FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

KIV/UIR - Semestrální práce

Automatický generátor labyrintů

Petr Laštovka
A15B0055K
jokertwo@students.zcu.cz

Obsah

1 Zadání **2**

2 Analýza **3**

2.1 Druhy algoritmů 3

2.2 Reprezentace bludiště 3

2.3 Popis řešení 4

2.3.1 Jednoduchý algoritmus 4

2.3.2 DFS algoritmus 7

3 Uživatelská Příručka **9**

3.1 Spuštění programu 9

3.2 Ovládání programu 9

3.2.1 Generování bludiště 9

3.2.2 Řešení bludiště 10

3.2.3 Export 10

3.2.4 Test 10

4 Závěr **11**

Kapitola 1

Zadání

Napiše program pro automatické generování labyrintů různých velikostí. Výstup umožněte do textového souboru (cesta = mezera, zeď = znak volitelný jako parametr) a do souboru grafického formátu (PNG, GIF, PCX, EPS nebo PS). Zároveň bude program umět předvést řešení úlohy - nalezení cesty od vchodu do labyrintu k východu z labyrintu resp. zevnitř labyrintu ven.

Kapitola 2

Analýza

2.1 Druhy algoritmů

Pro generování labyrintů exituje několik druhů algoritmů a rozdělují se podle toho jak se dané bludiště vytváří. Existují 3 základní druhy vytváření bludišt.

- Prvním způsobem je vytváření bludiště postupným přidáváním zdí. Takovým algoritmům se říká *zedníci*.
- Druhý způsob je prakticky opačný k prvnímu, kdy generování spočívá v postupném odebírání zdí. Algoritmy lze poté nazvat *horníky*.
- Posledním typem jsou algoritmy které vytvářejí bludiště na základě šablon.

2.2 Reprezentace bludiště

Teorie grafů popisuje graf $G = (V, E)$ jako matematickou strukturu tvořenou dvěma konečnými množinami V a E, kde prvky množiny V jsou nazvány uzly grafu a E hranami grafu. Každá hrana je tvořena množinou obsahující jeden nebo dva uzly, které se nazývají koncové body.

Každé bludistě se skládá křižovatek, slepých konců a cest, které celé bludiště propojují. Této definice lze využít pro to, aby bylo možné reprezentovat bludiště jako graf. Lze tedy cestám v bludišti přiřadit hrany grafu, stejně tak křižovatky v grafu budou reprezentovány uzly z nichž vychází více hran. A naopak slepá cesta bude reprezentována koncovým uzlem grafu.

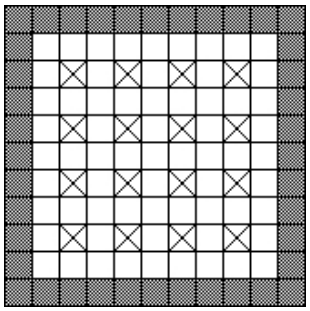
Pokud mohu labyrint reprezentovat jako graf nabízí se pro jeho generování případně i nalezení cesty mezi dvěma body uvnitř labyrintu některý z grafových algoritmů. Grafové algoritmy jako například *DFS (Deep First Search)* má tu vlastnost, že při prohledávání grafu vytváří strom. Této vlastnosti lze u bludišt také využít. Perfektní bludiště má mezi dvěma libovolnými body vždy jen jednu cestu. Toto tvrzení odpovídá definici stromu ve které se říká, že strom je spojitý graf bez smyček. Což znamená, že mezi dvěma libovolnými uzly stromu existuje právě jedna cesta.

2.3 Popis řešení

Pro řešení problému generování bludiště jsem si vybral dva algoritmy. Oba dva pro reprezentaci grafu používají dvourozměrné pole (matici). První algoritmus je založen na postupném procházení celé matice. Druhý pak na rekurzi. Dále bylo nutné pro pozdější práci s bludištěm, aby ob algoritmy používali stejný datový typ. Z mnoha praktických důvodů(označení stěn, cesty, „začátku a konce“ bludiště...) jsem zvolil datový typ *Integer*.

2.3.1 Jednoduchý algoritmus

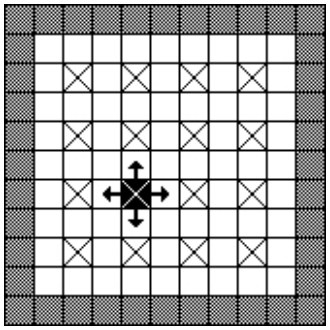
Tento algoritmus si na začatku vygeneruje základ bludiště. Kdy kolem dokola ohraničí prostor zdí a uvnitř vygeneruje počáteční body, odkud se později bude začínat stavět zeď. Základ bludište je na obrázku 2.1. Každý prvek 2D pole představuje jedno políčko. Šedivé políčko představuje *zeď*. Políčko s křížkem *základ* (imaginární, využije se jen při generování bludiště). *Volné políčko* představující cestu v bludišti.



Obrázek 2.1: Počáteční nastavení bludiště

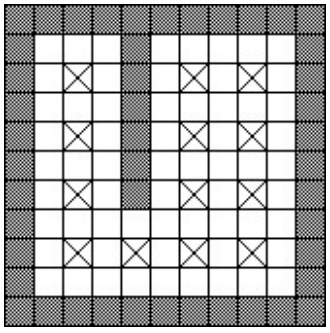
Dále je třeba nadefinovat funkci která bude počítat počet zbývajících políček označených jako *základ*. Například procházením celé matice při použití dvou *for* cyklů.

Samotné generování probíha v několika jednoduchých opakujících se krocích. Náhodně vybereme jedno základové políčko a směr kudy budeme stavět.



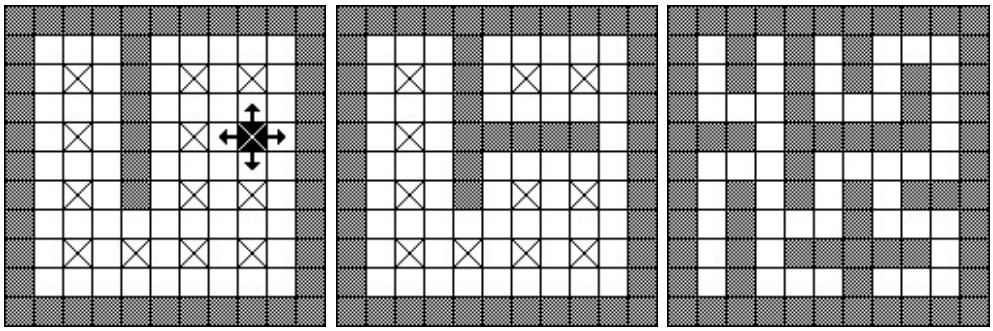
Obrázek 2.2: Je označeno zakladové políčko a znázorněny směry kterými lze stavět

Po zvlolení směru stavíme zeď. A to tak že všechna volná nebo základová políčka, na která po cestě narazíme, mění na zeď. A to tak dlouho dokud nenarazíme na zeď. Proto musí být na začátku pole ohraničené.



Obrázek 2.3: První postavená zeď

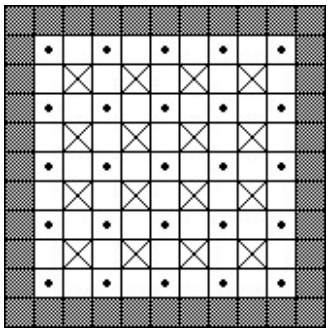
Odtud se opět vrátíme k výběru náhodného základu a směru. Zdi stavíme do té doby dokud je k dispozici alepsoň jeden základ.



Obrázek 2.4: Generování bludiště

Takto vygenerované bludiště má několik vlastností.

- Mezi dvěma libovolnými políčky existuje vždy právě jedna cesta. Je to způsobeno tím, že všechny zdi jsou navzájem spojené. Nevyskytují se zádné osamocené zdi volně plující v prostoru.
- Políčka označená tečkou na obrázku 2.5, budou vždy volná a bude existovat mezi nimi právě jedna cesta. Jsou tedy idealním potecionálním začátkem a koncem bludiště



Obrázek 2.5: Volná pole

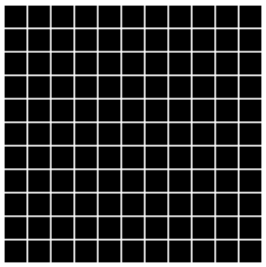
Takto generované bludiště má jednu zásadní nevýhodu. A to právě v potřebě neustále procházet celou matici z důvodu počítání zbývajících *základů*. Tento algoritmus má potom exponenciální složitost a při generování větších labirintů začíná být časová náročnost neúnosná.

2.3.2 DFS algoritmus

Prohledávaní do hloubky (v angličtině označované jako depth-fisrt search nebo zkratkou DFS) je grafový algoritmus pro procházení grafů metodou backtrackingu. Pracuje tak, že vždy expanduje prvního následníka každého vrcholu, pokud jej ještě nenavštívil. Pokud narazí na vrchol, z nějž už nelze dále pokračovat (nemá žádné následníky nebo byli všichni navštíveni), vrací se zpět backtrackingem. [1]

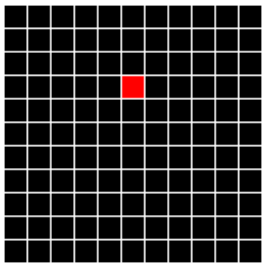
Před generováním samotného bludiště je třeba vytvořit dvourozměrné pole s lichým počtem sloupců a rádků. Toho docílím tak,že velikost zadanou uživatelem a dosadím do vzorce $a * 2 + 1$. Výsledek bude vždy liché číslo a vytvořím tak čtvercovou matici s lichým počtem řádků a sloupců.

V dalším kroku označím všechny prvky matice jako zdi. Obrázek 2.6



Obrázek 2.6: Pole kde všechny buňky představují zeď.

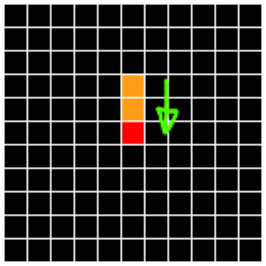
Následně vyberu začátek odkud začnu kopat cestu. Na obrázku 2.7 označeno červenou barvou.



Obrázek 2.7: Pole s označeným začátkem

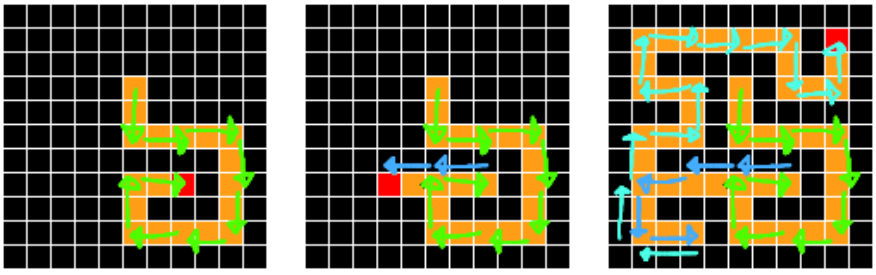
Dále vyberu náhodně jeden směr (nahoru, doprava, doleva, dolů), kterým se budu pohybovat. Vždy se pohnu o dvě buňky jak je ukázano na obrázku 2.8. Při kopání cesty je třeba ohlídat několik věcí, které buď umožní vykopat cestu daným směrem nebo ne.

- Musím zkontrolovat jestli v daném směru dvou buněk neopustím pole.
- Pokud je ve směru dvou buněk pouze zeď mohu vykopat cestu.
- Pokud je ve směru dvou buněk cesta musím se vrátit a vydat se jiným směrem.



Obrázek 2.8: Pole s první vykopanou cestou

Po určité době kopání se dostaneme do slepé uličky jak je znázorněno na obrázku 2.9a. Poté se pomocí backtrakingu vracím zpět až najdu směr kudy mohu pokračovat dále v kopání cesty 2.9b. Vysledné bluiště společně s naznačením jak bylo kopáno je na obrázku 2.9c.



(a) Slepá cesta

(b) Backtraking k nalezení možného směru

(c) Výsledné bludiště

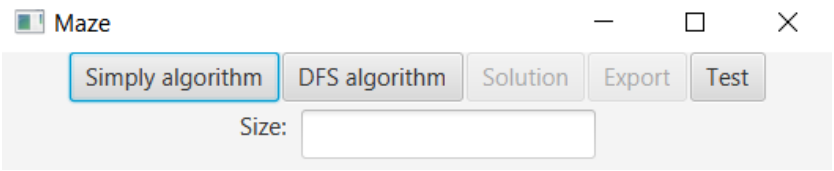
Obrázek 2.9: Postup při generování bludiště

Kapitola 3

Uživatelská Příručka

3.1 Spuštění programu

Program se spouští pomocí spustitelného souboru *SP_A15B0055K.jar*. Pro bezproblémové spustění programu je nutné mít nainstalovanou JRE (Java Runtime Environment)[2]. Po spuštění programu se objeví okno na obrázku 3.1.



Obrázek 3.1: První spuštění.

3.2 Ovládání programu

3.2.1 Generování bludiště

Pro generování bludiště jsou v programu dvě tlačítka. Tlačítko zvýrazněné na obrázku 3.1 generuje labyrint pomocí jednoduchého algoritmu. Tlačítko *DFS algorithm* generuje labyrint pomocí DFS algoritmu.

Vygenerování bludiště je možné až po zadání požadované velikosti labyrintu. Velikost se zadáva do pole *Size* pod tlačítky. Pro velikost je tu jisté

omezení. Nelze zadat menší číslo než 2. Horní hranice je 42. Horní hranice je zvolena pouze z důvodu grafického zobrazení. Větší labyrint je už příliš velký a nevejde se na celou obrazovku.

3.2.2 Řešení bludiště

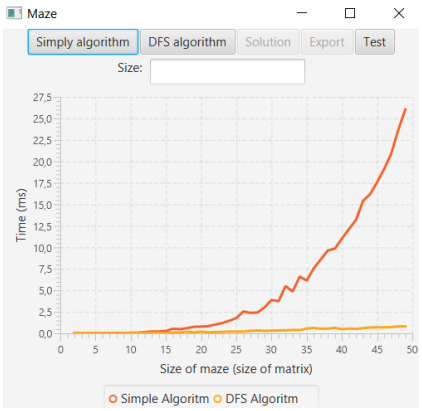
Pro řešení bludiště slouží tlačítko *Solution*. Je to třetí tlačítko zleva na obrázku 3.1. Na tlačítko nelze kliknout dokud není vygenerován alepoň jeden labyrint.

3.2.3 Export

Pro export vygenerovaného bludiště do textového souboru nebo několika grafických formátů slouží tlačítko *Export*. Po jeho stisknutí se objeví volba jak má být bludiště exportováno.

3.2.4 Test

Tlačítko *Test* slouží pro otestování obou algoritmů na konkrétním počítači. Po jeho stisknutí se spustí test obou algoritmů. Test probíhá tak že se postupně zvětšuje velikost generovaného bludiště a meří se čas k tomu potřebný. Navíc pro eliminaci krajních hodnot se pro každou velikost generování opakuje a nasledně dělí jako aritmetický průměr. Výsledkem je graf jako na obrázku 3.2



Obrázek 3.2: Test algoritmů.

Kapitola 4

Závěr

Aplikace převyšuje rozsah původního zadání ale není v rozporu s původním zadání. Důvod proč jsem se rozhodl vytvořit GUI aplikaci bylo jak z důvodu vhodného procvičení této techniky, tak vhodnosti konkrétní úlohy. Bludiště je reprezentováno maticí a je tedy velmi snadné ji vykreslit a i pro mne během ladění aplikace bylo lepší mít možnost vidět konkrétní vygenerované bludiště v okně než v konzoli. V současné podobě je do aplikace velmi snadné doimplementovat další způsoby generování bludišt pokud bude zachváno to, že výstupem bude opět dvourozměrné pole.

Další vhodné rozšíření aplikace by byl multithreading (použití více vláken). Protože pokud je zvýšen limit pro maximální velikost bludiště, tak doba pro generování bludiště velmi roste a aplikace po tu dobu zamrzá. Obzvláště během testování.

Závěrem bych chtěl říci, že práce na této semestrální práci mě velmi bavila a byla pro mne velkým přínosem.

Literatura

[1] Encyklopedie Wikipedia: *Prohledávání do hloubky*. WWW dokument dostupny na URL `https://cs.wikipedia.org/wiki/Prohledávání_do_hloubky`

[2] Java Runtime Environment. `https://www.java.com/en/download/`