VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

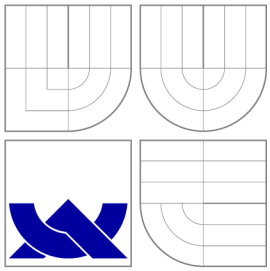
ŘEŠITEL HRY GRIDDLERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

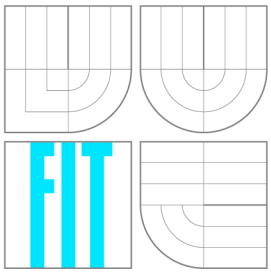
AUTOR PRÁCE
AUTHOR

LUKÁŠ BALCÁREK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ŘEŠITEL HRY GRIDDLERS

GRIDDLERS SOLVER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ BALCÁREK
AUTHOR

VEDOUCÍ PRÁCE — Mgr. FILIP GOLDEFUS
SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání bakalářské práce

Řešitel: **Balcárek Lukáš**
Obor: Informační technologie
Téma: **Řešitel hry griddlers**
Griddlers Solver
Kategorie: Umělá inteligence

Pokyny:

1. Nastudujte literaturu zabývající se hrou griddlers.
2. Po konzultaci s vedoucím navrhněte řešitele této hry.
3. Důsledně analyzujte alespoň tři uvažované algoritmy, formalizujte jejich činnost a porovnejte jejich kvality z pohledu časové a prostorové náročnosti.
4. Implementujte v některém z moderních programovacích jazyků, C++, Java, C#, ...
5. Vytvořte sadu příkladů dle pokynů vedoucího a ověřte jeho správnou činnost.
6. Zhodnoťte přínos své práce, diskutujte možná rozšíření, případné nedostatky. Podrobte program testům dle pokynů vedoucího a důkladně analyzujte algoritmy, jejich výhody a nevýhody.

Literatura:

- http://en.wikipedia.org/wiki/Griddlers

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Goldefus Filip, Mgr.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2009
Datum odevzdání: 19. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Hlavním motivem této práce je logická hra Griddlers, na které jsou ukázány různé postupy pro hledání správného řešení. Pro nalezení řešení daného zadání se používají neinformované a informované algortimy, které jsou přizpůsobeny pro tuto logickou hru. Součástí této práce je vytvořená konzolová aplikace (demonstrující hledání výsledků ze zadaných metadat), na které je možné testovat různá zadání a pozorovat měnící se výpočetní nároky. Vytvořená aplikace má za cíl jednoduchým způsobem ukázat propojení umělé inteligence, algoritmizace a zpracování dat ve zvolené hře Griddlers.

Abstract

The basic motive of this work is the logical game Griddlers. There are shown the various techniques for finding the right solution. To find solutions to the assignment are used uninformed and informed algorithms which are optimized for this logical game. Part of this work is also created console application demonstrating the search of the results from the given metadata. It is possible to test various specifications and observe the changing computing demands on them. This created application is designed to easily show the interconnection of artificial intelligence, algorithms and data processing in the selected game Griddlers.

Klíčová slova

Hra Griddlers, Nonogram, malování podle čísel, slepé prohledávání do šířky, slepé prohledávání do hloubky, modifikované prohledávání do hloubky, modifikované prohledávání do šířky, algoritmus griddler, BFS, DFS, MBFS, MDFS, neinformované metody, zpětné navracení, umělá inteligence, vytváření struktury XML souboru, řešitel hry Griddlers, testování řešitele.

Keywords

Game Griddlers, Nonogram, paint by numbers, breadth first search, depth first search, modified breadth first search, modified depth first search, algorithm griddler, BFS, DFS, MBFS, MDFS, uninformed methods, backtracking, artificial intelligence, structuring of the XML file, Griddlers solver, testing resolver.

Citace

Lukáš Balcárek: Řešitel hry Griddlers, bakalářská práce, Brno, FIT VUT v Brně, 2010

Řešitel hry Griddlers

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Filipa Goldefa. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Balcárek
12. května 2010

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé práce panu Mgr. Filipu Goldefovi, který mně byl nápomocen při vytváření této práce a s ní souvisejícího programu a který mně ochotně poskytoval konzultace a dobré rady. Dále děkuji všem v mém okolí, kteří se mnou měli trpělivost při vytváření této práce, protože bez nich bych práci nedokončil v pořádku.

Obsah

Kapitola 1

Úvod

Člověk, potažmo celé lidstvo, se v minulosti zabýval různými úlohami, jako jsou hádanky, hlavolamy a šifry, které sám vymýšlel nebo se snažil řešit. V těchto hádankách, hlavolamech a šifrách mohly být ukryté informace s různým obsahem. Informace, které člověk ukrýval do této formy komunikace, mohly obsahovat zprávy pro pobavení druhých, skrytá poselství, tajné zprávy nečitelné pro nepřítele atd.

S rozšířením informačních technologií a její dostupnosti, mohl člověk pro řešení různých úloh využívat místo svého myšlení počítače a jejich prostředky. Tato nová možnost v řešení úloh přinesla potřebu předat myšlení člověka počítači tak, aby byl schopen zadaný problém správně vyřešit. Vědecké odvětví, které se zabývá touto činností, se nazývá umělá inteligence.

Umělou inteligenci můžeme také použít pro řešení různých úkolů, hádanek a křížovek a pomocí vhodných metod nalézt požadované řešení. Delší dobu se zabývám řešením hlavolamů a logických křížovek. Mezi logické křížovky, kterými se zabývám, patří Sudoku a v nedávné době jsem se opět vrátil k řešení logické hry Griddlers.

1.1 Cíl práce

Touto prací bych chtěl ukázat jednu z možností využití umělé inteligence při řešení úloh. Pro ukázku řešení dané úlohy je zvolena logická hra Griddlers. V této hře použiji pro řešení pár základních algoritmů a postupů z oblasti umělé inteligence a posoudím jejich výhody a nevýhody z různých hledisek. Řešitel této hry bude zpracován jako konzolová aplikace se vstupními a odpovídajícími výstupními daty.

V závěru této práce bych chtěl zmínit možnosti použití a uplatnění podobných aplikací na různé odvětví lidské činnosti, zhodnotit dosažené výsledky a poukázat na další možné rozšíření této aplikace.

1.2 Prerekvizity

Text v této práci se zabývá některými základními pojmy z oborů umělé inteligence, převedení metod umělé inteligence na algoritmy, tvorbou XML dokumentů a objektovým programováním v jazyce C++.

Ke snadnějšímu pochopení zpracovávaného tématu této práce je vhodné mít alespoň základní znalosti z výše jmenovaných oborů. Každému pojmu a tématu je věnováno přiměřené

množství textu tak, aby čtenář mohl danou tématiku co nejlépe pochopit nebo si o ní doplnit nové znalosti. Většina termínů v následujícím textu je psána česky.

V případě vzniklých nejasností je možné vyhledat vysvětlení v textech o základech umělé inteligence [9], v příručce o XML dokumentech [3] a základní informace s údaji o hře Griddlers, která je vysvětlena v následující kapitole, je možné nalézt zde [8], [5].

1.3 Stručný obsah

V této první kapitole jsme se zabývali lehkým uvedením řešeného tématu do rozsáhlejšího kontextu. Další kapitoly budou obsahovat podrobnější rozbor této problematiky.

V pořadí druhá kapitola vysvětlí základní pojmy hry Griddlers, podstatu této logické hry a její možné varianty. Nedílnou součástí této kapitoly je také vysvětlení pravidel a základních způsobů řešení logické hry.

Třetí kapitola se bude zabývat návrhem řešitele, jeho vlastnostmi, uložením vstupních a výstupních dat a kontrolou zadaných vstupních dat.

Další neméně důležitá kapitola (v pořadí již čtvrtá) se věnuje analýze použitých algoritmů, jejich vlastnostem a vzájemným porovnáním algoritmů.

Pátá kapitola je zaměřena na samotnou implementaci aplikace pro řešení hry Griddlers a její vlastnosti.

V šesté kapitole bude zmíněn způsob testování aplikace pomocí sady příkladů a ověření správného vyhodnocení zadaných příkladů.

Předposlední sedmá kapitola bude obsahovat přínos této práce, návrh možných rozšíření, zhodnocení předností a případných nedostatků. V této části budou zmíněny také výhody a nevýhody použitých algoritmů.

Závěrečná osmá kapitola přináší celkové zhodnocení práce, její přínos a případné možnosti dalšího rozvoje této problematiky.

Kapitola 2

Hra Griddlers

Ve druhé kapitole se hlouběji seznámíme s hrou Griddlers, základními pojmy, pravidly hry a různými variantami řešení této hry. Okrajově zde budou zmíněny různé varianty této hry.

Logická hra Griddlers [5] je také někdy nazývaná jako Nonogram, logické hádanky, malování podle čísel nebo kódované obrázky. Zadání v sobě ukrývá skrytý obrázek, který je možné zjistit řešením této hry.

Hra Griddlers obsahuje pravoúhlou mřížku se čtvercovými buňkami, do kterých se doplňují určité barvy, a zadání vedle každého řádku a nad každým sloupcem. Pomocí zadání v řádcích a sloupcích vzniká výsledný obrázek. Podrobnější popis hry se nachází v podkapitole 2.1 Logické hádanky. Příklad zadání skrytého obrazce můžeme vidět na obrázku 2.1 a jeho správné řešení je zobrazeno na obrázku 2.2.

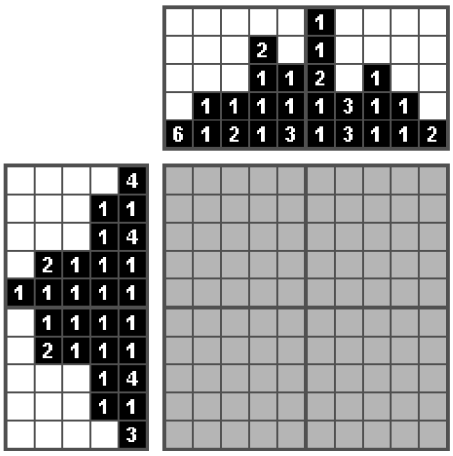
Vyřešené obrázky v mřížkách jsou nejčastěji v černobílém provedení, ale existuji i další různé varianty a rozšíření této hry. Rozšíření hry spočívá v použití více barev než jenom černé a bíle, např. také červené, zelené, modré atd. Další možnou variantou této hry jsou obrázky, které mají v zadání čtverce a trojúhelníky. Varianta s trojúhelníky umožňuje vytvářet obrázky s lepším vzhledem, protože se nebudou skládat pouze z čtvercových buněk, ale budou se doplňovat do buněk i trojúhelníky, které opticky lépe zaoblí výsledné obrázky. Mezi další varianty hry Griddlers se dá započítat hra Triddlers, která má podobná pravidla, ale pro luštění obrázku se používá jiná mřížka. U hry Triddlers je namísto pravoúhlé mřížky použita šestistěnná mřížka skládající se z trojúhelníkových buněk.

Pro ukázání základních principů řešitele hry Griddlers dále uvažujme pouze černobílé obrázky (např.: zadání obrázku 2.1 a řešení obrázku 2.2) skládající se z černých a bílých čtverců. V kapitole 4 zabývající se algoritmy, budou zmíněné algoritmy pracovat se základní verzí hry bez dalších rozšíření. Tuto variantu jsem zvolil pro snadnější pochopení dané tématiky a pro lepší pochopení činnosti algoritmů.

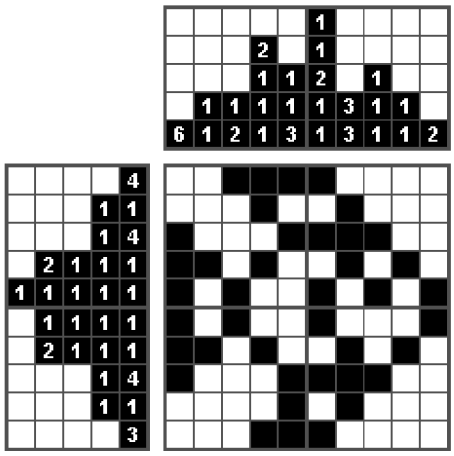
2.1 Logické hádanky

Hru Griddlers [5] můžeme také nazývat “logickými hádankami”, protože logickým uvažováním nad příslušným zadáním se snažíme nalézt správné řešení. V případě správného postupu není potřeba hádat správné řešení. Samotné odhadování výsledneho řešení je poslední možnost nalezení řešení, když všechny ostatní možnosti nejsou použitelné. Pravidla řešení jsou uvedena v podkapitole 2.2 a základní metody pro řešení jsou uvedeny v podkapitole 2.3. Nyní však pokračujme v popisu jednotlivých částí hry Griddlers.

Struktura hry se skládá z několika základních částí. V následujícím výčtu budou roze-



Obrázek 2.1: Příklad zadání hry Griddlers o rozměru mřížky 10x10 buněk.



Obrázek 2.2: Správné řešení hry Griddlers zachycující obrázek ryby.

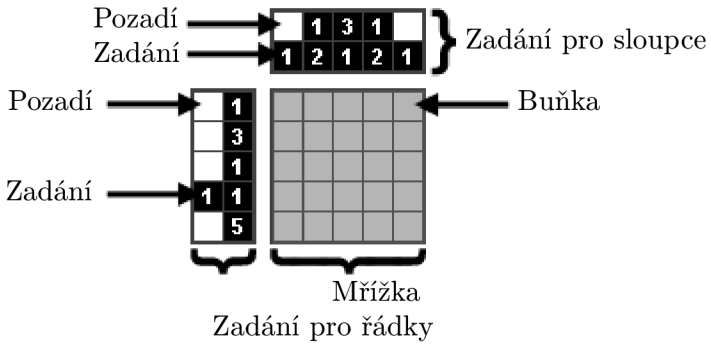
brány jednotlivé části této hry s jednoduchým komentářem ke každé části. Pro lepší názornost můžeme vycházet z obrázku 2.3, který zachycuje jednotlivé body výčtu.

Popis jednotlivých částí hry Griddlers:

- **Mřížka** – místo, které obsahuje výsledný obrázek. Rozměr mřížky není teoreticky omezen. Mřížka může obsahovat stejný počet řádků a sloupců (např.: 10 řádků a 10 sloupců) nebo rozdílný počet řádků a sloupců (např.: 20 řádků a 10 sloupců). Mřížka se skládá z jednotlivých buněk.
- **Buňka** – nejmenší prvek hry Griddlers, který nese barevný údaj o jednom čtverci mřížky. Buňka může nabývat, u černobílé varianty, třech barevných hodnot: barva popředí, barva pozadí a barva neřešené buňky. Pro barvu popředí je většinou volena černá barva, pro barvu pozadí je většinou volena bílá barva a pro barvu neřešené buňky je většinou volena šedá barva. Barva popředí a pozadí může být i jiná. Barvy vyřešené buňky vycházejí z barev "zadání" a "pozadí" na okraji mřížky. Pro další text předpokládejme, že barva popředí odpovídá černé barvě a barva pozadí odpovídá bílé barvě.
- **Zadání pro řádky** – vedle každého z řádku na okraji mřížky se vyskytuje záznam v číselné podobě nebo zde není uveden žádný záznam. Tyto záznamy udávají výslednou podobu zakódovaného obrázku. Přesný význam těchto hodnot bude vysvětlen v podkapitole 2.2 Pravidla řešení.
- **Zadání pro sloupce** – nad každým sloupcem na okraji mřížky se vyskytuje záznam v číselné podobě nebo zde není uveden žádný záznam, obdobně jako u záznamů pro řádky. Tyto záznamy udávají výslednou podobu zakódovaného obrázku. Přesný význam těchto hodnot bude vysvětlen v podkapitole 2.2 Pravidla řešení.
- **Zadání** – nachází se vedle řádku nebo nad sloupcem. Značí, že v daném řádku nebo v sloupci bude skupina čtverců jedné barvy o délce daného čísla nacházejícího se v zadání. Barva záznamu odpovídá barvě popředí buňky. Název "blok" nebo "skupina buněk" je jiný název pro "skupinu čtverců" a tyto pojmy jsou rovnocenné.

- **Pozadí** – nachází se vedle zadání či přímo vedle řádku nebo nad sloupcem a nese spíše orientační význam. Barva pozadí u zadání řádku nebo sloupce odpovídá barvě pozadí buňky.

Při luštění vícebarevné verze této hry může buňka nabývat všech barev, které jsou zadány. Ve verzi hry, která v zadání obsahuje trojúhelníky, mohou být buňky vybarveny příslušnou barvou z jedné poloviny. Tuto polovinu udává úhlopříčka buňky, která tak rozdělí buňku na dva trojúhelníky.



Obrázek 2.3: Popis základních částí hry Griddlers.

2.2 Pravidla řešení

Pravidla [6] řešení všech černobílých obrázků spočívají v několika následujících odstavcích. Pro úspěšné vyřešení obrázku je potřeba dodržet všechna pravidla, aby v něm nevznikly případné chyby. Všechna tato pravidla platí pro veškeré řádky a sloupce současně.

Pravidla řešení hry Griddlers:

- **První pravidlo** – všechna čísla v zadání řádku nebo sloupci odpovídají souvislé skupině buněk s jednou barvou, která odpovídá barvě popředí. Na obrázku 2.4 je znázorněn řádek, který má v zadání číslo 5. V mřížce tomu musí odpovídat skupina buněk obarvená barvou popředí. Tato skupina buněk tvoří jeden blok.

Obrázek 2.4: První pravidlo vyobrazující skupinu pěti buněk vedle sebe.

- **Druhé pravidlo** – je-li vedle řádku nebo nad sloupcem více číselných zadání, každé z těchto čísel odpovídá souvislé skupině buněk (viz první pravidlo). Mezi každou skupinou buněk se musí nalézat minimálně jedna prázdná buňka, která odděluje skupiny buněk. Tato oddělující buňka je tzv. prázdná a má barvu pozadí obrázku. Obrázek 2.5 má v zadání řádku čísla 3 a 1. V mřížce tomu musí odpovídat skupina tří buněk a druhá skupina o velikosti pouze jedné buňky. Mezi těmito dvěma skupinami se musí nacházet minimálně jedna buňka s barvou pozadí.

Prázdná buňka

Obrázek 2.5: Druhé pravidlo vyobrazující prázdnou buňku oddělující dvě skupiny buněk.

- **Třetí pravidlo** – čísla udávaná vedle řádku nebo nad sloupcem odpovídají blokům v mřížce. Např.: první číslo zleva vedle řádku odpovídá prvnímu bloku v mřížce zleva, druhé číslo zleva vedle řádku odpovídá druhému bloku v mřížce zleva, atd. Na obrázku 2.6 je znázorněno pravidlo, které je založeno na pořadí skupin buněk a tomu odpovídajících číselných zadání. První zleva je číslo 3 a to odpovídá zleva první skupině buněk o délce 3. Druhé zleva je číslo 1 a to odpovídá zleva druhé skupině buněk pouze o délce 1.

Pořadí skupin buněk

Obrázek 2.6: Třetí pravidlo vyobrazující pořadí jednotlivých skupin buněk.

- **Čtvrté pravidlo** – buňky, do kterých nebude zasahovat žádný blok popředí, budou obarveny barvou pozadí. V zadání jsou uvedeny pouze buňky popředí a ty nemusí vyplnit celý prostor, proto jsou ostatní buňky vyplněny barvou pozadí. Obrázek 2.7 znázorňuje umístění buňky s barvou popředí a na ostatní místa jsou umístěny buňky s barvou pozadí. Černá buňka je umístěna na prostřední pozici v této mřížce a předpokládejme, že v kontextu celého zadání odpovídá správnému řešení. Tento příklad je pouze ilustrativní. V případě, že by byla mřížka takto velká, budou nad sloupci mřížky číselná zadáná udávající umístění černé buňky.

Doplnění prázdných buněk

Obrázek 2.7: Čtvrté pravidlo vyobrazující doplnění prázdných buněk do okolí bloku.

2.3 Způsoby řešení

Při řešení jakéhokoliv zadání této hry je potřeba postupně určovat barvu jednotlivých buněk v mřížce. Určení barvy pozadí je stejně důležité, jako určení barvy popředí. Postupným řešením [2] celé mřížky nám budou už obarvené buňky barvou popředí nebo pozadí napovídat správné rozložení jednotlivých skupin čtverců. Pro přehlednost při luštění "na papíře" je dobré si zvyknout na označování vyřešených buněk jednotným značením – např.: označení buňky pozadí místo bílé barvy křížkem a označení buňky popředí vybarvením celé této buňky.

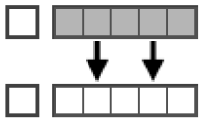
V postupu řešení je důležité se zaměřit pouze na obarvení takových buněk, u kterých logicky víme, že jsou obarveny správně. Není dobré při řešení tipovat, protože ve výsledném obrázku při neúspěšném tipu vznikne chyba. Tento jediný tip může ve výsledku znehodnotit celý obrázek a zpětné dohledání chyby je velmi náročné či skoro nemožné.

Zakódovaný obrázek nemá žádný vliv na postup luštění. Může se však stát, že z obrázku bude patrné správné obarvení nějaké buňky, ale není vhodné se na tento postup luštění spoléhat, protože obrázek může vypadat trochu odlišně, než jsme o něm měli svoji představu.

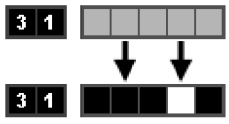
V následujících odrážkách jsou uvedeny nejběžnější způsoby řešení [7] hry Griddlers. Všechny tyto způsoby jsou aplikovatelné nejenom na řádky, ale i na sloupce. Některé způsoby řešení jsou zaměřeny na doplňování barvy popředí obrázku, barvy pozadí obrázku nebo doplnění barvy popředí a pozadí obrázku zároveň. Můžeme říci, že sloupec je o 90° pootočen (ve směru hodinových ručiček) od řádku. Při aplikování těchto způsobů řešení se musí dodržet všechna pravidla zmíněná v podkapitole 2.2 Pravidla řešení.

Nejčastěji používané způsoby řešení:

- **Přímé doplnění** – jedna z nejjednodušších metod řešení spočívá v přímém vyplnění buněk podle údajů před řádkem. Jestliže se před řešeným řádkem neobjevuje žádné číslo, bude celý řádek vyplněn barvou pozadí, viz obrázek 2.8. Jsou-li před řádkem nějaká čísla je potřeba spočítat čísla na okraji řádku a mezi jednotlivá čísla přičíst jednu volnou buňku oddělující tyto čísla. Po spočítání těchto čísel a porovnáním s délkou řádku v řešené mřížce je možné doplnit tyto číselné údaje do řádku pouze v případě, že délka řádku (např.: délka = 5) odpovídá součtu čísel ze zadání (např.: bloky o délce 3 a 1 a mezi nimi 1 volná buňka). Příklad doplnění takového řádku znázorňuje obrázek 2.9.

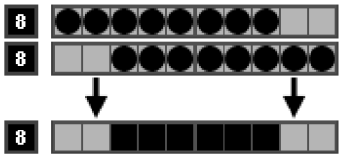


Obrázek 2.8: Vyplnění všech buněk v řádku barvou pozadí.

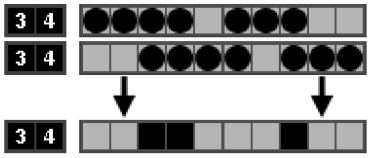


Obrázek 2.9: Vyplnění řádku skupinami bloků s barvou popředí a buňkou oddělující bloky.

- **Jednoduše obsaditelné bloky** – metoda spočívá v doplňování skupin čtverců do daného řádku. Pro uplatnění této metody musí být v zadání řádku součet čísel větší než polovina součtu nevyřešených buněk v řádku. Postup metody spočívá v umístění skupin čtverců do pomocného řádku od jeho začátku a do druhého pomocného řádku od jeho konce a vzájemného překrytí těchto pomocných řádků. Při doplňování skupin čtverců za sebe se mezi jednotlivé skupiny vloží mezera o délce jedné buňky, která oddělí tyto skupiny. Po doplnění skupin čtverců do pomocných řádků, se provede průnik těchto řádků. Kde se budou stejné bloky překrývat, bude ve výsledném řádku obarvena tato buňka barvou popředí. Názorný příklad použití této metody pro jeden číselný zaznam (např.: blok o délce 8) je na obrázku 2.10 a příklad se dvěma číselnými záznamy (např.: bloky o délce 3 a 4) je zobrazen na obrázku 2.11.

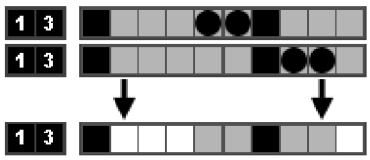


Obrázek 2.10: Doplnění skupiny buněk v barvě popředí pomocí jednoduše obsaditelného bloku o délce 8 buněk.



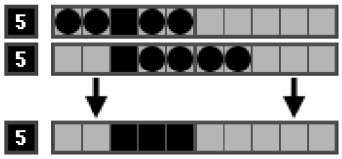
Obrázek 2.11: Doplnění skupin buněk v barvě popředí pomocí jednoduše obsaditelných bloků o délce 3 a 4 buňky.

- **Jednoduché volné bloky** – metoda má za úkol doplnit barvu pozadí na ta místa, do kterých nebude dosahovat žádná ze skupin čtverců. Při použití této metody se vychází z obarvených buněk barvou popředí, které jsou již zaznačeny v tomto řádku. Uplatnění této metody je znázorněno na obrázku 2.12. Do jednoho pomocného řádku se zarovnají vlevo zatím nezaznačené části bloků. Podobně se to provede v druhém pomocném řádku, kde se zarovnají nezaznačené bloky vpravo. Následně se provede sjednocení těchto pomocných řádků a do míst, kde nezasahuje žádná z částí bloků jsou umístěny buňky s barvou pozadí. Použití této metody není vždy správné, protože obarvené buňky v řádku mohou být ze stejné skupiny čtverců odděleny zatím nevyřešenou buňkou.

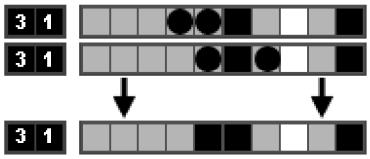


Obrázek 2.12: Doplnění volných buněk na místa, ve kterých se nemohou vyskytnou buňky s barvou popředí.

- **Sounáležitost obsaditelných bloků** – metoda je použitelná v řádcích, kde se nachází buňka vyplněná barvou popředí v blízkosti okraje řádku nebo v blízkosti buňky vyplněné barvou pozadí. Buňka s barvou popředí se může rozšířit i na okolní buňky v řádku podle číselného zadání na okraji řádku, protože máme jistotu, že v těchto nově obarvených buňkách se bude nacházet tato skupina čtverců nebo její část. Pro názornost zachycuje obrázek 2.13 použití této metody pro jeden číselný záznam (např:. blok délky 5 buněk) blízko okraje a další obrázek 2.14 znázorňuje použití této metody pro dva číselné záznamy (např.: délky bloků 1 a 3 buňky) v okolí buňky s barvou pozadí. Tato metoda je podobná metodě s jednoduše obsaditelnými bloky.



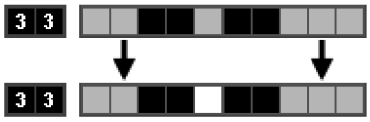
Obrázek 2.13: Doplnění sounáležitých buněk s barvou popředí na místo, ve kterém se bude nalézat skupina buněk.



Obrázek 2.14: Doplnění sounáležitých buněk s barvou popředí na místa, ve kterých se budou nalézat skupiny buněk.

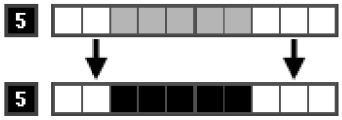
- **Spojování a rozdělování bloků popředí a pozadí** – jedna z náročnějších metod, která po správném doplnění buněk usnadní další luštění. Spočívá ve spojení skupin čtverců s barvou popředí do jedné delší skupiny nebo, v případě rozdělování bloků, vložením mezi bloky s barvou popředí buňky s barvou pozadí a tím se bloky oddělí. Metoda vychází z již doplněných buněk v řádku. Spojení buněk znázorňuje obrázek 2.15 a rozdělení buněk znázorňuje obrázek 2.16. Veškeré spojování a rozdělování bloků spočívá v znalosti číselného zadání řádku a správné interpretaci na příslušné buňky v řádku. Tato metoda je jednou ze zásadnějších v řešení, protože při nesprávném uplatnění může dojít k nevratnému znehodnocení řešeného obrázku.

Obrázek 2.15: Spojení dvou bloků buněk do jednoho bloku odpovídajícímu zadaní pro tento řádek.

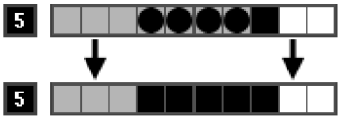


Obrázek 2.16: Rozdělení dvou bloků buněk vložením volné buňky s barvou pozadí.

Všechny tyto základní metody lze vzájemně kombinovat a používat pro řádky, sloupce a u složitějších případů řešení jsou použitelné pro kombinaci řádků a sloupců dohromady. Jednotlivé metody mohou být uplatněny i na část řádku nebo sloupce. Známe-li obsah některých buněk, které umožňují uplatnit minimálně jednu z metod řešení na část daného řádku nebo sloupce, je vhodné tuto metodu použít. Příklad uplatnění metody přímého doplnění na část řádku je znázorněn na obrázku 2.17. Obrázek 2.18 znározňuje uplatnění metody sounáležitosti obsaditelného bloku.



Obrázek 2.17: Vyplnění zbývajících buněk v řádku barvou popředí.



Obrázek 2.18: Doplnění sounáležitých buněk s barvou popředí na místa, ve kterých se bude nalézat skupina buněk.

V této kapitole jsme se seznámili s hrou Griddlers. Osvětlili jsme si její pravidla řešení a základní způsoby řešení této hry. V příloze C na konci práce jsou vloženy další neřešené příklady této hry i se správnými výsledky (pro případné zájemce). Příloha D obsahuje další varianty hry Griddlers. Tyto varianty hry jsou vyobrazeny pouze jako řešené obrázky se zadáním.

Kapitola 3

Návrh řešitele hry Griddlers

Předchozí kapitola vysvětlila hru Griddlers a s ní spojená pravidla i základní způsoby řešení. Se všemi těmito znalostmi přistoupíme k samotnému návrhu řešitele. Tato kapitola se bude zabývat návrhem řešitele hry Griddlers a věcí s tím spojených – načítání zadání ze souboru, uložení nalezeného řešení do souboru, kontrola načtených dat a návrh struktury aplikace.

3.1 Uložení dat

Před jakoukoliv prací s aplikací řešitele je potřeba správně navrhnout strukturu uložení dat. Pro uložení všech dat je vybrán XML soubor [3]. Každý XML soubor bude obsahovat jedno zadání hry Griddlers nebo správné řešení jednoho zadání nalezeného pomocí řešitele.

Při spuštění aplikace se předávají mezi vstupními parametry programu názvy XML souborů obsahující zadání hry Griddlers a parametry specifikující činnost programu. Výstup programu uloží do XML souboru výsledek řešení hry Griddlers. Pro každý vstupní XML soubor bude vytvořen nový XML soubor nebo bude přepsán XML soubor s řešením. Každý výstupní XML soubor bude obsahovat nalezené řešení a zadání tohoto řešení.

Specifikace DTD pro popis XML struktury obsahující vstupní data aplikace:

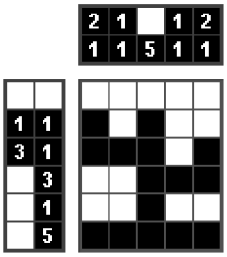
```
<!ELEMENT matice (sloupec*, radek*)>
<!ATTLIST matice
          nazev CDATA #REQUIRED
          pocetradku CDATA #REQUIRED
          pocetsloupcu CDATA #REQUIRED>
<!ELEMENT sloupec (#PCDATA)>
<!ATTLIST sloupec
          poradi CDATA #REQUIRED>
<!ELEMENT radek (#PCDATA)>
<!ATTLIST radek
          poradi CDATA #REQUIRED>
```

Struktura vstupního XML souboru má hlavní element s názvem `matice`. Hlavní element obsahuje ještě atributy s názvem zakódovaného obrázku `nazev`, s počtem řádků `pocetradku` a s počtem sloupců `pocetsloupcu`. Mezi počátečním a koncovým elementem `matice` se nachází dva druhy elementů odpovídající zadání řádků `radek` a sloupců `sloupec` zakódovaného obrázku. Tyto elementy mají atribut pořadí `poradi`, který indexuje od nulté pozice umístění daného řádku nebo sloupce od levého horního rohu mřížky. Element `radek`

a `sloupec` obsahuje samotné číselné zadání pro jednotlivé řádky a sloupce. V případě více čísel pro jeden řádek, jsou tato čísla oddělena čárkou.

Příklad vstupních dat obsahující zakódovaný obrázek 3.1:

```
<matice nazev="kaktus" pocetradku="6" pocetsloupcu="5">
    <sloupec poradi="0">2,1</sloupec>
    <sloupec poradi="1">1,1</sloupec>
    <sloupec poradi="2">5</sloupec>
    <sloupec poradi="3">1,1</sloupec>
    <sloupec poradi="4">2,1</sloupec>
    <radek poradi="0"></radek>
    <radek poradi="1">1,1</radek>
    <radek poradi="2">3,1</radek>
    <radek poradi="3">3</radek>
    <radek poradi="4">1</radek>
    <radek poradi="5">5</radek>
</matice>
```



Obrázek 3.1: Příklad řešení zakódovaného obrázku.

Specifikace DTD pro popis XML struktury obsahující výstupní data aplikace:

```
<!ELEMENT matice (sloupec*, radek*, reseni)>
<!ATTLIST matice
          nazev CDATA #REQUIRED
          pocetradku CDATA #REQUIRED
          pocetsloupcu CDATA #REQUIRED>
<!ELEMENT sloupec (#PCDATA)>
<!ATTLIST sloupec
          poradi CDATA #REQUIRED>
<!ELEMENT radek (#PCDATA)>
<!ATTLIST radek
          poradi CDATA #REQUIRED>
<!ELEMENT reseni (radekreseni*)>
<!ELEMENT radekreseni (#PCDATA)>
<!ATTLIST radekreseni
          poradi CDATA #REQUIRED>
```

Struktura výstupního XML souboru je podobná struktuře vstupního souboru. Navíc ve struktuře výstupního XML souboru je element `reseni`, který je vnořen do hlavního elementu `matice`. Element `reseni` obsahuje element `radekreseni`, který zachycuje vyřešený zakódovaný obrázek. Každý jeden element `radekreseni` nese údaj o jednom vyřešeném

řádku z mřížky. Pro přehlednost element `radekreseni` má atribut `poradi`, který indexuje od horní nulté pozice vyřešené řádky.

Příklad výstupních dat obsahující vyřešený zakódovaný obrázek 3.1:

```
<matice nazev="kaktus" pocetradku="6" pocetsloupcu="5">
    <sloupec poradi="0">2,1</sloupec>
    <sloupec poradi="1">1,1</sloupec>
    <sloupec poradi="2">5</sloupec>
    <sloupec poradi="3">1,1</sloupec>
    <sloupec poradi="4">2,1</sloupec>
    <radek poradi="0"></radek>
    <radek poradi="1">1,1</radek>
    <radek poradi="2">3,1</radek>
    <radek poradi="3">3</radek>
    <radek poradi="4">1</radek>
    <radek poradi="5">5</radek>
    <reseni>
        <radekreseni poradi="0">OOOOO</radekreseni>
        <radekreseni poradi="1">XOXOO</radekreseni>
        <radekreseni poradi="2">XXXOX</radekreseni>
        <radekreseni poradi="3">OOXXX</radekreseni>
        <radekreseni poradi="4">OOXOO</radekreseni>
        <radekreseni poradi="5">XXXXX</radekreseni>
    </reseni>
</matice>
```

Výstupní XML soubor obsahuje jednotlivé řádky mřížky s nalezeným řešením. Jednotlivé buňky mřížky jsou reprezentovány znakem `X`, který zastupuje buňku obarvenou barvou popředí, a nebo znakem `O`, který zastupuje buňku obarvenou barvou pozadí obrázku.

3.2 Kontrola dat

V předchozí podkapitole 3.1 jsme si navrhli a popsali strukturu XML souboru obsahujícího zadání kódovaného obrázku. Už při načítání dat z XML souboru můžeme provést základní kontrolu správnosti uložených dat.

Základní kontrola vstupních dat:

- **Rozměr mřížky** – pro daný rozměr mřížky, který je udán v atributech `pocetradku` a `pocetsloupcu` hlavního elementu `matice`, musí být odpovídající počet záznamů pro řádky a pro sloupce. V případě, že se bude nějaký záznam pro řádek nebo sloupec opakovat, bude uživatel upozorněn na vzniklou chybu a XML soubor nebude dále načítán. Jestliže budou v XML souboru chybět nějaké záznamy pro řádky nebo sloupce, budou tato chybějící data nahrazena záznamem, který bude obsahovat pouze informaci o pozadí bez číselného záznamu.
- **Správné číselné údaje** – v záznamu vedle každého řádku a nad každým sloupcem celé mřížky musí být správné číselné údaje. Čísla v zadání musí být v daném rozmezí. Číslo c musí být větší nebo rovno 0 a zároveň menší nebo rovno délce d řádku nebo sloupce ($c \geq 0 \wedge c \leq d$). V opačném případě zadání obsahuje chybu. Příklad kontroly pomocí tohoto bodu si můžeme ukázat na obrázku 3.2, který obsahuje chybně zadaný

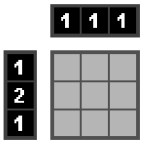
řádek. Číslo c v zadání řádku je 8 ($c = 8$). Délka d řádku je rovna 5 ($d = 5$). Uplatněním podmínky na číslo c nám vznikne chyba. Po dosazení do logického výrazu $c \geq 0 \wedge c \leq d$ zjistíme, že zadané číslo c není ve správném rozmezí (chybný výraz: $8 \geq 0 \wedge 8 \leq 5$).

Obrázek 3.2: Chyba čísleného zadání pro řádek.

- **Součet čísel záznamu** – součet čísel záznamu vedle každého řádku a nad každým sloupcem celé mřížky musí být ve správném rozmezí. Součet čísel záznamu z je tvořen sečtením číselných udajů před řádkem nebo nad sloupcem. Při sčítání více čísel v řádku nebo sloupci je vždy potřeba přičíst mezi dvěma čísly hodnotu jedna (odpovídající velikosti prázdné buňky oddělující jednotlivé skupiny buněk). Součet čísel záznamu z musí být větší nebo roven 0 a menší nebo roven délce d řádku nebo sloupce ($z \geq 0 \wedge z \leq d$) v opačném případě je v zadání chyba. Příklad kontroly pomocí tohoto bodu si můžeme ukázat na obrázku 3.3, který obsahuje chybně zadaný řádek. Zadání řádku obsahuje čísla 3 a 4. Při sčítání nesmíme zapomenout přičíst prázdný blok oddělující jednotlivá čísla. Součtem čísel a mezery dojdeme k hodnotě 8 (blok s délkou 3 + mezera s délkou 1 + blok s délkou 4 = 8). Součet z je roven 8 ($z = 8$). Délka d řádku je rovna 5 ($d = 5$). Uplatněním podmínky na součet z nám vznikne chyba. Po dosazení do logického výrazu $z \geq 0 \wedge z \leq d$ zjistíme, že součet z není ve správném rozmezí (chybný výraz: $8 \geq 0 \wedge 8 \leq 5$).

Obrázek 3.3: Chyba součtu číselného zadání pro řádek.

- **Součet všech čísel** – poslední ze základních kontrol spočívá v sečtení všech číselných zadání pro řádky a sečtení všech číselných zadání pro sloupce. Součet čísel řádků r musí být roven součtu čísel sloupců s ($r = s$) v opačném případě je v zadání chyba. Příklad kontroly pomocí tohoto bodu si můžeme ukázat na obrázku 3.4, který obsahuje chybné zadání. Součtem všech čísel v řádcích dojdeme k hodnotě 4 ($1 + 2 + 1 = 4$) Součet čísel z řádků r je roven 4 ($r = 4$). Součtem všech čísel ve sloupcích dojdeme k hodnotě 3 ($1 + 1 + 1 = 3$) Součet čísel ze sloupců s je roven 3 ($s = 3$). Uplatněním podmínky na řádky r a sloupce s nám vznikne chyba. Po dosazení do logického výrazu $r = s$ zjistíme, že součet řádků r a sloupců s si není roven (chybný výraz: $4 = 3$).



Obrázek 3.4: Chyba součtu číselných zadání pro řádky a sloupce.

Základní kontrola dat slouží k včasnému odhalení chyby zadání a vyvarování se problémům při řešení zakódovaného obrázku.

3.3 Řešení hry

Řešitel hry Griddlers je navržen jako konzolová aplikace. Vstup dat pro aplikaci je uložen v XML souboru a výsledný výstup dat bude uložen do nového XML souboru. Po spuštění aplikace s příslušnými parametry a soubory začne řešitel provádět svoji činnost. Nejprve budou zpracovány vstupní parametry a soubory. Zpracování XML souborů spočívá v načtení obsahu zakódovaného obrázku a uložení všech načtených dat do třídy, která reprezentuje jedno zadání. Všechny třídy obsahující zadání jsou předány ve frontě hlavní části aplikace, která se stará o samotné luštění daných obrázků.

Hlavní jádro aplikace, starající se o luštění, si načte jedno zadání zakódovaného obrázku z fronty a následně volá implementované algoritmy pro vyřešení tohoto zadání. Při luštění zakódovaného obrázku se uplatní pouze ty algoritmy, které byly specifikovány v parametru při spuštění programu. V případě, že nebyl specifikován žádný algoritmus, budou automaticky použity všechny dostupné algoritmy pro řešení. Použitý algoritmus vrátí výsledek řešení, spotřebovaný uživatelský čas, spotřebovaný reálný čas a spotřebovanou paměť.

Jednotlivé algoritmy se budou lišit svojí rychlostí a paměťovou náročností. Tento rozdíl je zapříčiněn samotnou činností těchto algoritmů. Jednotlivé uvažované algoritmy, jejich výhody a nevýhody spolu s kvalitou časové a prostorové náročnosti, jsou zmíněny v následující kapitole 4 o algoritmech.

Hlavní jádro programu pracující s algoritmy pro řešení zakódovaného obrázku je jednoduše rozšiřitelné. Rozšíření spočívá v přidání dalších algoritmů, nebo v úpravě použitých algoritmů. Po dokončení luštění je možné výsledky zobrazit v konzolovém okně a v případě potřeby uložit výstup programu do XML souboru.

V následující podkapitole 3.4 jsou shrnuty základní vlastnosti navrhovaného řešitele hry Griddlers.

3.4 Vlastnosti řešitele

Řešitel hry Griddlers bude spouštěn pomocí příkazů zadaných do konzole v příslušném umístění s tímto programem.

Běh programu může být ovlivněn zadanými přepínači. Na vstupu programu společně s XML soubory může být zadán přepínač pro vypsání základní nápovědy k programu, volbu algoritmů použitých pro řešení, zobrazení a uložení vyřešeného zadání.

Načítání zadání kódovaného obrázku a ukládání vyřešeného kódovaného obrázku bude probíhat pomocí XML souborů. Zadání, které požadujeme vyřešit, uvedeme mezi vstupní parametry programu. Vyřešené zadání je uloženo do nového XML souboru, který se nachází ve stejném umístění jako původní soubor. V případě neúspěšného uložení vyřešených dat do nového souboru bude proveden pokus o přepsání vstupního souboru. Jestliže nebude možné přepsat ani vstupní soubor, bude na tuto skutečnost uživatel upozorněn. Po uložení do nového nebo vstupního souboru bude uživateli sděleno, kde se nachází výsledná data.

Při řešení zadání budou postupně vypisovány do konzole údaje o právě použitém algoritmu, názvu řešeného zadání, výsledku řešení, spotřebovaném čase, počtu vytvořených tříd potřebných k nalezení výsledku a maximální velikosti fronty (respektive zásobníku

použitého při řešení). Hledání výsledku bude možné pomocí různých algoritmů. Celá následující kapitola 4 je věnována algoritmům a možnosti jejich uplatnění v řešiteli.

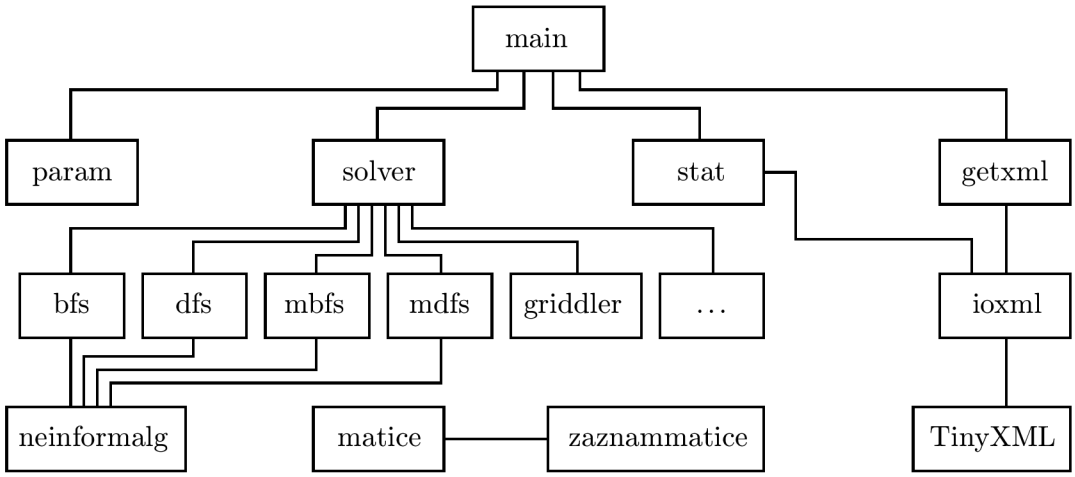
Popis implementace a jednotlivých implementovaných vlastností se nachází v kapitole 5 a v programátorské příručce.

3.5 Diagram tříd

Na obrázku 3.5 je znázorněn navrhovaný diagram tříť řešitele hry Griddlers. Celý řešitel bude ovládán hlavní funkcí `main`, která při spuštění zavolá třídu `param` za učelem zpracování parametrů programu. Následně je voláná třída `getxml` pro načtení XML souborů. Třída `ioxml` slouží pro načítání a ukládání XML souborů. Tato třída spolupracuje s XML parsrem TinyXML. Po načtení všech zadání ze souborů následuje hlavní činnost řešitele.

Třída `solver` se stará o řešení všech zadání pomocí zvolených algoritmů, které jsou postupně volány z této třídy. Třídy `bfs, dfs, mbfs, mdfs a griddler` obsahují jednotlivé algoritmy pro hledání řešení. Třída s názvem `...` značí možnost přidání do tohoto místa diagramu tříd další algoritmy vytvořené v nových třídách pro řešení zakódovaných obrázků. Třída `neinfomtalg` obsahuje metody použitelné u více algoritmů, které je možné zdědit.

Ve spodní části diagramu tříd jsou znázorněny třídy `matice` a `zaznammatice`. Třída `matice` slouží k uložení celého zadání a třída `zaznammatice` slouží k uložení jednoho řádku nebo sloupce zadání. Tyto dvě třídy slouží pouze k uchovávání dat v řešiteli.



Obrázek 3.5: Návrh diagramu tříd.

V této kapitole jsme se seznámili se základním návrhem celého řešitele hry Griddlers, jeho vlastnostmi a popsali jsme si struktury používaných XML souborů. Následující kapitola 4 pojednává o jednotlivých použitých algoritmech a jejich vlastnostech. V Kapitole 5 Implementace řešitele je popsána výsledná implementace programu a jeho vlastnosti.

Kapitola 4

Používané algoritmy

Již jsme se seznámili s hrou Griddlers a návrhem řešitele hry Griddlers. V této kapitole se zaměříme na algoritmy, které budou použity pro řešení zakódovaného obrázku. Specifikujeme si jejich základní činnost, výhody a nevýhody, časovou a paměťovou náročnost a vzájemně porovnáme jejich kvalitu. Pro řešení byly vybrány algoritmy prohledávání do šířky, prohledávání do hloubky a algoritmus založený na principu zpětného prohledávání.

Na začátku této kapitoly je potřeba, abychom si definovali základní termíny, které se váží s problematikou algoritmů a jejich řešení, protože nadefinované termíny budou dále používány při popisu algoritmů.

4.1 Základní pojmy

Algoritmy zabývající se prohledáváním stavového prostou (mřížka buněk) můžeme rozdělit do dvou skupin. První skupina obsahuje neinformované metody algoritmů a druhá obsahuje informované metody algoritmy. Dále můžem dělit algoritmy podle hodnotících kritérií.

V následujících odstavcích jsou shrnuty základní vlastnosti algoritmů [9].

Neinformované metody nepoužívají k prohledávání stavového prostoru žádné informace o cílovém stavu ani nepoužívají žádné prostředky pro ohodnocení aktuálního stavu.

Informované metody používají při prohledávání stavového prostoru potřebné informace o cílovém stavu a využívají prostředky k ohodnocení aktuálního stavu prohledávání.

Hodnotící kritéria se uplatňují na algoritmy a jejich vlastnosti. Kritéria pro hodnocení jsou:

- Úplnost – značí, jestli algoritmus nalezne řešení, pokud existuje
- Optimálnost – značí, jestli algoritmus nalezne nejlepší řešení
- Časová náročnost – udává pomocí matematického vztahu časovou náročnost daného algoritmu
- Paměťová náročnost – udává pomocí matematického vztahu paměťovou náročnost daného algoritmu

Matematické zkratky použité u časové a paměťové náročnosti:

O – počet operátorů daného algoritmu
b – faktor větvení, neboli průměrný počet bezprostředních následníků každého uzlu řešení
d – hloubka nejlepšího řešení, neboli řešení, které se nachází v nejmenší hloubce
m – maximální prohledávaná hloubka (šířka mřížky)
n – maximální prohledávaná hloubka (výška mřížky)

4.2 Neinformované algoritmy

Při řešení hry Griddlers je možné použít pro luštění, z řad neinformovaných algoritmů [9], slepé prohledávaní do šířky (viz podkapitola 4.2.1) a slepé prohledávaní do hloubky (viz podkapitola 4.2.2).

4.2.1 Slepé prohledávání do šířky (BFS)

Popis algoritmu: algoritmus slepého prohledávání do šířky (BFS Breadth–first search) [1] patří mezi základní algoritmy prohledávání stavového prostoru. Prohledávání stavového prostoru pomocí BFS je úplné a optimální v případě, že je počet bezprostředních následníků každého uzlu konečný. Časová a paměťová náročnost algoritmu BFS je exponenciální $O * (b^{d+1})$. Exponenciální náročnost je zapříčiněna konstrukcí tohoto algoritmu. BFS patří do skupiny neinformovaných algoritmů, protože prohledává celý stavový prostor mezi počátečním a cílovým stavem a při hledání nepoužívá žádnou heuristiku.

BFS není použitelný pro řešení složitých úloh [9] díky své exponenciální časové a paměťové náročnosti. I přes tento nedostatek je BFS používán pro svoji jednoduchost. Při používání na jednoduchých úlohách se objevuje závažný problém, který je zapříčiněn opakovaným generování již expandovaných uzlů.

Algoritmus slepého prohledávání do šířky:

1. Sestroj frontu OPEN, která bude obsahovat všechny uzly určené pro expanzi, a umísti do fronty OPEN počáteční uzel.
2. Jestli je fronta OPEN prázdná, potom úloha nemá řešení a ukonči prohledávání jako neúspěšné. Jestli fronta není prázdná, pokračuj dál.
3. Vyber první uzel z čela fronty OPEN.
4. Jestli je vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a navrať cestu od počátečního kořenového uzlu k cílovému uzlu. Jestli vybraný uzel není uzlem cílovým, pokračuj dál.
5. Vybraný uzel expanduj, všechny jeho bezprostřední následovníky umísti do fronty OPEN a pokračuj dál od bodu 2.

Pseudokód: v následujícím pseudokódu [4] je znázorněn princip algoritmu slepého prohledávání do šířky.

```
1.  bool BFS (prvni_uzel){
2.    Sestroj fronta_OPEN;
3.    Vlož prvni_uzel do fronta_OPEN;
4.    while (fronta_OPEN != prázdná){
5.      uzel = Vyber z fronta_OPEN;
6.      if(uzel == cíl){
7.        return true;
8.      }
9.      else{
10.       Expanduj uzel a vlož do fronta_OPEN;
11.     }
12.   }
13.   return false;
14. }
```

Vlastnosti algoritmu slepého prohledávání do šířky:

Úplnost – ANO
Optimálnost – ANO
Časová náročnost – exponenciální $O * (b^{d+1})$
Paměťová náročnost – exponenciální $O * (b^{d+1})$

4.2.2 Slepé prohledávání do hloubky (DFS)

Popis algoritmu: algoritmus slepého prohledávání do hloubky (DFS Depth–first search) [1] patří mezi jednoduché algoritmy prohledávání stavového prostoru. Prohledávání stavového prostoru pomocí DFS není, na rozdíl od BFS, úplné a ani optimální. Časová náročnost algoritmu DFS je exponenciální $O * (b^m)$. Paměťová náročnost je však lineární $O * (b * m)$. DFS patří do skupiny neinformovaných algoritmů, stejně jako BFS. Při prohledávání stavového prostoru mezi počátečním a cílovým stavem, může DFS snadno selhat. Tento jev je zapříčiněn opakovaným generováním stejných uzlů, které již byly generovány.

Algoritmus slepého prohledávání do hloubky:

1. Sestroj zásobník OPEN, který bude obsahovat všechny uzly určené pro expanzi, a umísti do zásobníku OPEN počáteční uzel.
2. Jestli je zásobník OPEN prázdný, potom úloha nemá řešení a ukonči prohledávání jako neúspěšné. Jestli zásobník OPEN není prázdný, pokračuj dál.
3. Vyber první uzel z vrcholu zásobníku OPEN.
4. Jestli je vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a navrať cestu od počátečního kořenového uzlu k cílovému uzlu. Jestli vybraný uzel není uzlem cílovým, pokračuj dál.
5. Vybraný uzel expanduj, všechny jeho bezprostřední následovníky umísti do zásobníku OPEN a pokračuj dál od bodu 2.

Pseudokód: v následujícím pseudokódu [4] je znázorněn princip algoritmu slepého prohledávání do hloubky.

```
1.  bool DFS (prvni_uzel){
2.    Sestroj zasobnik_OPEN;
3.    Vlož prvni_uzel do zasobnik_OPEN;
4.    while (zasobnik_OPEN != prázdný){
5.      uzel = Vyber z zasobnik_OPEN;
6.      if(uzel == cíl){
7.        return true;
8.      }
9.      else{
10.       Expanduj uzel a vlož do zasobnik_OPEN;
11.     }
12.   }
13.   return false;
14. }
```

Vlastnosti algoritmu slepého prohledávání do hloubky:

Úplnost – NE
Optimálnost – NE
Časová náročnost – exponenciální $O * (b^m)$
Paměťová náročnost – lineární $O * (b * m)$

4.2.3 Aplikace neinformovaných algoritmů na hru Griddlers

Algoritmy BFS a DFS pro luštění zakódovaného obrázku používají svůj základní algoritmus.

Za počáteční stav těchto algoritmů se považuje prázdná mřížka. Cílový stav odpovídá mřížce, která je správně vyluštěná. Kontrola cílového stavu se provede otestováním každého řádku a sloupce. Testování spočívá v kontrole správného rozložení a velikosti skupin buněk v řádcích a sloupcích.

Jednotlivé uzly algoritmu BFS, DFS odpovídají sloupcům mřížky. Při expandování uzlu se rozgenerují pro daný sloupec všechny možné varianty umístění skupin bloků.

Hloubka prohledávání stavového prostoru algoritmů BFS a DFS je omezena na rozměr mřížky kódovaného obrázku. Hloubka prohledávání stavového prostoru se řídí počtem sloupců mřížky a pro každé zadání kódovaného obrázku může být jiná.

Operátor expandování je použitelný pro každý uzel zadání a je pouze jeden. Tento jediný operátor postupně posouvá všechny skupiny buněk v uzlu tak, aby při kompletním rozgenerování byly skupiny buněk rozmístěny ve všech možných a především správných kombinacích pro daný uzel.

4.3 Informovaný algoritmus

Pro úspěšné řešení hry Griddlers pomocí informovaného algoritmu navrhuji použít specifický algoritmus vycházející přímo z hry Griddlers, která je popsána v kapitole 2. Tento specifický informovaný algoritmus můžeme pojmenovat názvem: algoritmus griddler.

4.3.1 Algoritmus griddler

Popis algoritmu: algoritmus griddler patří mezi informované algoritmy. Při prohledávání stavového prostoru a ve snaze nalézt správné řešení zakódovaného obrázku vychází algoritmus z jednotlivých buněk v mřížce, záznamů vedle řádků a nad sloupci. Buňky mohou mít při řešení barvu popředí, pozadí nebo ještě nebyly vyplněny barvou.

Prohledávání a řešení stavového prostoru není úplné a ani optimální, protože při aplikování špatného pořadí operátorů na daný řádek nebo sloupec může dojít k drobné změně výsledného řešení a zakódovaný obrázek se už nemusí podařit vyřešit nebo jeho řešení nebude správné. Aplikace operátorů je přizpůsobena tak, aby zásadnější operátory řešení, které mohou zapříčinit chybu, byly aplikovány později než operátory, které chybu nezapříčiní. Časová náročnost algoritmu griddler je $\sum_{i=1}^{x} O_i * (m * n)$. Časová náročnost vychází ze součtu všech dob $\sum_{i=1}^{x}$ při aplikování operátorů O na jednotlivé sloupce m a řádky n. Paměťová náročnost je lineární $b * d$ a odpovídá průměrnému faktoru větvení b při použití metody backtracking v hloubce d se správným řešením hry Griddlers. Minimální paměťová náročnost odpovídá velikosti třídy, která obsahuje zadání a řešení zakódovaného obrázku.

Prohledávání stavového prostoru je možné do doby, dokud se dá uplatnit jakýkoliv z množiny operátorů. V případě nemožnosti uplatnění jakéhokoliv operátoru bude použita metoda zpětného prohledávání neboli backtracking. Díky této metodě může být posléze nalezeno správné řešení nebo navrácení se do výchozího stavu a pokračování další variantou při hledání správného řešení.

Algoritmus griddler:

1. Dokud není nalezeno řešení, pokračuj dále, jestli je nalezeno správné řešení, ukonči prohledávání jako úspěšné.
2. Aplikuj postupně jeden operátor z množiny operátorů na řádek nebo sloupec.
3. Jestli jsi aplikoval minimálně jeden operátor na řádek nebo sloupec, pokračuj dál od bodu 1.
4. Neaplikoval-li jsi ani jeden operátor na řádek nebo sloupec a lze použít metodu backtracking, použij tuto metodu a pokračuj dál od bodu 1.
5. Nelze-li použít žádný operátor a ani metodu backtracking, ukonči prohledávání jako neúspěšné.

Pseudokód: v následujícím pseudokódu je znázorněn princip algoritmu griddler.

```
1.  bool griddler (zadani){
2.    while (zadani != cíl){
3.      Řádky a sloupce zadani = aplikuj množinu operátorů;
4.      if(aplikován operátor == false){
5.        if(uplatnění backtracking == true){
6.          Použij metodu backtracking;
7.        }
8.        else{
9.          return false;
10.       }
11.     }
12.   }
13.   return true;
14. }
```

Vlastnosti algoritmu griddler:

Úplnost – NE
Optimálnost – NE
Časová náročnost – $\sum_{i=1}^{x} O_i * (m * n)$
Paměťová náročnost – lineární $b * d$

4.3.2 Aplikace informovaného algoritmu na hru Griddlers

Algoritmus griddler vychází ze základních metod pro řešení hry Griddlers a zachovává její pravidla.

Počáteční stav algoritmu griddler je prázdná mřížka a cílový stav odpovídá mřížce, která je správně vyřešená. Kontrola cílového stavu se provede otestováním každého řádku a sloupce. Testování spočívá v kontrole správného rozložení a velikosti skupin buněk v řádcích a sloupcích.

Na jednotlivé řádky a sloupce jsou aplikovány metody řešení této hry. Pro případ, že řešení nebude možné nalézt, bude použita metoda backtracking a potom se budou dále uplatňovat původní operátory.

4.4 Porovnání algoritmů

Při porovnávání kvality jednotlivých algoritmů je potřeba se zaměřit na jejich vlastnosti a porovnávat je podle hodnotících kritérií.

Porovnání algoritmů je provedeno podle kritérií úplnosti, optimálnosti, časové náročnosti a paměťové náročnosti. Úplnost a optimálnost je důležitá z hlediska nalezení správného řešení, protože je dobré vědět, pomocí kterého algoritmu dostaneme zaručený výsledek. Časová a paměťová náročnost nám ukáže, který algoritmus je vhodný pro hledání řešení s co nejmenšími systémovými požadavky.

Následující tabulka 4.1 znázorňuje přehledné porovnání základních algoritmů a jejich vlastností zmíněných v této kapitole.

Název algoritmu	**Úplnost**	**Optimál.**	**Časová n.**	**Paměťová n.**
Prohledávání do šířky	ANO	ANO	$O * (b^{d+1})$	$O * (b^{d+1})$
Prohledávání do hloubky	NE	NE	$O * (b^m)$	$O * (b * m)$
Algoritmus griddler	NE	NE	$\sum_{i=1}^{n} O_i * (m * n)$	$b * d$

Tabulka 4.1: Porovnání algoritmů a jejich vlastností.

Porovnání úplnosti a optimálnosti algoritmů. Z předchozí tabulky vidíme, že algoritmus prohledávání do šířky je úplný a optimální. Ostatní algoritmy také mohou nalézt správné řešení, ale toto řešení nemusí být optimální. Nalezení neoptimálního řešení je zapříčiněno vnitřní strukturou ostatních algoritmů. Původní varianta algoritmu prohledávání do hloubky není úplná, ale při aplikování na hru Griddlers, kde se u prohledávání do hloubky nastaví maximální hloubka na šířku mřížky, je prohledávání do hloubky úplné, protože při rozgenerování všech možností pro řešený uzel se vždy aplikuje pouze jeden operátor. Algoritmus griddler není úplný, protože v případě špatného uplatnění pořadí operátorů nemusí nalézt řešení.

Porovnání časové náročnosti algoritmů. Nejnáročnější algoritmus na spotřebovaný čas je metoda prohledávání do šířky, protože tato metoda rozgeneruje uzel a s každým rozgenerovaným uzlem provádí nové generování, dokud tímto postupem není nalezeno správné řešení zakódovaného obrázku. O něco méně časově náročná je metoda slepého prohledávání do hloubky, která generuje uzly postupně a prohledává stavový prostor do hloubky díky své vnitřní struktuře. Algoritmus griddler pracuje při řešení se záznamy okolo mřížky a díky této nápovědě může pracovat o mnoho rychleji. Jeho časová náročnost odpovídá součtu všech časů potřebných pro uplatňování jednotlivých operátorů na řádky a sloupce mřížky.

Porovnání paměťové náročnosti algoritmů. Porovnávání algoritmů z hlediska paměťové náročnosti je také jedna z důležitých věcí při porovnávání algoritmů. Nejnáročnější z algoritmů na spotřebovanou paměť je metoda slepého prohledávání do šířky, protože se ve frontě uchovávají všechny rozgenerované uzly, dokud není nalezen správný uzel odpovídající zadání. O něco lepší algoritmus na paměťovou náročnost z pohledu spotřebovaného množství paměti je metoda slepého prohledávání do hloubky, kde se v zásobníku také udržují všechny rozgenerované uzly, ale díky vnitřní struktuře algoritmu je paměťová náročnost pouze lineární narozdíl od exponenciální náročnosti u prohledávání do šířky. Paměťová

náročnost algoritmu griddler odpovídá součinu faktoru větvení a hloubce, která odpovídá správnému řešení zakódovaného obrázku při použití algoritmu backtracking.

Celkové srovnání algoritmů. Z celkového pohledu na vlastnosti jednotlivých algoritmů sepsaných v tabulce 4.1 můžeme říci, že pro mřížky o malých rozměrech (např.: 5 řádků a 5 sloupců) jsou použitelné všechny algoritmy. Pro mřížky o větších rozměrech (např.: 10 řádků a 10 sloupců) jsou použitelné také všechny algoritmy, ale aplikování neinformovaných algoritmů není už moc vhodné díky stoupajícím časovým a paměťovým nárokům na řešení. Mřížky o velkých rozměrech (např.: 20 řádků a 20 sloupců) jsou vhodné spíše už jenom pro informované algoritmy, protože pracují se záznamy vedle řádků a nad sloupci. Souhrnně by se dalo říci, že čím je větší mřížka v zadání pro jakýkoliv algoritmus, tím jsou větší nároky algoritmu na čas a paměť.

Zlepšení vnitřní struktury jednotlivých algoritmů může přinést velkou časovou a paměťovou úsporu. V další podkapitole 4.5 jsou diskutována možná rozšíření algoritmů nebo použití nových algoritmů pro řešení hry Griddlers.

4.5 Další možné algoritmy pro řešení

Další možné algoritmy pro řešení hry Griddlers můžeme rozdělit do dvou kategorií. První kategorie algoritmů odpovídá úpravě stávajících neinformovaných a informovaných algoritmů (viz podkapitola 4.5.1) za účelem zvýšení jejich efektivnosti a zmenšení jejich časových a paměťových nároků. Do druhé kategorie spadají takové algoritmy (viz podkapitola 4.5.2), které zde nejsou vyjmenovány a mohly by svým postupem také umožnit řešení zadaného kódovaného obrázku.

4.5.1 Rozšíření použitých algoritmů

V následujících odstavcích jsou zmíněny a navrženy možně modifikace původních algoritmů. Tyto modifikace mohou být rozšířeny o další varianty a tímto způsobem může být zdokonalován způsob luštění zakódovaného obrázku.

Modifikované slepé prohledávání do šířky (MBFS) vychází z původního algoritmu prohledávání do šířky. Rozdíl mezi původní variantou a modifikovanou spočívá v ukládání uzlů řešení do fronty OPEN. V případě, že je expandován uzel, který nemůže už vést ke správnému řešení, potom tento uzel není umístěn do fronty OPEN. Další rozdíl spočívá v prohledávané hloubce, kdy se neukládají do fronty OPEN uzly, u kterých je dosaženo maximální prohledávané hloubky, a rovnou jsou tyto uzly vyhodnoceny, jestli odpovídají požadovanému řešení. Při nalezení správného řešení je prohledávání ukončeno jako úspěšné. Po modifikování původního BFS se modifikovaný algoritmus MBFS přesouvá do skupiny informovaných algoritmů, protože při vyhodnocování uzlů už neukládá do fronty OPEN všechny uzly, ale pouze ty, které mohou vést ke správnému řešení.

Modifikované slepé prohledávání do hloubky (MDFS) vychází z původního algoritmu prohledávání do hloubky a modifikace je podobná jako u modifikovaného slepého prohledávání do šířky. Rozdíl mezi DFS a MDFS spočívá v ukládání uzlů řešení do zásobníku OPEN. V případě, že je expandován uzel, který už nemůže vést ke správnému řešení, potom tento uzel není umístěn do zásobníku OPEN. Další rozdíl spočívá v prohledávané hloubce, kdy se neukládají do zásobníku OPEN uzly, u kterých je dosaženo maximální prohledávané

hloubky, a rovnou jsou tyto uzly vyhodnoceny, jestli odpovídají požadovanému řešení. Při nalezení správného řešení je prohledávání ukončeno jako úspěšné. Po modifikování původního DFS se modifikovaný algoritmus MDFS přesouvá také do skupiny informovaných algoritmů kvůli stejnému důvodu jako BFS.

Modifikovaný algoritmus griddler vychází z původního algoritmu griddler. Modifikace může spočívat v přepracování jednotlivých operátorů pro řádky a sloupce, vytvoření nových sofistikovanějších operátorů, aplikování operátorů na řádky a sloupce, které byly v poslední době modifikovány, nebo upravení vnitřní struktury modifikovaného algoritmu griddler.

4.5.2 Rozšíření o nové algoritmy

Mezi nové algoritmy [9] pro řešení hry Griddlers mohou patřit metody omezeného prohledávání do hloubky, metoda postupného zanořování, metoda zpětného navracení, vytvoření detailnějšího algoritmu pracujícího podle způsobu řešení hry Griddlers atd.

Další variantou při zpracování je přenést řešení zakódovaného obrázku ze sériového zpracování řádků a sloupců na paralelní zpracování všech řádků zároveň a poté zpracování všech sloupců zároveň.

V této kapitole jsme si zmínili hlavní tři algoritmy (BFS, DFS a algoritmus griddler) při řešení hry Griddlers, které budou použity v řešiteli této hry. Všechny tyto algoritmy jsou zde jednotlivě rozebrány podle jejich vlastností a jsou zde i porovnány jejich jednotlivé vlastnosti navzájem. Dále v závěru této kapitoly jsme se zmínili o možnosti rozšíření o nové a stávající algoritmy při řešení zakódovaného obrázku. Následující kapitola 5 obsahuje informace o implementaci řešitele hry Griddlers. V implementaci jsou využity znalosti a poznatky z dosud uvedených kapitol.

Kapitola 5

Implementace řešitele

V předchozích kapitolách jsme si popsali hru Griddlers, návrh řešitele a základní algoritmy pro řešení této hry.

Současná kapitola se zabývá implementací řešitele hry Gridddlers a věcí s tím spojených. Především použitými vývojovými prostředky, uložením dat do XML souboru, implementací algoritmů, popisem vlastností a vytvořených tříd aplikace.

5.1 Použité vývojové prostředky

Při vytváření aplikace jsem měl na výběr z moderních programovacích jazyků jako je C++, Java, C# atd. Pro tvorbu aplikace jsem si zvolil objektově orientovaný programovací jazyk C++. Tento jazyk jsem vybral z důvodu snadnějšího pochopení implementace řešitele a samotné hry Griddlers.

Veškeré zdrojové kódy této aplikace jsem editoval ve vývojovém prostředí NetBeans IDE verze 6.8 pomocí modulu podporující jazyk C/C++. Zdrojové kódy překládám pomocí kompilátoru g++ verze 4.4.1 a vytvořeného makefile souboru s nastavením kompilace.

V aplikaci jsem potřeboval načítat a ukládat XML soubory. Pro zpracování a tvorbu XML souborů jsem si zvolil XML parser TinyXML verze 2.5.3, protože se mi líbí jeho jednoduchost, snadné propojení s vytvářenou aplikací a pohodlná práce s XML soubory.

Programátorskou dokumentaci vytvářím pomocí dokumentačního generátoru Doxygen verze 1.6.1. Každá třída je zdokumentována v programové dokumentaci a většina zdrojových kódu je popsána v řádkových komentářích přímo ve zdrojovém kódu.

Veškeré zdrojové kódy a XML soubory jsou vytvořeny v kódování ISO-8859-2. Dokumentace a vygenerována dokumentace (pomocí nástroje Doxygen) je v kódování UTF-8.

5.2 Uložení kódovaného obrázku

Jednou z nejdůležitějších součástí řešitele jsou dvě třídy, které obsahují celé a částečné zadání kódovaného obrázku, proto si více rozebereme obě tyto třídy.

První třída popsána v podkapitole 5.2.1 s názvem `matice`, obsahuje celé zadání, které se má vyřešit, a slouží k uložení a průběžnému řešení dat. Velká část aplikace tuto třídu využívá tím způsobem, že si pomocí ní předává zadání ke zpracování, částečné řešení zadání nebo už vyřešené zadání.

Druhá třída popsaná v podkapitole 5.2.2, s názvem `zaznammatice`, obsahuje pouze jeden řádek nebo sloupec, který se má vyřešit. Třída `zaznammatice` slouží k předávání řádku nebo sloupce se zadáním pro metody pracující s jednotlivými řádky nebo sloupci.

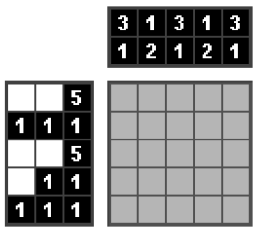
5.2.1 Třída `matice`

Celá tato třída je navržena tak, aby dokázala pojmout zadání jakékoliv zakódovaného obrázku. Třída `matice` obsahuje proměnné se základní údaji zadání, samotné zadaní a mřížku s buňkami.

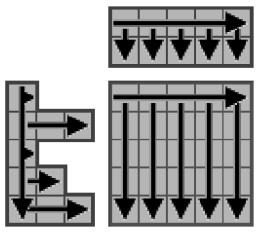
Základní údaje obsahují proměnné s názvem vstupního XML souboru, názvem řešeného obrázku, počtem řádků a počtem sloupců mřížky a logickou proměnnou udávající, jestli pří načítání dat nenastala chyba. Mezi základní údaje ještě můžeme zařadit dva vektory obsahující logické hodnoty. Vektor `vyresenyradek` odpovídá svoji délkou počtu řádků v zadání a hodnota `true` na některé pozici tohoto vektoru značí, že odpovídající řádek této pozice je správně vyřešen. Vektor `vyresenysloupec` odpovídá svoji délkou počtu sloupců v zadání a má stejný význam jako vektor `vyresenyradek`.

Samotné zadání pro řádky a sloupce je uloženo ve vektoru a každá pozice vektoru obsahuje další vektor se zadáním pro jednotlivé buňky. Znázornění vytvoření vektorů pro zadání obrázku 5.1 je ukázáno na obrázku 5.2. Zadání pro řádky je uloženo ve vektoru `radky` a zadání pro sloupce je uloženo ve vektoru `sloupce`. Při vytváření vektoru pro řádky a sloupce je dobré si povšimnout, že jednotlivé vektory určující zadání řádků a sloupců nemusí být stejně dlouhé. Vytvořený vektor je uspořádán tak, že pro obrázek 5.1 se bude v prvním sloupci a prvním záznamu vektoru `sloupce` nacházet číslo 3. Toto uspořádání vektorů vychází z logické reprezentace záznamů pro jednotlivé řádky a sloupce. Obrázek 5.2 zachycuje tuto skutečnost.

Mřížka s buňkami je tvořena vektorem `obrazek` a každá pozice vektoru obsahuje další vektor, který udává stav jednotlivých buněk mřížky. Buňka mřížky může být prázná, vybarvená barvou pozadí nebo vybarvená barvou popředí. Vektor `obrazek` je vytvořen tak, že první vektor je stejně dlouhý jako počet sloupců v zadání, a druhý vektor, na každé pozici prvního vektoru, je stejně dlouhý jako počet řádků v zadání. Mřížka v obrázku 5.2 znázorňuje tuto skutečnost.



Obrázek 5.1: Zadani kódovaného obrázku načteného z XML souboru.



Obrázek 5.2: Vektory obsahující zadání a mřížku načteného obrázku.

Podrobnější informace o těchto proměnných jsou uvedeny v dokumentaci k aplikaci. Jednotlivé metody pro vytváření dat a práci s daty této třídy jsou popsány taktéž v dokumentaci a přímo ve zdrojových kódech jednotlivých metod.

5.2.2 Třída `zaznammatice`

Třída **`zaznammatice`** je navržena pouze pro práci s jedním řádkem nebo sloupcem zadání. Příklad s jedním řádkem znázorňuje obrázek 5.3. Zobrazený řádek vychází z obrázku 5.1. Používání této třídy umožňuje snížit potřebné nároky programu na počítačové zdroje, protože se nemusí předávat tam, kde to není potřeba, celá třída **`matice`**, ale pouze jeden řádek nebo sloupec z této třídy.

Tato třída obsahuje logickou proměnnou **`sloupec`**. Ta udává, jestli se jedná o záznam řádku nebo sloupce. Hodnota **`true`** značí, že daný záznam odpovídá sloupci. Dále třída obsahuje proměnnou **`poradi`** určující pořadové číslo řádku nebo sloupce, vektor **`zaznam`** obsahující zadání pro řádek nebo sloupec a vektor **`data`** obsahující jednotlivé buňky řádku nebo sloupce. Na obrázku 5.4 jsou znázorněny dva vektory odpovídající řádku z obrázku 5.3. První vektor zleva odpovídá vektoru **`zaznam`** a druhý vektor odpovídá vektoru **`data`**.

Obrázek 5.3: Zadani jednoho řádku načteného z XML souboru.

Obrázek 5.4: Vektor obsahující zadání a jeden řádek obrázku.

Podrobnější informace o těchto proměnných jsou uvedeny v dokumentaci k aplikaci. Jednotlivé metody pro práci se záznamem řádku nebo sloupce této třídy jsou popsány taktéž v dokumentaci a přímo ve zdrojových kódech jednotlivých metod.

5.3 Implementace algoritmů

V kapitole 4 jsme se seznámili s některými základními algoritmy pro řešení hry Griddlers, především s algoritmy slepého prohledávání do šířky (viz podkapitola 4.2.1), slepého prohledávání do hloubky (viz podkapitola 4.2.2) a algoritmem griddler (viz podkapitola 4.3.1).

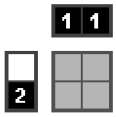
Vytvořená aplikace obsahuje 5 algoritmů použitelných pro řešení zadání. Mezi těchto 5 algoritmů patří slepé prohledávání do šířky, slepé prohledávání do hloubky, algoritmus griddler, modifikovaný algoritmus slepého prohledávání do šířky a do hloubky. Modifikované algoritmy vycházejí z odstavců v podkapitole 4.5.1 zabývající se modifikováním slepého prohledávání do šířky a hloubky.

Slepé prohledávání do šířky a do hloubky (BFS, DFS). Algoritmy BFS a DFS jsou si velmi podobné z implementačního hlediska, ale vlastnosti algoritmů jsou velmi rozdílné. Oba algoritmy dědí z třídy **`neinformalg`** všechny metody a proměnné, které jsou navrženy tak, že jsou použitelné v mnoha algoritmech. Podrobnější informace o třídě **`neinformalg`** jsou uvedeny v podkapitole 5.5 a v dokumentaci. Zde bude zmíněn pouze operátor **`shift`** z této třídy.

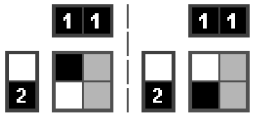
Implementace BFS a DFS je velmi podobná uvedeným algoritmům. Při řešení se používá fronta respektive zásobník OPEN, do které se ukládá celá třída **`matice`** obsahující zadání a postupně i řešení. Princip práce s frontou je následující:

1. Na začátku algoritmu ulož do fronty OPEN prázdnou třídu `matice`.
2. Při expandování uzlu vyber třídu `matice` z fronty OPEN.
3. Ve vybrané třídě `matice` najdi první sloupec, který ještě není vyřešený, a poznač si jeho index.
4. Na nalezený sloupec, odpovídající poznačenému indexu, aplikuj operátor `shift` – pokud je to ještě možné.
5. Po každém aplikování operátoru `shift` ulož do fronty OPEN třídu `matice`, která obsahuje pozměněný sloupec operátorem `shift`.

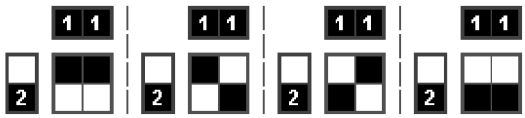
Tímto způsobem hledám postupně řešení celého zadání (obrázek 5.5). Nejprve hledám všechny možnosti řešení pro první sloupec zadání (obrázek 5.6), poté pro druhý sloupec zadání (obrázek 5.7) a až nakonec dojdu k poslednímu sloupci zadání. Po expandování posledního sloupce zadání, by se měl ve frontě OPEN nacházet správný výsledek odpovídající zadání (obrázek 5.7). Stejný princip, popsaný v předchozím postupu, platí i pro zásobník OPEN v algoritmu DFS.



Obrázek 5.5: Obsah fronty OPEN po vložení třídy `matice`.



Obrázek 5.6: Obsah fronty OPEN po expandování 1. sloupce.

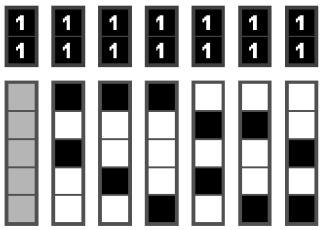


Obrázek 5.7: Obsah fronty OPEN po expandování 2. sloupce.

Operátor `shift` pracuje na principu posouvání skupin buněk s barvou popředí v zadaném řádku nebo sloupci. Na obrázku 5.8 je znázorněn princip tohoto operátoru. Popis činnosti operátoru `shift`:

1. Jestli operátor dostane nevyřešený sloupec (viz 1. sloupec zleva v obrázku 5.8), operátor doplní první možnou variantu tohoto sloupce od jeho vrcholu (viz 2. sloupec zleva).
2. Jestli operátor dostane vyřešený sloupec (viz 2. sloupec zleva), pokusí se v tomto sloupci posunout poslední skupinu buněk o jednu pozici dolů (viz 3. sloupec zleva) a při dalším uplatnění operátoru posune skupinu buněk znova o jednu pozici dolů (viz 4. sloupec zleva).
3. Jestli nelze posunout poslední skupinu buněk o jednu pozici dolů (viz 4. sloupec zleva), posune operátor druhou skupinu buněk od spodu sloupce o jednu pozici dolů a poslední skupinu buněk přisune nahoru (viz 5. sloupec zleva).

4. Tento princip posouvání je uplatnitelný, dokud operátor bude mít možnost posouvat nějakou skupinu buněk od vrchu dolů. Po posunutí poslední možné skupiny buněk dolů (viz 7. sloupec zleva) operátor rozgeneroval všechny možné kombinace správného řešení daného sloupce.



Obrázek 5.8: Znázornění činnosti operátoru shift na zadání sloupce.

Operátor `shift` je možné se stejným postupem použít i na řádky, ale není to moc vhodné, protože při aplikování operátoru na řádku je potřeba projít všechny vektory sloupců a v nich najít hodnotu příslušného řádku. Díky nutnosti procházet všechny vektory sloupců dochází k větším výpočetním nárokům algoritmů. Nevýhoda aplikování operátoru `shift` na řádky je zapříčiněna způsobem tvoření vektorů ukázaným na obrázku 5.2.

Algoritmus griddler. Implementace algoritmu griddler částečně vychází z pseudokódu v podkapitole 4.3.1. Tento algoritmus při uplatňování jednotlivých operátorů na řádky a sloupce dodržuje pravidla hry Griddlers a tím zmenšuje svoji výpočetní náročnost.

V algoritmu griddler je použita metoda zpětného navracení (backtracking). Implementace metody backtracking pro algorimuts griddler zapříčinila částečnou změnu výsledného kódu oproti ukázanému pseudokódu v podkapitole 4.3.1, ale smysl výsledného kódu a pseudokódu je si velmi blízký. Princip použití metody backtracking v algoritmu griddler:

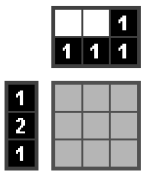
1. Na začátku algoritmu ulož do zásobníku OPEN prázdnou třídu `matice`, kterou budeš řešit.
2. Vyber z vrcholu zásobníku OPEN třídu `matice`.
3. Vybranou třídu `matice` nech zpracovávat algoritmem griddler. Jestli algoritmus nalezne požadovaný výsledek, řešení ukonči jako úspěšné. V případě, že nelze uplatnit žádný operátor na třídu `matice`, pokračuj ve 4. bodě.
4. Uplatnily-li se operátory v takovém pořadí, že neprovedly žádnou změnu, pokračuj znova od bodu 2. V opačném případě se pokus najít nejvhodnější sloupec v řešené mřížce, na který uplatníš metodu backtracking.
5. Při uplatnění metody backtracking vygeneruj všechny možné varianty pro zvolený sloupec a postupně ulož všechny třídy `matice` se změněným sloupcem do zásobníku OPEN. Vygenerování všech možných variant pro daný sloupec je nutné při použití této metody, protože není možné zpětně v řešení dohledat místo, kde byla tato metoda uplatněna. Dál pokračuj od bodu 2.
6. Jestli nebude možné v průběhu řešení dál používat metodu backtracking a nebude možné vybrat z vrcholu zásobníku OPEN třídu `matice`, ukonči hledání řešení jako neúspěšné.

Modifikované slepé prohledávání do šířky a do hloubky (MBFS, MDFS). Algoritmy MBFS a MDFS jsou z implementačního hlediska podobné, a proto je můžeme popsat společně. Oba tyto algoritmy vycházejí z algoritmů BFS a DFS a dědí z třídy `neinformalg` všechny metody a proměnné. Podrobnější informace o třídě `neinformalg` jsou uvedeny v podkapitole 5.5 a v dokumentaci.

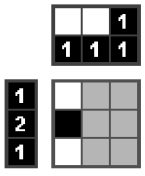
Implementace MBFS a MDFS vychází z implementace algoritmům BFS a DFS, ale jsou zde některé změny, které umožňují MBFS a MDFS dosáhnout především menších paměťových nároků. Při řešení se používá fronta respektive zásobník OPEN, do které se ukládá celá třída `matice` obsahující zadání a postupně i řešení. Princip práce s frontou je následující:

1. Na začátku algoritmu ulož do fronty OPEN prázdnou třídu `matice`.
2. Při expandování uzlu vyber třídu `matice` z fronty OPEN.
3. Ve vybrané třídě `matice` najdi první sloupec, který ještě není vyřešený, a poznač si jeho index.
4. Na nalezený sloupec odpovídající poznačenému indexu aplikuj operátor `shift` – pokud je to ještě možné.
5. Jestliže pracuješ s posledním sloupcem zadání, tak už neukládej žádnou třídu `matice` do fronty OPEN, ale rovnou vyhodnoť, jestli tato třída obsahuje správný výsledek.
6. Jestli ještě nepracuješ s posledním sloupcem zadání, tak po každém aplikování operátoru `shift` na třídu `matice` zkontroluj, jestli tato třída může vést ke správnému řešení. V případě, že pozměněná třída může vést ke správnému řešení, tak ulož tuto třídu do fronty OPEN, v opačném případě tuto třídu už nikam neukládej.

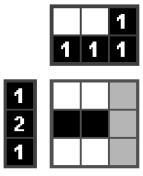
Postupné hledání řešení zadání (obrázek 5.9) je stejné jako u algoritmů BFS a DFS. Rozdíl je pouze v tom, co se ukládá do fronty OPEN. Nejprve hledám všechny možnosti řešení pro první sloupec zadání (obrázek 5.10), které mohou vést ke správnému řešení, poté hledám všechny možnosti řešení pro druhý sloupec zadání (obrázek 5.11), které mohou vést ke správnému řešení. Tímto postupem pokračuji až k poslednímu sloupci (obrázek 5.11), který už neukládám do fronty, ale rovnou vyhodnotím, jestli jsem nalezl správné řešení. Stejný princip, popsaný v předchozím postupu, platí i pro zásobník OPEN v algoritmu MDFS.



Obrázek 5.9: Obsah fronty OPEN po vložení třídy `matice`.



Obrázek 5.10: Obsah fronty OPEN po expandování 1. sloupce pomoci MBFS.



Obrázek 5.11: Obsah fronty OPEN po expandování 2. sloupce pomoci MBFS.

5.4 Implementované vlastnosti

Vytvořená aplikace vychází z navrhovaných vlastností v podkapitole 3.4, které byly rozšířeny o měření spotřebovaného času a počtu vytvořených tříd `matice` použitých při řešení.

Shrnutí implementovaných vlastností řešitele:

- Program je spustitelný v konzoli a pracuje se zadanými vstupními parametry při spuštění.
- Běh programu je možné ovlivnit přepínači zadanými při spuštění. Jsou implementovány přepínače pro: vypsání základní nápovědy k programu, volbu algoritmů použitých pro řešení, zobrazení a uložení vyřešeného zadání. Seznam implementovaných přepínačů:

  $-h$ – vypsání nápovědy k programu do konzole

  $-i$ – zobrazení nalezeného řešení do konzole

  $-s$ – uložení nalezeného řešení do XML souboru

  $-a$ alg – volba algoritmů, které se mají použít pro řešení. Bez uvedení přepínače $-a$ se implicitně použijí všechny algoritmy. Parametr alg specifikuje požadované algoritmy. Znaky použitelné pro požadované algoritmy jsou: b = BFS, d = DFS, g = algoritmus griddler, m = MBFS a n = MDFS. Stejné znaky algoritmů lze také zadávat i velkými písmeny (B, D, G, M a N).
- Zadání kódovaných obrázků je uloženo v XML souborech, které se uvádějí mezi parametry při spuštění.
- V případě, že je použit přepínač $-s$, je uložen výsledek do XML souboru. Název vytvořeného XML souboru se skládá z názvu XML souboru se zadáním a přidáním slova $final$. V případě problémů s vytvořením nového souboru se program pokusí přepsat vstupní soubor. Název souboru s výsledkem je vypsán do konzole.
- Každému algoritmu je změřen jeho spotřebovaný reálný a uživatelský čas.
- Při běhu algoritmů se počítá kolik tříd `matice` je použito pro řešení a jaká je maximální velikost fronty nebo zásobníku OPEN.
- Výsledek řešení je postupně vypisován do konzole po skončení běhu každého algoritmu. Je vypisován:

  název použitého algoritmu

  název zadání

  výsledek řešení (zda bylo nalezeno řešení)

  spotřebovaný reálný a uživatelský čas

  počet vytvořených tříd `matice` a maximální délka fronty nebo zásobníku OPEN

Příklad výpisu výsledků řešení jednoho zadání:

```
+===============+===============+=======+===============+======================+
| algorithm     | matrix name   | resul.| real / user t.| use / max matrix
+===============+===============+=======+===============+======================+
| BFS           | 1000x250      | true  | 4 / 3.54      | 251 / 1
| MBFS          | 1000x250      | true  | 13 / 12.09    | 251 / 1
| DFS           | 1000x250      | true  | 4 / 3.59      | 251 / 1
| MDFS          | 1000x250      | true  | 13 / 12.61    | 251 / 1
| griddler      | 1000x250      | true  | 0 / 0.07      | 1 / 1
+===============+===============+=======+===============+======================+
```

5.5 Popis vytvořených tříd řešitele

Na následujících řádcích se nachází přehled vytvořených tříd aplikace s jejich popisem. Vytvořená aplikace vychází z návrhu diagramu tříd uvedeného v podkapitole 3.5.

`bfs` – řešení zadaného obrázku pomocí algoritmu slepého prohledávání do šířky. Třída `bfs` dědí metody od třídy `neinformalg`, které jsou stejné pro všechny neinformované algoritmy. Ve třídě `bfs` je implementována metoda slepého prohledávání do šířky.

`dfs` – řešení zadaného obrázku pomocí algoritmu slepého prohledávání do hloubky. Třída `dfs` dědí metody od třídy `neinformalg`, které jsou stejné pro všechny neinformované algoritmy. Ve třídě `dfs` je implementována metoda slepého prohledávání do hloubky.

`getxml` – třída sloužící jako prostředník mezi funkcí main a třídou `ioxml`. Tato třída postupně volá třídu `ioxml` a požaduje po ní načtení dat ze zadaného XML souboru.

`griddler` – řešení zadaného obrázku pomocí algoritmu griddler. Ve třídě `griddler` je implementován algoritmus griddler pracující pomocí základních principů používaných ve hře Griddlers.

`ioxml` – třída sloužící pro načítání dat z XML souborů do třídy `matice` a ukládání třídy `matice` do XML souborů. Tato třída spolupracuje s XML parsrem TinyXML.

`main` – hlavní funkce programu starající se o postupné provádění operací. Funkce `main` provede zpracování parametrů třídou `param`, dále načte data z XML souborů, předá je třídě `solver` na zpracování a výsledek řešení je možné uložit a zobrazit pomocí třídy `stat`.

`matice` – třída speciálně navržená pro ukládání dat načtených z XML souboru s následnou možností jejich uložení zpátky do XML souboru. Tato třída obsahuje mimo jiné metody pro kontrolu správného načtení dat z XML souboru, načtení řádku nebo sloupce do třídy `zaznammatice` a uložení pozměněného řádku nebo sloupce ze třídy `zaznammatice`.

`mbfs` – řešení zadaného obrázku pomocí modifikovaného algoritmu slepého prohledávání do šířky. Třída `mbfs` dědí metody od třídy `neinformalg`, které jsou stejné pro všechny neinformované algoritmy. Ve třídě `mbfs` je implementována metoda modifikovaného prohledávání do šířky. Tato metoda vychází z metody implementované ve třídě `bfs`.

`mdfs` – řešení zadaného obrázku pomocí modifikovaného algoritmu slepého prohledávání do hloubky. Třída `mdfs` dědí metody od třídy `neinformalg`, které jsou stejné pro všechny neinformované algoritmy. Ve třídě `mdfs` je implementována metoda modifikovaného prohledávání do hloubky. Tato metoda vychází z metody implementované ve třídě `dfs`.

`neinformalg` – třída, ze které je možné zdědit metody použitelné u podobných algoritmů (především u neinformovaných algoritmů). Třída `neinformalg` nabízí metody pro posunutí bloků v řádku nebo sloupci, nastavení a zjištění spotřebovaného času, zjištění výsledu algoritmu a třídu `matice` s výsledkem řešení.

`param` – první třída volaná v řešiteli, která se stará o zpracování zadaných parametrů a použitých přepínačů ovlivňující činnost programu.

`solver` – hlavní jádro řešitele starající se o postupné volání tříd s jednotlivými algoritmy, které se snaží definovaným způsobem nalézt řešení. Tato třída umožňuje přidávání nových tříd s dalšími implementovanými algoritmy a zároveň se stará o vypisování tabulky s výsledky řešení pro daný algoritmus.

`stat` – třída sloužící pouze jako prostředník pro ukládání třídy `matice` do XML souboru voláním metod třídy `ioxml` a vyobrazením nalezeného výsledku do konzole.

`zaznammatice` – třída speciálně navržená pro práci s jedním řádkem nebo sloupcem třídy `matice`. V této třídě je možné přímo přistupovat pomocí metod k vektoru se zadáním a k vektoru s daty řádku respektive sloupce. Vektor s daty řádku respektive sloupce lze změněný zpátky uložit do této třídy a následným voláním příslušné metody ve třídě `matice` změnit řádek respektive sloupce s daty.

V této kapitole jsme se seznámili se samotnou implementací řešitele hry Griddlers. Uvedli jsme si zde všechny použité vývojové prostředky, ukládání kódovaného obrázku do tříd `matice` a `zaznammatice`, implementaci jednotlivých algoritmů, implementované vlastnosti a popsali jsme si vytvořené třídy řešitele. Následující kapitola se bude zabývat samotným testováním řešitele pomocí různých druhů zadání a zhodnocením především časových nároků na těchto zadáních.

Kapitola 6

Testování řešitele

Seznámili jsme se již s hrou Griddlers, návrhem řešitele této hry, používanými algoritmy a samotnou implementací řešitele.

Současná kapitola se zabývá testováním řešitele pomocí 4 kategorií kódovaných obrázků a vyhodnocením získaných výsledků těchto testů. Ve všech kategorií bude uveden popis testu, tabulka obsahující naměřené výsledky a znázorněn zadaný obrázek testu, bude-li to potřeba.

Testování probíhalo ve virtuálním prostředí s nainstalovaným operačním systémem Ubuntu verze 9.10 (karmic) s jádrem Linux 2.6.31-14-generic. Dostupné hardwarové prostředky při testování byly: dvoujádrový procesor Intel Core2 T5600 na frekvenci 1.83GHz, operační paměť o velikosti 1024 MB a odkládací prostor na disku o velikosti 400 MB.

6.1 Sada testovacích příkladů

Testování jsem rozdělil do 4 kategorií podle velikosti a obtížnosti jednotlivých zadání. První kategorie, popsaná v podkapitole 6.1.1, se zabývá jednoduchými obrázky s maximálním rozměrem 10 řádků a 10 sloupců. Druhá kategorie, popsaná v podkapitole 6.1.2, se zabývá složitějšími obrázky s maximálním rozměrem 20 řádků a 20 sloupců. V pořadí třetí kategorie, popsaná v podkapitole 6.1.3, se zabývá extrémně velkými zadáními například s rozměrem 500 řádků a 500 sloupců. Poslední čtvrtá kategorie, popsaná v podkapitole 6.1.4, se zabývá zadáními, které jsou nejednoznačné.

6.1.1 Testování pomocí jednoduchých obrázků

Testování v této kategorii probíhalo ve dvou variantách. V první variantě byl vytvořen obrazec šipky, který byl postupně otáčen o 90° vpravo. Otáčením obrázku docházelo k znatelným rozdílům naměřených hodnot při testování. Druhá varianta testování tvoří skupinu různých obrázků, které jsou rozdílné ve svých nárocích na zdroje počítače. Všechny použité obrázky byly o maximální velikosti 10 řádků a 10 sloupců.

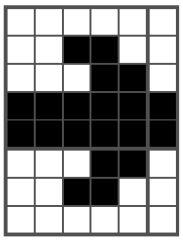
První varianta s obrázkem šipky, který byl postupně otáčen o 90° vpravo. Výchozí poloha šipky (otočená vpravo) je znázorněna na obrázku 6.1. Pro větší přehlednost tabulka 6.1 ukazuje výsledky měření seřazené nejprve podle použitého algoritmu a potom jsou uvedeny jednotlivé směry otočení obrázku. Tento způsob řazení je odlišný od řazení v tabulce, kterou řešitel vypíše do konzole. Na obrázku je zachycena pouze mřížka se šipkou bez zadání, protože po každém otočení se zadání změní. Obrázek 6.1 znázorňuje počáteční

Název alg.	Otočení obrázku	Reál. čas	Uživ. čas	Použ. tř.	Max. tř.
BFS	vpravo	4,8 s	4,258 s	60817	51450
	dolů	0,2 s	0,146 s	1457	625
	vlevo	4,8 s	4,252 s	59998	51450
	nahoru	0,2 s	0,156 s	1457	625
MBFS	vpravo	0 s	0,004 s	37	1
	dolů	0 s	0,01 s	58	13
	vlevo	0 s	0,006 s	63	3
	nahoru	0 s	0,008 s	58	13
DFS	vpravo	2,8 s	2,586 s	30244	34
	dolů	0,2 s	0,072 s	674	17
	vlevo	2,8 s	2,566 s	29935	34
	nahoru	0,2 s	0,11 s	808	17
MDFS	vpravo	0 s	0,004 s	37	1
	dolů	0 s	0,006 s	44	8
	vlevo	0 s	0,006 s	50	3
	nahoru	0 s	0,006 s	39	6

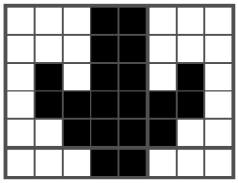
Tabulka 6.1: Porovnání naměřených hodnot při postupném otáčení obrázku.

otočením šipky vpravo. Obrázek 6.2 ukazuje pootočení šipky o 90° dolů. Položka ve sloupci otočení obrázku udává směr, kterým byl otočen obrázek při testování.

Tabulka 6.1 neobsahuje výsledky měření pro algoritmus griddler, protože pomocí tohoto algoritmu byly naměřené časy velmi blízké 0 s. Počet použitých tříd byl roven 1 a maximální délka zásobníku byla taká rovna 1.



Obrázek 6.1: Obrázek používaný pro měření, který je otočen vpravo.



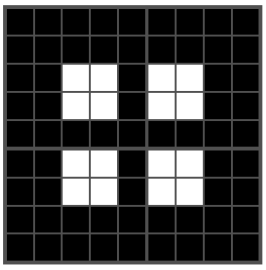
Obrázek 6.2: Obrázek používaný pro měření, který je otočen dolů.

Druhá varianta s různě obtížnými obrázky. Tabulka 6.2 ukazuje výsledky měření seřazené podle použitého algoritmu a potom jsou uvedeny jednotlivé řešené zadání. Tento způsob řazení je odlišný od řazení v tabulce, kterou řešitel vypíše do konzole. Položka ve sloupci název obrázku udává jméno obrázku, který byl řešen. Zadání s oknem je na obrázku 6.3, se spirálou je na obrázku 6.4 a zadání se znakem je na obrázku 6.5.

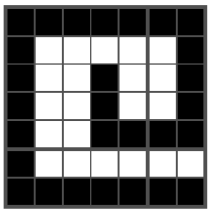
Tabulka 6.2 opět neobsahuje výsledky měření pro algoritmus griddler, protože pomocí tohoto algoritmu byly naměřené časy velmi blízké 0 s. Počet použitých tříd byl roven 1 a maximální délka zásobníku byla taká rovna 1.

Název alg.	Název obrázku	Reál. čas	Uživ. čas	Použ. tř.	Max. tř.
BFS	Okno	3,6 *s*	3,344 *s*	31213	10000
	Spirála	4,4 *s*	4,032 *s*	47717	22500
	Znak	2,2 *s*	1,824 *s*	21574	9216
MBFS	Okno	0 *s*	0,01 *s*	46	1
	Spirála	0 *s*	0,008 *s*	54	1
	Znak	0 *s*	0,02 *s*	134	14
DFS	Okno	2,2 *s*	1,968 *s*	17361	37
	Spirála	3,8 *s*	3,434 *s*	34048	47
	Znak	1,6 *s*	1,47 *s*	14880	20
MDFS	Okno	0 *s*	0,01 *s*	46	1
	Spirála	0 *s*	0,006 *s*	54	1
	Znak	0 *s*	0,012 *s*	79	6

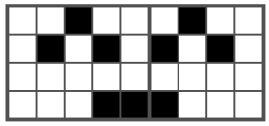
Tabulka 6.2: Porovnání naměřených hodnot pro malé obrázky.



Obrázek 6.3: Obrázek s oknem.



Obrázek 6.4: Obrázek se spirálou.



Obrázek 6.5: Obrázek se znakem.

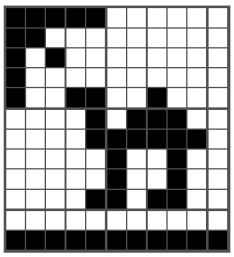
6.1.2 Testování pomocí složitějších obrázků

Testování v této kategorii probíhalo pouze s algoritmy MBFS, MDFS a algoritmem griddler. Pro testování jsou použity obrázky o maximálním rozměru 20 řádků a 20 sloupců. Algoritmy BFS a DFS jsem v testování záměrně vynechal kvůli jejich velké časové a paměťové náročnosti. Tyto vynechané algoritmy by také dokázaly vyřešit testovací zadání v této kategorii, ale spotřebovaly by velké množství výpočetního výkonu.

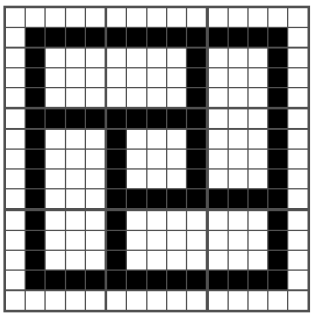
Tabulka 6.3 ukazuje výsledky měření seřazené podle použitého algoritmu a potom jsou uvedeny jednotlivé řešené zadání. Tento způsob řazení je odlišný od řazení v tabulce, kterou řešitel vypíše do konzole. Položka ve sloupci název obrázku udává jméno obrázku, který byl řešen. Zadání s pouští je na obrázku 6.6, se čtvercem je na obrázku 6.7 a zadání s domem je na obrázku 6.8.

Název alg.	Název obrázku	Reál. čas	Uživ. čas	Použ. tř.	Max. tř.
MBFS	Poušť	46 *s*	38,842 *s*	358469	5783
	Čtverec	47,4 *s*	39,49 *s*	342351	1610
	Dům	26,6 *s*	22,508 *s*	146808	1648
MDFS	Poušť	46,4 *s*	39,174 *s*	358364	55
	Čtverec	22,2 *s*	18,804 *s*	175810	382
	Dům	9,6 *s*	8,126 *s*	53036	129
griddler	Poušť	0 *s*	0,006 *s*	1	1
	Čtverec	0 *s*	0,006 *s*	1	1
	Dům	0,2 *s*	0,01 *s*	1	1

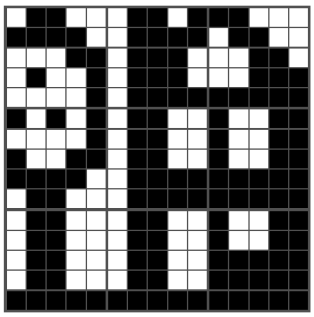
Tabulka 6.3: Porovnání naměřených hodnot pro složitější obrázky.



Obrázek 6.6: Obrázek s pouští.



Obrázek 6.7: Obrázek se čtvercem.



Obrázek 6.8: Obrázek s domem.

6.1.3 Testování pomocí extrémně velkých obrázků

V této podkapitole si ukážeme, jak ovlivňuje výsledek testu, především naměřeného času, samotný rozměr obrázku. Všechna testovaná zadání budou obsahovat stejný počet buněk k vyřešení.

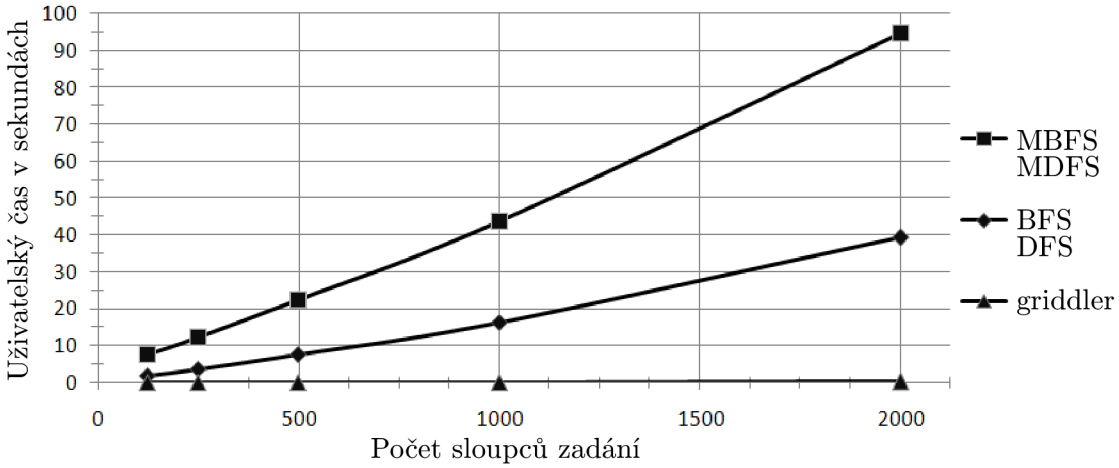
Pro ukázání tohoto jevu jsem zvolil mřížku s 250000 buňkami. Všechny buňky tohoto testu budou obsahovat pouze barvu pozadí, aby nevytvářely žádný obrazec, který by zpomaloval samotný průběh řešení. Výsledek řešení bude pouze ovlivňovat samotný rozměr mřížky. Rozměry mřížky jsem zvolil následující: 2000 řádků x 125 sloupců, 1000 řádků x 250 sloupců, 500 řádků x 500 sloupců, 250 řádků x 1000 sloupců a 125 řádků x 2000 sloupců.

V této kategorii testování jsem použil všechny implementované algoritmy, protože řešená mřížka bude ve všech buňkách obsahovat pouze barvu pozadí. Tudíž jsou použitelné všechny algoritmy i BFS a DFS.

Z naměřených výsledků je patrné, že jsou zpracovávány rychleji mřížky s větším počtem řádků a menším počtem sloupců. Tento jev je zapříčiněný vnitřní strukturou ukládání dat, který je popsán v podkapitole 5.2.1, a tato struktura je znázorněna na obrázku 5.2. V naměřených hodnotách je viditelná určitá časová závislost. Tato závislost se dá popsat takto: čím je větší počet sloupců v zadání, tím stoupá časová náročnost na zpracování, i když pořád zpracováváme stejný počet buněk. Grafická reprezentace nameřených hodnot je znázorněna na obrázku 6.9.

Název alg.	Rozměr obrázku	Reál. čas	Uživ. čas	Použ. tř.	Max. tř.
BFS	2000 ř. x 125 s.	2 *s*	1,89 *s*	126	1
	1000 ř. x 250 s.	4 *s*	3,542 *s*	251	1
	500 ř. x 500 s.	8,4 *s*	7,532 *s*	501	1
	250 ř. x 1000 s.	17,6 *s*	16,236 *s*	1001	1
	125 ř. x 2000 s.	42,4 *s*	39,178 *s*	2001	1
MBFS	2000 ř. x 125 s.	8,6 *s*	7,656 *s*	126	1
	1000 ř. x 250 s.	13 *s*	12,096 *s*	251	1
	500 ř. x 500 s.	24,2 *s*	22,518 *s*	501	1
	250 ř. x 1000 s.	47,2 *s*	43,624 *s*	1001	1
	125 ř. x 2000 s.	102,4 *s*	94,586 *s*	2001	1
DFS	2000 ř. x 125 s.	2 *s*	1,958 *s*	126	1
	1000 ř. x 250 s.	4 *s*	3,596 *s*	251	1
	500 ř. x 500 s.	8 *s*	7,186 *s*	501	1
	250 ř. x 1000 s.	17,4 *s*	15,968 *s*	1001	1
	125 ř. x 2000 s.	42,4 *s*	39,466 *s*	2001	1
MDFS	2000 ř. x 125 s.	9 *s*	8,038 *s*	126	1
	1000 ř. x 250 s.	13,4 *s*	12,612 *s*	251	1
	500 ř. x 500 s.	24 *s*	22,218 *s*	501	1
	250 ř. x 1000 s.	47,4 *s*	43,454 *s*	1001	1
	125 ř. x 2000 s.	101,4 *s*	93,858 *s*	2001	1
griddler	2000 ř. x 125 s.	0 *s*	0,068 *s*	1	1
	1000 ř. x 250 s.	0 *s*	0,074 *s*	1	1
	500 ř. x 500 s.	0 *s*	0,074 *s*	1	1
	250 ř. x 1000 s.	0 *s*	0,092 *s*	1	1
	125 ř. x 2000 s.	0 *s*	0,104 *s*	1	1

Tabulka 6.4: Porovnání naměřených hodnot na obrázcích s extrémními rozměry.



Obrázek 6.9: Graf závislosti spotřebovaného času na počtu řešených sloupců.

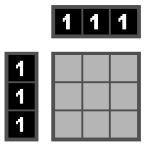
6.1.4 Testování pomocí nejednoznačného zadání

Poslední kategorie testování spočívala v obrázcích, které obsahují nejednoznačné zadání. Nejednoznačné zadání obrázku vypadá na první pohled úplně stejně jako klasické zadání, ale při řešení tohoto zadání se dostaneme do míst, kde si musíme zvolit nějakou z možných variant, abychom dospěli k jednomu z více možných řešení.

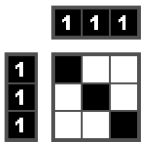
V této kategorii testovacích příkladů nejde tolik o naměřené časy a počty použitých tříd `matice`, ale spíše o to ukázat, jaké z možných řešení naleznou jednotlivé algoritmy. Nalezení různých řešení nejednoznačného zadání je zapříčiněno strukturou algoritmu a jeho vnitřní činnosti.

Výsledky této kategorie si ukážeme na malém obrázku 6.10 s nejednoznačným zadáním. Algoritmy BFS a MBFS naleznou řešení ukázané na obrázku 6.11. Nalezení tohoto výsledku je zapříčiněno používáním fronty OPEN. Algoritmy DFS, MDFS a griddler naleznou řešení ukázané na obrázku 6.12. Nalezení tohoto výsledku je zapříčiněno používáním zásobníku OPEN.

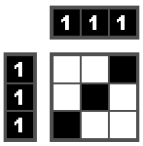
Na obrázcích 6.13 až 6.16 jsou ukázány další správné výsledky řešení tohoto nejednoznačného zadání.



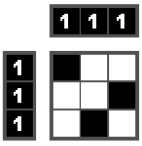
Obrázek 6.10: Zadání s nejednoznačným řešením.



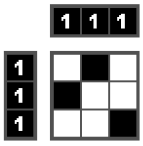
Obrázek 6.11: Varianta řešení číslo 1.



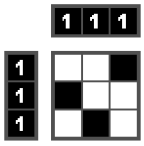
Obrázek 6.12: Varianta řešení číslo 2.



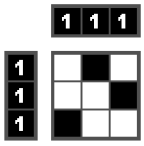
Obrázek 6.13: Varianta řešení číslo 3.



Obrázek 6.14: Varianta řešení číslo 4.



Obrázek 6.15: Varianta řešení číslo 5.



Obrázek 6.16: Varianta řešení číslo 6.

6.2 Zhodnocení testovacích příkladů

Testování řešitele probíhalo v již dříve zmíněném prostředí. Veškeré testování řešitele jsem spouštěl samostatně bez dalších zbytečně běžících procesů, které by spotřebovávaly výpočetní výkon. Každé zadání jsem spouštěl minimálně 5 krát, abych získal přesnější výsledky. Následně jsem z výsledků udělal průměr a zanesl je do tabulek 6.1 až 6.4 v této kapitole. U některých měření je reálně spotřebovaný čas menší než spotřebovaný uživatelský čas. Tato nepřesnost je způsobena v rozdílných metodách měření času a zaokrouhlování na menší počet desetinných míst. Na měření uživatelského času je použitá dostupná metoda, která je přesnější než metoda pro měření reálného času. V následujících odstavcích jsou zhodnoceny testy jednotlivých kategorií.

Testování pomocí jednoduchých obrázků. V této kategorii probíhalo testování řešitele na zadání o maximálním rozměru 10 řádků a 10 sloupců a mělo dvě varianty. Pro testování byly použity všechny implementované algoritmy. V tabulce 6.1 jsou uvedeny naměřené výsledky první varianty a v tabulce 6.2 jsou uvedeny naměřené výsledky druhé varianty testu. Z tabulky 6.1 je patrné, jak se odlišují výsledky měření uvedených algoritmů při použití stejného obrázku, který se pouze otáčí o 90° vpravo. Tabulka 6.2 ukazuje naměřené výsledky algoritmů různých testovacích obrázků. V naměřených hodnotách můžeme zpozorovat, jak se liší výsledky měření jednotlivých obrázků s podobnou velikostí. V této kategorii malých obrázků jsou použitelné všechny vytvořené algoritmy s relativně rychlým výsledkem nalezení řešení. Řešení zadaných obrázků nejrychleji nalezly algoritmy MBFS, MDFS a griddler.

Testování pomocí složitějších obrázků. V druhé testovací kategorii probíhalo testování řešitele na zadání o maximálním rozměru 20 řádků a 20 sloupců. Pro řešení těchto zadání byly použity algoritmy MBFS, MDFS a griddler. Úmyslně jsem vynechal v testování algoritmy BFS a DFS kvůli jejich velké výpočetní náročnosti. Tabulka 6.3 ukazuje naměřené výsledky druhé kategorie. Z naměřených výsledků je patrné, že náročnost těchto zadání stoupla a algoritmům trvá zpracování obrázků značně delší dobu. Řešení zadaných obrázků nejrychleji nalezl algoritmus griddler. Algoritmy MBFS a MDFS, které byly v předchozí testovací kategorii velmi úspěšné, nyní spotřebují větší množství výpočetních zdrojů na vyřešení zadání.

Testování pomocí extrémně velkých obrázků. Kategorie s testováním řešitele pomocí extrémně velkých zadání ukazuje jednu zajímavost. Pro testování řešitele byla použita zadání s mřížkou obsahující vždy stejný počet buněk a to 250000. Jelikož zadání obsahovalo pouze buňky s barvou pozadí, byly použitelné pro tyto testy všechny vytvořené algoritmy. Tabulka 6.4 ukazuje naměřené výsledky, které jsou graficky znázorněny na obrázku 6.9. Z obrázku je patrné, že počet sloupců v zadání obrázku ovlivňuje rychlost řešení jednotlivých algoritmů. Zvětšení počtu sloupců o dvojnásobek (se zachováním počtu buněk) vede k zvýšení doby řešení přibližně na dvojnásobnou hodnotu. Nejrychleji pracoval algoritmus griddler. Naopak algoritmy MBFS a MDFS jsou oproti BFS a DFS o mnoho pomalejší než v jiných testech. Důvod tohoto zpomalení u MBFS a MDFS je zapříčiněn jejich větší výpočetní náročností při řešení oproti BFS a DFS.

Testování pomocí nejednoznačného zadání. V této kategorii testování nešlo tolik o naměřené výsledky, ale spíše o nalezení řešení pro nejednoznačné zadání. V testování byly použity všechny algoritmy. Díky odlišné vnitřní struktuře algoritmů byly nalézány rozdílná řešení pro jedno zadání.

V této kapitole jsme se seznámili s výsledky měření testovacích příkladů a s jednotlivými obrázky pro tato zadání. Mezi zajímavé části této kapitoly patří podkapitola 6.1.1 ukazující výsledky testování v závislosti na pootočení obrázku, podkapitola 6.1.3 s výsledky extrémně velkých zadání a podkapitola 6.1.4 s ukázkou nejednoznačného zadání a možných řešení. Následující kapitola se bude zabývat přínosem této práce, možnými rozšířeními a vyhodnocením algoritmů.

Kapitola 7

Přínos práce

Celá tato kapitola vychází z předchozích stran a vytvořeného řešitele hry Griddlers. V podkapitolách 7.1 a 7.2 jsou především diskutovány přínosy této práce a její možné rozšíření. Podkapitola 7.3 vyhodnocuje použité algoritmy a jejich kvality.

7.1 Přínos práce

Přínos této práce spočívá v poukázání na různé metody hledání výsledků ze zadaných metadat. Na předchozích stranách je detailně popsán způsob nalezení výsledků v problematice zabývající se hrou Griddlers.

Součástí této práce je i vytvořený řešitel hry Griddlers, který vychází z popsaných způsobů. Řešitel byl vytvořen úmyslně v programovacím jazyce C++, aby byl snadněji pochopitelný a čitelný pro zájemce této problematiky. Na vytvořeném řešiteli je demonstrováno hledání správných výsledků hry Griddlers v její základní černobílé variantě, aby bylo snadnější pochopit použité metody řešení.

Při hledání výsledků z metadat můžeme přirovnat metadata k číselnému zadání řádků a sloupců. Za výsledek našeho hledání můžeme považovat mřížku s buňkami, které jsou vyplněné po ukončení hledání.

Podobným způsobem (hledání výsledků z metadat) můžeme nacházet uplatnění programů pracujících na obdobných principech v různých odvětvích lidské činnosti. Takovéto programy mohou nalézt uplatnění v opravě poškozených datových záznamů na různých typech médií (kde se snaží vycházet ze zbylých dat a určitých zákonitostí), ve skládání datových i jiných segmentů do určité podoby, v rekonstrukci a vytváření datových záznamů v lékařském prostředí, v navrhování restaurace poškozených uměleckých děl (kde program může vycházet ze znalostí na okraji poškozeného místa a z vlastností opravovaného díla) a v dalších odvětvích zabývajících se touto problematikou.

7.2 Možná rozšíření a nedostatky řešitele

Zaměříme-li se na další možné rozšíření řešitele hry Griddlers, pak zjistíme, že je mnoho oblastí, ve kterých může být rozšířen a upravován.

Příklady rozšíření a úprav řešitele: vytvoření nových algoritmů pro hledání výsledků, upravení stávajících tříd a metod pro práci s barevným zadáním hry Griddlers, rozšíření řešení i na zadání s trojúhelníky ve čtvercové nebo šestistěnné mřížce, změna vnitřní struktury za účelem zvýšení rychlosti zpracování zadání nebo převedením sériového zpracování

zadání na paralelní. Teoretická možnost rozšíření může také spočívat v převedení zadání do 3D prostoru, kde by zadání obsahovalo mřížku se třemi rozměry složenou z krychlových buněk a zadáním pro každý sloupec z bočního, předního a horního pohledu.

Rozšíření řešitele se může týkat i vytvoření grafické nadstavby programu, rozšíření vstupních a výstupních druhů souborů se zadáním nebo přidání algoritmů pro generování jednotlivých zadání ze zadaných obrázků.

Další z možností rozšíření řešitele spočívá v úpravě jeho nedostatků. Jedná se především o ošetření chybových stavů, kdy uživatel zadá nekorektní data a následně program skončí s chybou. Dále o modifikování a rozšíření metod řešení u algoritmu griddler tak, aby nemusel začít používat metodu backtracking, když se dostane do neřešitelného místa. Lze také upravit ostatní algoritmy za účelem snížení jejich výpočetní náročnosti.

7.3 Vyhodnocení použitých algoritmů

V kapitole zabývající se algoritmy jsou popsány základní algoritmy (BFS, DFS a griddler) použité ve vytvořené aplikaci. Dále v kapitole popisující implementaci jsou uvedeny další dva algoritmy (MBFS a MDFS), které jsou rovněž implementovány ve vytvořené aplikaci. Vyhodnocení výše zmíněných algoritmů můžeme provést pomocí několika kritérií, jako je spotřebovaný čas, počet vytvořených tříd při řešení, nalezení výsledku a všestrannost.

Pokud chceme nalézt řešení co nejrychleji, je dobré použít algoritmus griddler. V případě malých zadání je ještě přijatelné použití algoritmů MBFS a MDFS. Časové nejvíce náročné jsou algoritmy BFS a DFS. Při extrémně velkých zadáních, která dokážou vyřešit všechny implementované algoritmy, se MBFS a MDFS stávají značně pomalejší než ostatní – kvůli jejich větším výpočetním nárokům.

Při vyhodnocení použitelnosti algoritmů z pohledu vytvořených tříd v průběhu řešení je na tom nejlépe algoritmus griddler, protože pokud nezačne používat metodu backtracking, vytvoří pouze jednu třídu potřebnou pro nalezení výsledku. Pořadí dalších algoritmů je následující: MDFS, MBFS, DFS a BFS. V tomto pořadí algoritmus BFS vytvoří největší množství tříd oproti algoritmu MDFS, který vytvoří o mnoho menší počet tříd.

Pokud existuje výsledek zakódovaného obrázku, dokážou ho nalézt algortimy BFS, DFS, MBFS a MDFS. Může se ovšem stát (v závislosti na zadání), že algoritmus griddler nedokáže nalézt řešení, i když existuje. Problém nenalezení řešení je zapříčiněn nesprávným pořadím použitých operátorů při řešení nebo velmi rozsáhlým zadáním, na které nelze vhodně uplatnit žádný z množiny operátorů.

Nejvšestrannější algoritmus pro hledání výsledků je algoritmus griddler i za cenu toho, že ne vždy dokáže nalézt řešení. Je-li náš požadavek zaměřen na nalezení výsledku bez ohledu na výpočetní nároky, je dobré použít algoritmy prohledávání do hloubky (MDFS a DFS).

Jednoznačně nelze určit, který z uvedených algoritmů je nejlepší, protože každý má svoje přednosti i nedostatky. Nejoptimálnější je zvolit si takový algoritmus, který si dokáže nejlépe poradit s konkrétním zadáním.

V této kapitole jsme se seznámili s přínosy této práce, možnými rozšířeními a vyhodnocením algoritmů. V následující kapitole, která je poslední, si shrneme dosažené výsledky a uvedeme závěr celé práce.

Kapitola 8

Závěr

Cílem této práce bylo vytvořit řešitele hry Griddlers a demonstrovat na něm různé přístupy k řešení daného problému. Na první pohled se to nezdá, ale celá práce zabíhá do různých odvětví lidské činnosti. Především do odvětví umělé inteligence, vytváření a navrhování algoritmů pro řešení, předávání myšlení člověka počítači, navrhování struktur XML souborů a používání objektově orientovaného programovacího jazyka. Všechna vyjmenovaná odvětví jsem uplatnil na hře Griddlers a samotném vytvoření aplikace.

Vytvořená aplikace demonstruje použití různých algoritmů za účelem nalezení správného výsledku. Pro řešení jsem použil algoritmy slepého prohledávání do šířky a do hloubky, dále jsem tyto algoritmy modifikoval, abych získal lepší výsledky při hledání řešení, a navíc jsem si ještě vytvořil vlastní algoritmus griddler, který vychází ze samotného řešení hry Griddlers. Implementování vlastního algoritmu griddler bylo pro mě velkou výzvou, kterou jsem chtěl zdolat. Nejhorší na zvládnutí této výzvy bylo to, že jsem potřeboval předat různé způsoby řešení, které používám při luštění, počítači tak, aby byly obecně použitelné pro jakékoliv zadání. Jedna ze zajímavých částí této práce byla pro mě testování vytvořených algoritmů na vzorových příkladech a odhalování skrytých nedokonalostí.

Vytvořením aplikace v objektově orientovaném jazyce C++ jsem si zdokonalil znalosti z oblasti vytváření objektově orientovaných aplikací. Za prohloubení znalostí z oblasti objektového programování jsem velmi rád, protože předpokládám, že získané dovednosti se mi v budoucnu ještě budou hodit a snadněji se mohu zapojit do vytváření složitějších programů. Díky prohloubení znalostí v objektovém programování, které mě velmi zaujalo, se domnívám, že přechod na jiný objektový jazyk mně nebude činit velké potíže.

Jsem velmi rád, že jsem mohl vytvořit tuto práci, ve které jsem se znovu a hlouběji seznámil s hrou Griddlers a následně z nově získaných znalostí vytvořil aplikaci. Pomocí vytvořených algoritmů a navržených postupů mám lepší představu o využití řešitele jako demonstrační aplikace v dalších odvětvích lidské činnosti hledající výsledek z metadat. Jsem velmi vděčný svému vedoucímu, který při konzultacích poukazoval na různé způsoby řešení a ochotně naslouchal mým návrhům souvisejících s vytvářenou aplikací a celou prací.

Vytvořené práce a s ní související řešitel splnil všechny mé předpoklady a hlavně i požadavky, které byly stanoveny jako cílové. Ovšem každý projekt není nikdy definitivně hotový, dokončený a bez chyb, protože se vždy najde nějaká oblast, kde by se dal dále rozšiřovat a zdokonalovat. Jestli budu mít v budoucnu víc času, rád bych přidal další algoritmy, pokusil se vylepšit zdrojový kód, vytvořit grafické rozhraní nebo rozšířit aplikaci na další varianty hry Griddlers.

Literatura

[1] CORMEN, T. H.: *Introduction to algorithms*. Cambridge: MIT Press, druhé vydání, 2001, ISBN 0-262-03293-7.

[2] SILENTIUM: Pravidla pro řešení malovaných křížovek. *Malované křížovky – kódované obrázky*, ročník IX, č. 2, 2010: str. 42, ISSN 1335-9533.

[3] SKONNARD, A.; GUDGIN, M.: *XML – pohotová referenční příručka: referenční příručka programátora ke XML, XPath, XSLT, XML Schema, SOAP a dalším*. Praha: Grada, první vydání, 2006, ISBN 80-247-0972-4.

[4] WRÓBLEWSKI, P.: *Algoritmy: datové struktury a programovací techniky*. Brno: Computer Press, první vydání, 2004, ISBN 80-251-0343-9.

[5] WWW stránky: Nonogram. [online], [cit. 2010-03-02].
URL http://en.wikipedia.org/wiki/Griddlers

[6] WWW stránky: Průvodce – Snadné křížovky. [online], [cit. 2010-03-03].
URL http://www.griddlers.net/pages/gt1

[7] WWW stránky: Solution techniques. [online], [cit. 2010-03-04].
URL http://en.wikipedia.org/wiki/Griddlers#Solution_techniques

[8] WWW stránky: Griddlers Net – Logické křížovky. [online], [cit. 2010-04-04].
URL http://www.griddlers.net/

[9] ZBOŘIL, F. V.; ZBOŘIL, F.: *Základy umělé inteligence: IZU*. Brno: Fakulta informačních technologií VUT, třetí vydání, 2006, studijní opora.

Seznam obrázků

Seznam tabulek

Dodatek A

Seznam příloh

Příloha číslo 1: CD s vytvořeným řešitelem hry Griddlers, zdrojovými kódy, dokumentací a elektronickou verzí této práce. Popis k obsahu CD se nachází v dodatku B.

Příloha číslo 2: ukázky dalších zadání a řešení obrázků hry Griddlers. Příloha číslo dvě se nachází v dodatku C.

Příloha číslo 3: ukázky dalších variant hry Griddlers. Příloha číslo tři se nachází v dodatku D.

Dodatek B

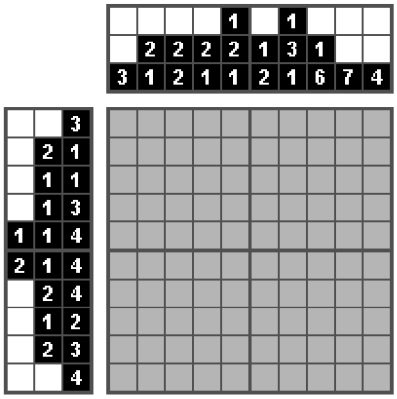
Obsah CD

Přiložené CD na zadní straně desek obsahuje data související s touto prací. Na CD se nachází:

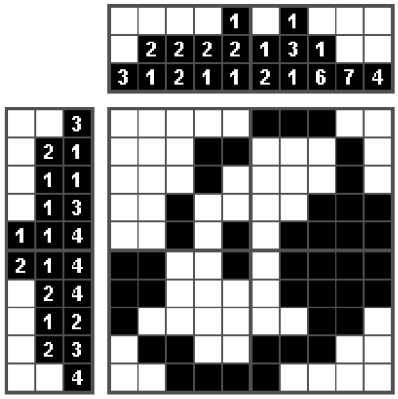
- Složka `bp` obsahuje elektronickou verzi této bakalářské práce.
- Složka `dokumentace` obsahuje vygenerovanou dokumentaci k řešiteli hry Griddlers, uživatelskou a programátorskou příručku.
- Složka `gridd` obsahuje vytvořeného řešitele hry Griddlers s testovacími příklady a skripty pro snadnější spouštění testů.
- Složka `zdroj` obsahuje zdrojové kódy řešitele hry Griddlers, testovací příklady a skripty.

Dodatek C

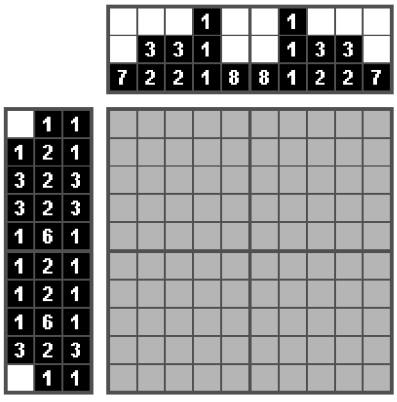
Zadání a řešení dalších obrázků hry Griddlers



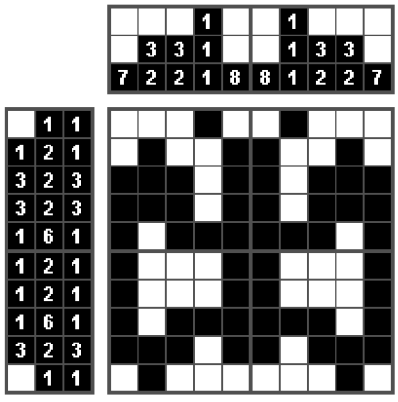
Obrázek C.1: Zadání číslo 1.



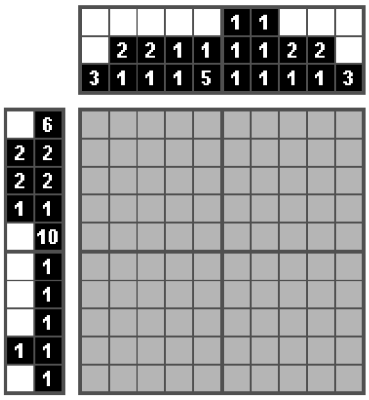
Obrázek C.2: Řešení číslo 1.



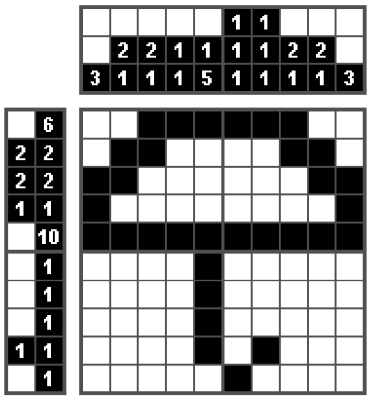
Obrázek C.3: Zadání číslo 2.



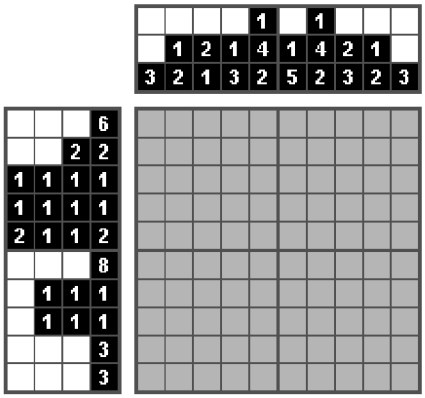
Obrázek C.4: Řešení číslo 2.



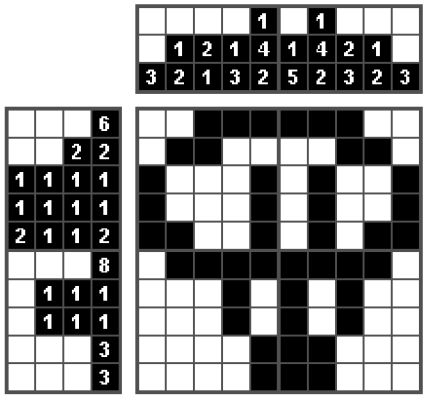
Obrázek C.5: Zadání číslo 3.



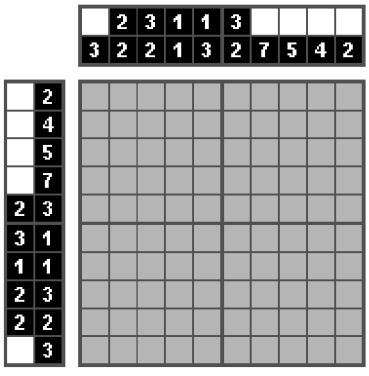
Obrázek C.6: Řešení číslo 3.



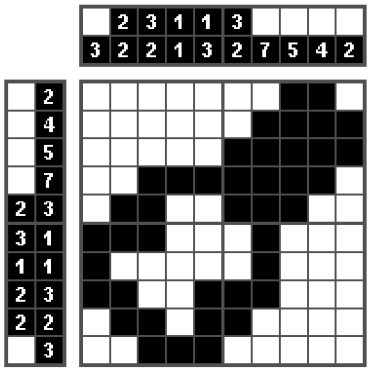
Obrázek C.7: Zadání číslo 4.



Obrázek C.8: Řešení číslo 4.



Obrázek C.9: Zadání číslo 5.



Obrázek C.10: Řešení číslo 5.

Dodatek D

Ukázka různých variant hry Griddlers

V následujících odstavcích jsou popsány další varianty hry Griddlers a Triddlers s jednoduchým popisem způsobu luštění. Pro následující varianty hry Griddlers a Triddlers platí stejná pravidla řešení jako u základní verze hry Griddlers.

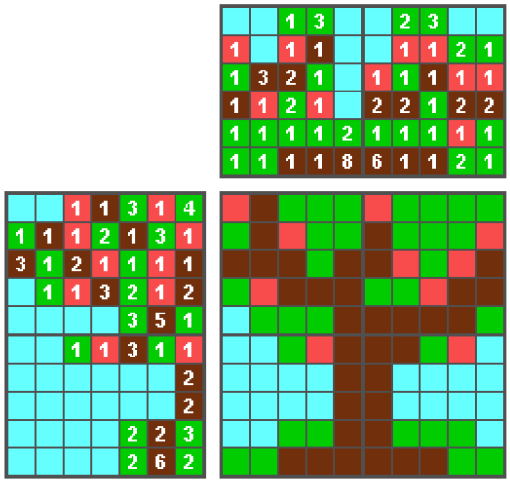
Obrázky D.1 a D.2 ukazují rozšíření základní verze hry Griddlers z černobílé varianty na variantu s různými barvami. Způsob luštění je podobný základní verzi s tím rozdílem, že skupiny buněk se stejnou barvou popředí musí být odděleny minimálně jednou buňkou s barvou pozadí. Skupiny buněk s rozdílnými barvami popředí mohou ležet vedle sebe bez oddělující buňky s barvou pozadí.

Obrázky D.3 a D.4 ukazují rozšíření základní verze hry Griddlers s černobílými buňkami na variantu s různými barvami buněk a barevnými trojúhelníkovými buňkami. Způsob luštění je podobný základní verzi s tím rozdílem, že skupiny buněk se stejnou barvou popředí musí být odděleny minimálně jednou buňkou s barvou pozadí. Skupiny buněk s rozdílnými barvami popředí mohou ležet vedle sebe bez oddělující buňky s barvou pozadí. Buňky obsahující trojúhelníky nemusí být odděleny od ostatních skupin buněk s barvou popředí prázdnou buňkou s barvou pozadí.

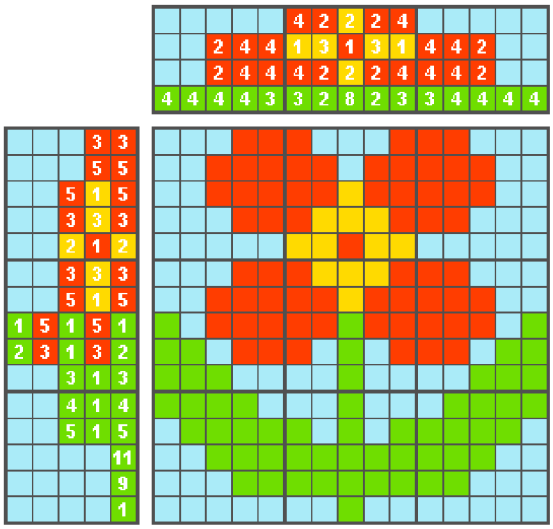
Obrázky D.5 a D.6 ukazují rozšíření základní černobílé verze hry Griddlers o buňky obsahující trojúhelníky. Způsob luštění je podobný základní verzi s tím rozdílem, že buňky obsahující trojúhelníky nemusí být odděleny od ostatních skupin buněk s barvou popředí prázdnou buňkou s barvou pozadí.

Obrázky D.7 a D.8 ukazují základní variantu hry Triddlers. Hra Triddlers je velmi podobná ve způsobu luštění hře Griddlers. Obrázky ukazují základní černobílou variantu této hry. Místo čtvercových buněk se používají trojúhelníkové buňky.

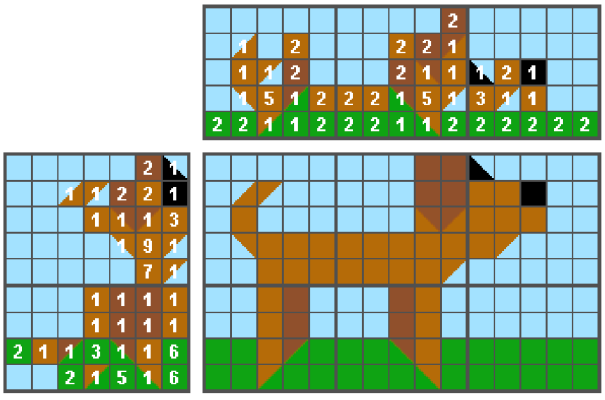
Obrázky D.9 a D.10 ukazují rozšířenou variantu hry Triddlers. Rozšířená varianta hry Triddlers má buňky s více barvami. Způsob luštění je podobný základní verzi s tím rozdílem, že skupiny buněk se stejnou barvou popředí musí být odděleny minimálně jednou buňkou s barvou pozadí. Skupiny buněk s rozdílnými barvami popředí mohou ležet vedle sebe bez oddělující buňky s barvou pozadí. Místo čtvercových buněk se používají trojúhelníkové buňky.



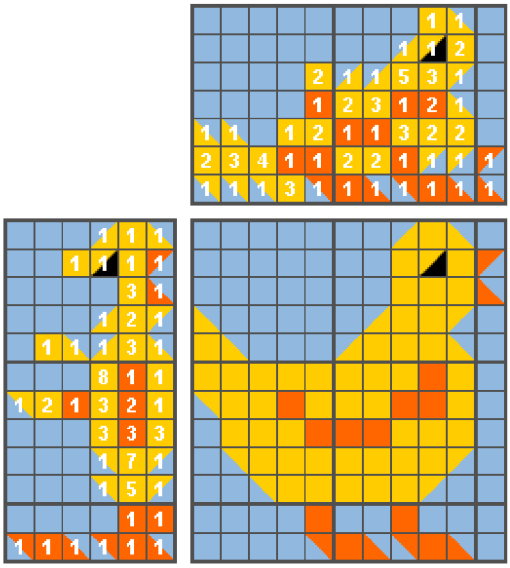
Obrázek D.1: Příklad s barevným obrázkem stromu.



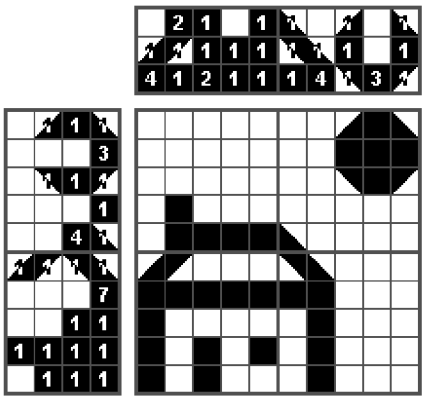
Obrázek D.2: Příklad s barevným obrázkem květiny.



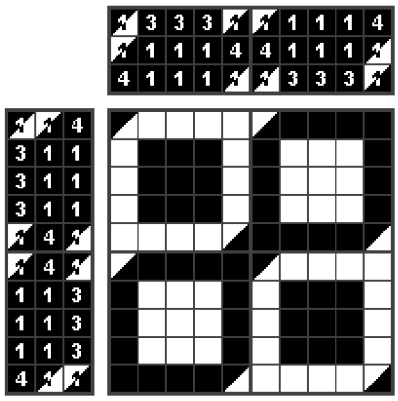
Obrázek D.3: Příklad s barevným obrázkem psa.



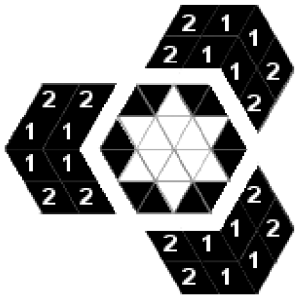
Obrázek D.4: Příklad s barevným obrázkem kuřete.



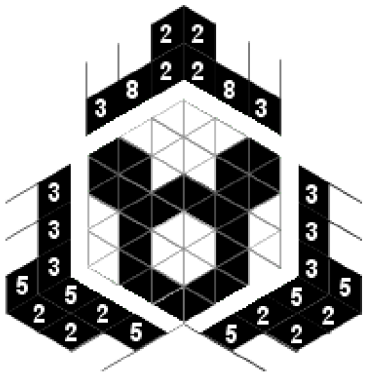
Obrázek D.5: Příklad s obrázkem vesnice.



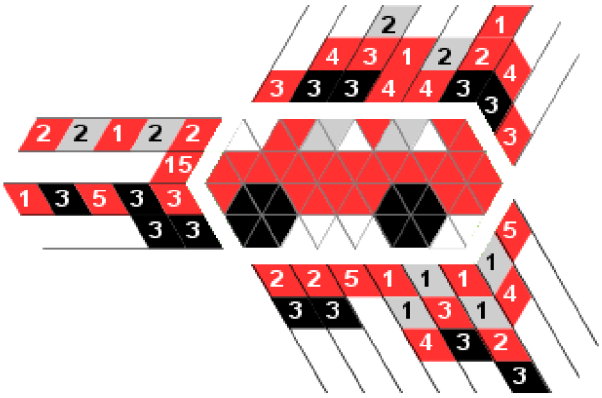
Obrázek D.6: Příklad s abstraktím obrázkem.



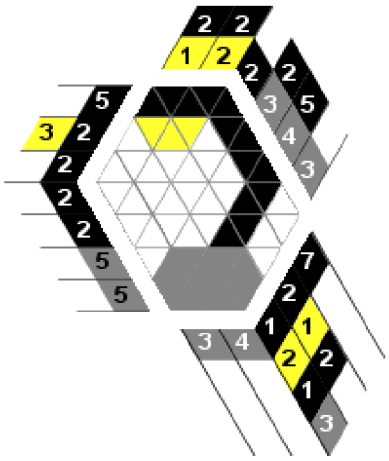
Obrázek D.7: Příklad s obrázkem hvězdy.



Obrázek D.8: Příklad s obrázkem Mickey Mouse.



Obrázek D.9: Příklad s barevným obrázkem auta.



Obrázek D.10: Příklad s barevným obrázkem lampy.