldové neuronové sítě

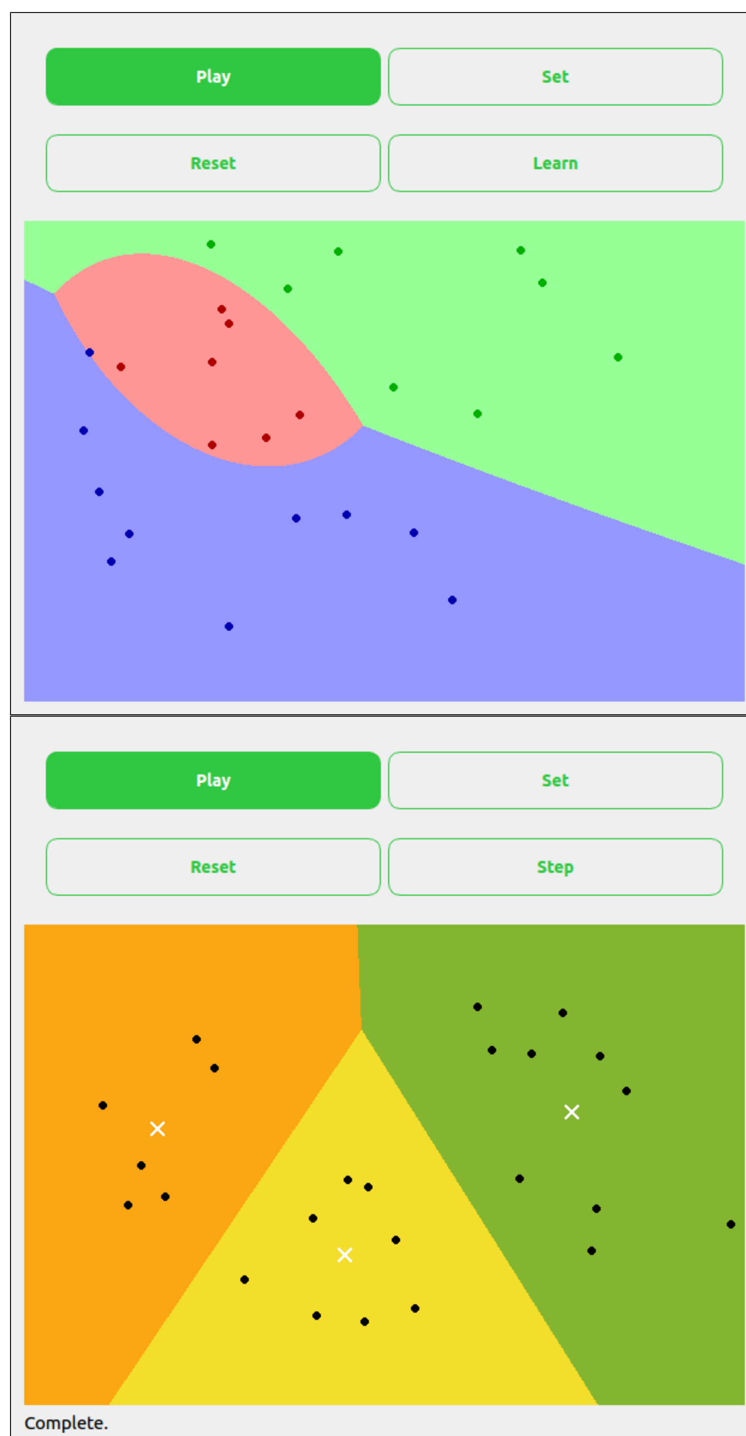
Aplikace lin_klas a aplikace perceptron se od původní verze příliš neliší. Byl zvolen jiný způsob označení bodů, aby byly na zobrazovacím poli viditelnější. Zvětšením bodů a změnou jejich barev jsou teď body různých tříd rozpoznatelné od sebe na první pohled.





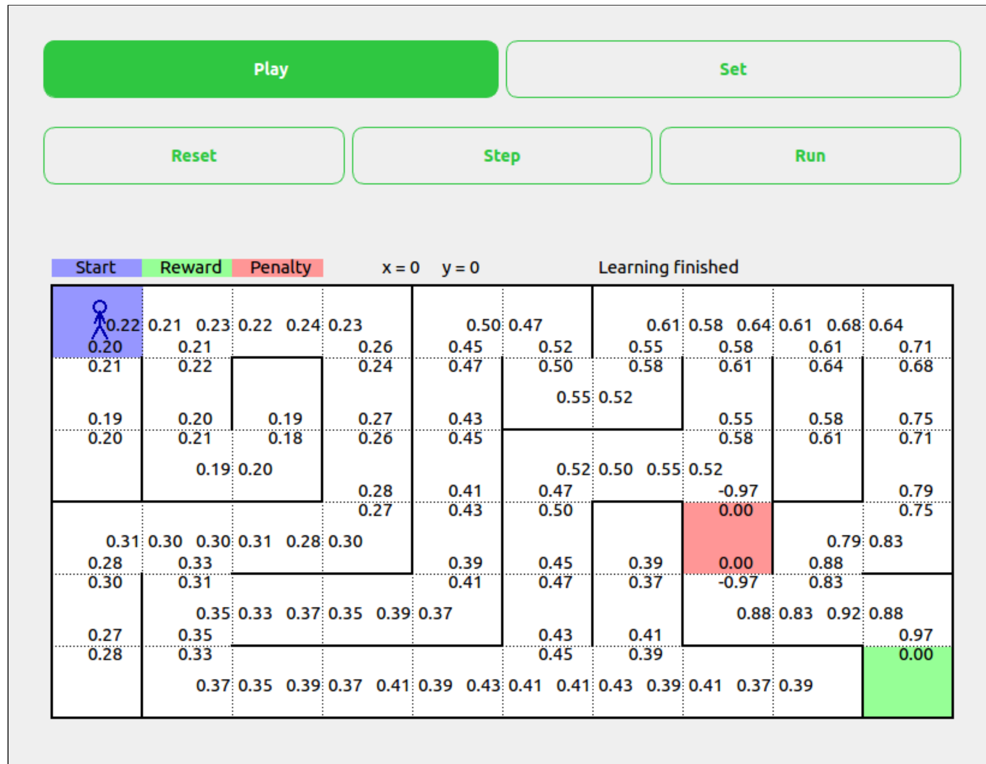
Obrázek 5.5: Nová aplikace pro lineární klasifikátor a perceptron

U aplikací nelin_klas a k_means došlo od původní verze především k barevným změnám, kde byly voleny tlumenější barvy, které jsou od sebe dobře rozpoznatelné, a samotné body byly mírně zvětšeny. U aplikace k_means pak byl změněn způsob označení těžiště shluku, aby byl na zobrazovacím poli lépe viditelný.



Obrázek 5.6: Nová aplikace pro nelineární klasifikátor a k-means

Pro aplikaci reinforcement_learning byl změněn nepatrně způsob zobrazení bludiště. Základní zdi byly zesíleny a průchody vyobrazeny čerchovaným stylem čar, který byl navíc zesvětlen. Pomocné informace pro jednotlivé metody byly vypsány tlumenější barvou.



Obrázek 5.7: Nová aplikace pro posilované učení

Kapitola 6

Shrnutí

Na privátních stránkách předmětu Základy umělé inteligence (IZU) byly nalezeny odkazy na vypracované applety, které měly být v rámci této bakalářské práce přepracovány. Od vedoucího práce dostal řešitel zdrojové kódy k těmto osmi appletům. Ty si řešitel prostudoval a navrhl změny ve zdrojových souborech, aby bylo možné je snadno rozšiřovat o nové metody nebo je nahrazovat. Tyto změny byly konzultovány s vedoucím práce a následně byly implementovány. Při této tvorbě byl kladen důraz na snadnější čitelnost kódu a jednoduchou práci s aplikacemi. Dále se práce věnuje přehlednosti zobrazovaných metod a zvýrazněním nejdůležitějších informací, především nově získaných. Pro každou aplikaci bylo vytvořeno několik příkladů, na kterých probíhala kontrola funkčnosti a správnosti aplikací.

Literatura

[1] MAŘÍK, V.; ŠTĚPÁNKOVÁ, O.; LAŽANSKÝ, J.: *Umělá inteligence 1*. Akademie věd České republiky, 1993, ISBN 8020004963.

[2] MAŘÍK, V.; ŠTĚPÁNKOVÁ, O.; LAŽANSKÝ, J.: *Umělá inteligence 4*. Akademie věd České republiky, 2003, ISBN 8020010440.

[3] SUTTON, R. S.; BARTO, A. G.: *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts : The MIT Press, 1998, ISBN 0262193981.

[4] Toshinori, M.: *Fundamentals of the New Artificial Intelligence*. Springer, 2008, ISBN 9781846288388.

[5] ZBOŘIL, F. V.: Soft computing. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/SFC/private/18sfc_5.pdf

[6] ZBOŘIL, F. V.: Soft computing. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/SFC/private/18sfc_1.pdf

[7] ZBOŘIL, F. V.: Základy umělé inteligence. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_1.pdf

[8] ZBOŘIL, F. V.: Základy umělé inteligence. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_2.pdf

[9] ZBOŘIL, F. V.: Základy umělé inteligence. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_5.pdf

[10] ZBOŘIL, F. V.: Základy umělé inteligence. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_9.pdf

[11] ZBOŘIL, F. V.: Základy umělé inteligence. [Online; navštíveno 03.06.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/1819izu_8.pdf

[12] ZBOŘIL, F. V.; ZBOŘIL, F.: Základy umělé inteligence – Studijní opora. Květen 2012, [Online; navštíveno 26.04.2019]. URL https://www.fit.vutbr.cz/study/courses/IZU/private/oporaizu-esf-5a.pdf