FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Tvorba a řešení bludišť

Dominik Zappe

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Tvorba a řešení bludišť

Dominik Zappe

Vedoucí práce
Prof. Dr. Ing. Ivana Kolingerová

PLZEŇ **2023**

Citace v seznamu literatury:
ZAPPE, Dominik. *Tvorba a řešení bludišť*. Plzeň, 2023. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Prof. Dr. Ing. Ivana Kolingerová.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik ZAPPE**
Osobní číslo:	**A20B0279P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Tvorba a řešení bludišť**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s algoritmy tvorby různých druhů bludišť a jejich řešení.
2. Navrhněte pro alespoň dva vybrané druhy bludišť vhodné algoritmy tvorby a průchodu a implementujte.
3. Implementované řešení opatřete vhodným uživatelským rozhraním.
4. Ověřte na sadě netriviálních případů.
5. Zhodnoťte dosažené výsledky.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Prof. Dr. Ing. Ivana Kolingerová**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **3. října 2022**
Termín odevzdání bakalářské práce: **4. května 2023**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2022

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Plzni dne 23. dubna 2023

. . . . . .
Dominik Zappe

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Tato práce se zaměřuje na vytváření dynamických bludišť, tj. bludišť která se mohou měnit v čase. Dynamická bludiště je možné reprezentovat pomocí buněčných automatů, jedná se ale o nepříliš prozkoumané téma. Dosavadní generátory bludišť, které využívají buněčné automaty a matice, jsou omezené a nepřinášejí spolehlivé výsledky. Tato práce navrhuje nový způsob reprezentace buněčných automatů pomocí grafů a nabízí nové kombinace pravidel a grafů, které vedou k lepším výsledkům. Takto vygenerovaná bludiště jsou zajímavější z pohledu řešení a některé z generujících kombinací navíc byly v práci ověřeny jako spolehlivé a rychlé.

Abstract

This work focuses on creating dynamic mazes, i.e., the mazes that can change over time. Dynamic mazes can be represented using cellular automata, but this is an area that has not been extensively researched. Existing maze generators that use cellular automata and matrices are limited and do not produce reliable results. This work proposes a new method of representing cellular automata using graphs and also offers new combinations of rules and graphs that lead to better results. The mazes generated using these methods are more interesting from the perspective of solvability, and some of the generating combinations have been experimentally verified as reliable and fast in this work.

Klíčová slova

bludiště • generování bludišť • řešení bludišť • bludiště reprezentované grafem • v čase proměnlivá bludiště • buněčný automat reprezentovaný pomocí grafů

Poděkování

Chtěl bych poděkovat Prof. Dr. Ing. Ivaně Kolingerové, své vedoucí práce, za její odborné rady, cenné nápady a ochotu věnovat mi svůj čas a energii. Bez její podpory by tato práce nebyla možná.

Rád bych také poděkoval své rodině a přátelům za jejich podporu, povzbuzení a laskavá slova během mého studia.

Obsah

Úvod 1

Bludiště jsou známým hlavolamem na papíře, vytisknuté bludiště se těžko může měnit. Nač se však omezovat na statická, v čase konstantní bludiště? Tato práce prohlubuje znalosti o neobvyklém druhu bludišť – v čase proměnlivá, dynamická bludiště. Tato bludiště by bylo vhodné reprezentovat např. buněčnými automaty, avšak generování bludišť pomocí buněčných automatů není příliš prozkoumané téma, navíc buněčné automaty se téměř vždy reprezentují maticí. Známé jsou pouze dva řetězce pravidel buněčných automatů, které vedou k vygenerování bludiště. Nejedná se však o spolehlivé generátory bludišť – výsledná bludiště jsou často nespojitá a neřešitelná. Reprezentace buněčného automatu maticí navíc limituje možné kombinace. Tato práce navrhuje řešení pro reprezentaci buněčného automatu pomocí grafů. Navíc také uvádí nové kombinace pravidel a grafů, které vedou ke spolehlivějším generátorům bludišť. Experimentálně byla měřena spolehlivost a rychlost zkoumaných kombinací. Výsledná nově navržená bludiště mohou být zajímavější z pohledu řešitele než doposud známé dynamické kombinace.

Cílem práce je seznámit se s algoritmy pro tvorbu a řešení různých druhů bludišť, dále pro vybrané druhy bludišť navrhnout a implementovat vhodné algoritmy generování a průchodu. Následně je cílem opatřit aplikaci vhodným uživatelským rozhraním.

1.1 Struktura práce

Tato úvodní kapitola má za účel objasnit základní koncepty ohledně bludišť spolu s jejich historií a kategorizací. Druhá kapitola je věnována již existujícím metodám reprezentace, generování a řešení bludišť. Kapitola se rovněž věnuje popisu buněčných automatů. Dále ve třetí kapitole budou popsány použité technologie, navržená a implementovaná řešení zmíněných metod a popis vizualizace. Čtvrtá kapitola se věnuje popisu experimentů a jejich výsledků. Kapitola se nejprve zabývá

experimenty s generováním bludišť, dále budou popsány i experimenty řešení bludišť. V závěrečné kapitole jsou shrnuty dosažené výsledky v rámci této práce a jsou navržena další vylepšení do budoucna.

1.2 Bludiště

Bludiště je hlavolam s jednou nebo více cestami od startu k cíli. Nejčastější tvar bludiště je obdélníkový s buňkami v mřížce a s vyznačeným začátkem a koncem [12]. Začátek a konec nemusí být jen na okraji bludiště – typicky může jeden z nich být uvnitř. Řešení takových bludišť je složitější, cesta se nedá nalézt pouhým zatáčením do stále stejného směru.

Pojem bludiště se často zaměňuje s termínem labyrint. V labyrintu, na rozdíl od bludiště, neexistuje možnost větvení. Do cíle vede jediná cesta a není možné z ní sejít, neboť labyrint neobsahuje žádné křižovatky. Cíl labyrintu často bývá ve středu. Bludiště naopak mohou mít více cest vedoucích k cíli a obsahují křižovatky. Řešitel labyrintu se tedy nemůže ztratit, kdežto řešitel bludiště může i zabloudit, když si zvolí špatnou cestu na křižovatkách. Takto definovaný labyrint je podmnožinou bludišť.

1.3 Historie bludišť

Bludiště byla používána jako ornamenty na slavnostních rouchách křesťanských vládců již před devátým stoletím a bludišti se zdobily i chrámy a katedrály. Původně bludiště neměla smysl zábavy jako dnes. Záměrem bylo zobrazit složité nástrahy hříchů, kterými je člověk během života obklopen.

Primární účel bludišť se postupně měnil s historií lidstva. Se vzestupem zájmu o vzdělání a poznání se bludiště stala populární zábavou. V souvislosti s popularitou se také bludiště stala středem pozornosti matematiků. S nástupem počítačové éry se lidé začali zajímat o automatizaci všeho – včetně generování a procházení bludišť.

1.4 Druhy bludišť

Walter D. Pullen [12] ve svých pracích uvedl několik kritérií, podle kterých se bludiště dají dělit. Tato práce jeho kategorizaci přebírá. Sedm vybraných kritérií je: *dimenze* (Dimension), *hyperdimenze* (Hyperdimension), *topologie* (Topology), *teselace* (Tessellation), *směrování* (Routing), *textura* (Texture) a *záměr* (Focus). Bludiště

obecně mohou mít vlastnosti odpovídající více kategoriím. Obrázky jednotlivých druhů bludišť se nacházejí v Příloze B.

Dimenze. *Dimenzí* se rozumí obecná dimenze prostoru, ve kterém dané bludiště existuje [12]. Typickým příkladem je 2D a 3D bludiště. Dimenzím však není kladen žádný limit, bludiště proto mohou být klidně i vícedimenzionální. Typickým příkladem mohou být 4D bludiště, která se dají vykreslit jako 3D bludiště s portály do jiného času – čtvrtá dimenze se dá chápat jako čas. 2D bludiště je vidět na Obrázku B.1, 3D bludiště je možné si prohlédnout na Obrázku B.2.

Hyperdimenze. Kategorie *hyperdimenze* se soustředí na dimenzi objektu, který se pohybuje bludištěm [12]. Nezabývá se tedy dimenzí bludiště samotnou. *Ne–hyperbludiště* (Non–hypermaze) je klasické bludiště, kde se pohybuje bod. Většina bludišť je tohoto typu. Naproti tomu existují i *hyperbludiště* (Hypermaze) – v nich se nepohybuje pouze bod, ale úsečka nebo těleso. Pohyb je zde značně komplikovanější a je potřeba brát v potaz možné rotace a ohyby pohybujícího se objektu.

Topologie. *Topologie* popisuje prostor, ve kterém dané bludiště existuje jako celek [12]. Klasickým příkladem je *Euklidovský prostor*. Oproti tomu může bludiště existovat například na *Möbiově pásce*, být vytvořené užitím *hyperbolické geometrie* nebo jiných *neeuklidovských geometrií*. Bludiště vytvořené v *neeuklidovském prostoru* je vyobrazeno na Obrázku B.3.

Teselace. *Teselací* se rozumí tvar jednotlivých buněk, které dohromady tvoří dané bludiště [12]. Nejběžnější *teselací* je *ortogonální mřížka* – buňky mají tvar čtverců nebo krychlí a jsou ekvidistantní, viz Obrázek B.1. Další klasickou možností je *delta*, respektive *sigma teselace* – jedná se o rovnostranné trojúhelníky, respektive hexagony, viz Obrázek B.4 a Obrázek B.5. Jinou možnou teselací může být kombinace oktagonů a čtverců – *upsilon* teselace, viz Obrázek B.6. Hojně užívanou teselací je *theta* – bludiště má tvar soustředných kružnic a buňky jsou výseče mezikruží, viz Obrázek B.7. Atypickou teselací je tzv. *crack* bludiště – jedná se o bludiště, která nemá žádnou teselaci, tedy jednotlivé buňky nemají pevně daný tvar – spíše se tedy jedná o zdi spojené pod různými úhly. Další atypickou teselací je *fraktál* – každá buňka bludiště je stejným, menším bludištěm sama v sobě. Crack a fraktálovou teselaci je možné si prohlédnout na Obrázku B.8 a Obrázku B.9.

Směrování. Kategorie *směrování* odkazuje na typy cest s ohledem na geometrii a výše zmíněné kategorie. Typickými příklady jsou *dokonalé bludiště* (Perfect Maze),

spletené bludiště (Braid Maze), *jednocestné bludiště* (Unicursal Maze) a *řídké bludiště* (Sparse Maze) [12]. *Dokonalé bludiště* je takové bludiště, ve kterém existuje právě jedna cesta z jakékoliv buňky do jakékoliv jiné. To znamená, že toto bludiště neobsahuje žádné smyčky a izolované oblasti. Naopak obsahuje hodně slepých uliček. Dokonalé bludiště je vykreslené na Obrázku B.1. *Spletené bludiště* je takové bludiště, které neobsahuje žádnou slepou uličku a žádné izolované části. Typicky se v tomto bludišti řešitel točí v kruzích. Spletené bludiště je vidět na Obrázku B.10. *Jednocestné bludiště* se též nazývá labyrint. Jedná se o bludiště, které nemá žádné křižovatky, a tudíž obsahuje pouze jednu cestu, viz Obrázek B.11. *Řídké bludiště* je takové bludiště, která při vygenerování nevyužilo všechny dostupné buňky. Mohou tak vznikat izolované oblasti mimo hlavní cestu. Navíc mohou vznikat větší místnosti, které nebudou velikosti pouze jedné buňky. Řídké bludiště je vyobrazeno na Obrázku B.12.

Textura. *Texturou* bludiště se rozumí styl cest v libovolném *směrování* a geometrii. Stylem nejsou myšleny binární příznaky, ale obecná témata [12]. Příklad *textury* může být *bias* – ten určuje poměr počtu horizontálních zdí ku počtu vertikálních zdí. Bludiště s horizontálním biasem má dlouhé cesty zleva doprava a jen krátké průchody nahoru a dolů, viz Obrázek B.13, opačně vertikální bias. Další *texturou* je *run*. Čím větší tento faktor je, tím více bludiště obsahuje dlouhé rovné cesty, viz Obrázek B.14. Bludiště s nízkým faktorem *run* vypadají náhodněji a obsahují více křižovatek a zatáček. *Elitismus* je dalším měřeným faktorem. Jedná se o poměr cest, kterými je potřeba jít pro vyřešení bludiště, ku počtu všech cest. Pokud má bludiště tento faktor vysoký, je řešení takového bludiště často krátké a přímočaré. Pokud je tento faktor nízký, řešení často zahrnuje velkou část bludiště. *Symetrie* je další klasickou *texturou*. Může být jak osová, tak středová. Středově symetrické bludiště je na Obrázku B.15.

Záměr. Poslední kategorie *záměru* (Focus) je obskurní, prakticky pouze rozděluje algoritmy pro generování bludišť do dvou kategorií podle jejich záměru – *stavitelé* (Wall Adders) a *tesaři* (Passage Carvers), [12]. *Stavitelský* algoritmus začíná s velkou místností a postupně přidává jednotlivé zdi. *Tesařský* algoritmus začíná s plně zazděnou místností a postupně ubírá jednotlivé zdi.

Existující řešení 2

2.1 Reprezentace bludišť

V této práci je zvolena reprezentace bludišť pomocí grafů, protože se jedná o obecnější řešení, než reprezentace maticí.

Matice

Jedná se o nejjednodušší možnost reprezentace bludišť, ale značně limituje *teselaci*. Reprezentace maticí je vhodná pro bludiště s *ortogonální mřížkovou teselací*.

Graf

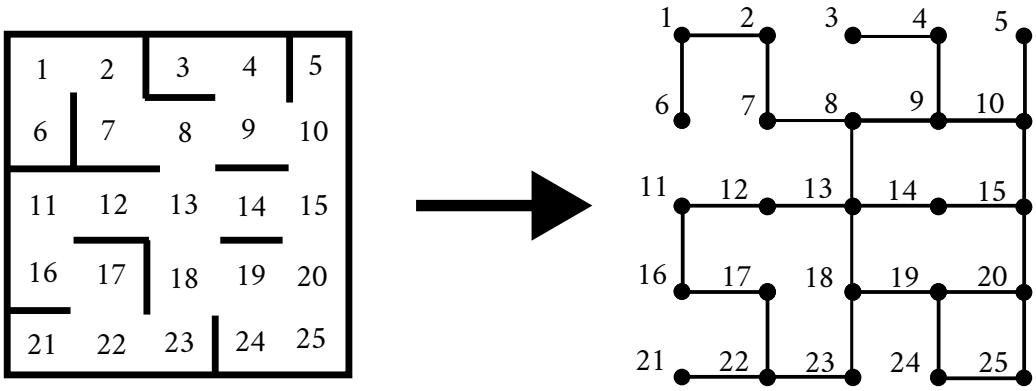
Graf je trojice $G = (V, E, \epsilon)$, kde V je neprázdná konečná množina vrcholů, E je konečná množina hran a ϵ je zobrazení nazývané *vztahem incidence*. Zobrazení ϵ přiřazuje každé hraně $e \in E$ uspořádanou dvojici vrcholů (x, y). Vrchol x se označuje jako *počáteční vrchol hrany* a vrchol y se nazývá *koncový vrchol hrany* [4, 6]. Hrany mohou být také *neorientované*, lze je chápat jako symetrické spojení dvou vrcholů – každý z vrcholů je počátečním i koncovým. Navíc lze hranám přiřadit číselné ohodnocení (nejčastěji reálná čísla), vzniká tak *hranově ohodnocený graf*.

Graf G' nazýváme *faktorem* grafu G, pokud vznikne z grafu G pouhým vynecháním některých hran [4]. K vynechání hrany ani dojít nemusí – platí, že každý graf je zároveň i svým faktorem. Platí tedy, že:
$V(G) = V(G') \wedge E(G') \subseteq E(G)$.

Strom je graf, který neobsahuje žádné *kružnice* – smyčky [4]. Navíc tento graf musí být *souvislý*. Z toho vyplývá, že z každého vrcholu vede do jakéhokoliv jiného vrcholu právě jedna cesta.

Kostra grafu je faktor, který je navíc stromem. Platí, že každý souvislý graf má kostru [4].

Bludiště se dá chápat jako graf, viz Obrázek 2.1. Bludiště v mřížce je zobrazení několika buněk, které jsou spojeny cestami. Na buňky se tak dá nahlížet jako na vrcholy grafu a cesty mezi buňkami je možné chápat jako hrany grafu. Jinými slovy – když mezi buňkami v bludišti není zeď, je mezi odpovídajícími vrcholy v grafu hrana.



Obrázek 2.1: Bludiště reprezentované jako graf

Např. výše definované *dokonalé bludiště* je reprezentováno *kostrou grafu*, protože dokonalá bludiště neobsahují žádné smyčky a z každé buňky existuje právě jedna cesta do každé jiné buňky. Obdobně kostra grafu neobsahuje žádné kružnice a z každého vrcholu existuje právě jedna cesta do každého jiného vrcholu.

Díky této podobnosti s grafy lze pro generování a řešení bludišť využít grafové algoritmy.

2.2 Buněčné automaty

Buněčné automaty jsou matematické modely, které popisují chování jednoduchých buněk v jednorozměrném, dvourozměrném nebo vícerozměrném prostoru. Tyto modely jsou založeny na množině pravidel, která řídí, jak se chovají jednotlivé buňky na základě stavu svých sousedů [1]. Nejčastěji je model reprezentován dvoudimenzionálním polem se sousedností v 8–okolí.

Každá buňka v buněčném automatu může být v jednom ze stavů, například „živá“ nebo „mrtvá“. Pravidla popisují, jak se stavy buněk mohou měnit v závislosti na stavech jejich sousedů. Tyto pravidla se aplikují na každou buňku v systému současně, což způsobuje změny stavů v čase [1].

Nejznámější příklad buněčného automatu je *Hra života* (Game of Life) od Johna Conwaye, která se hraje na dvourozměrné mřížce. V této hře je každá buňka buď živá nebo mrtvá. Pravidla určují, že každá buňka přežije, pokud má právě dva nebo tři sousedy, jinak zemře. Pokud je mrtvá buňka obklopena přesně třemi živými buňkami, stává se živou buňkou. Tyto pravidla se aplikují na každou buňku v každé iteraci hry [1].

Buněčné automaty mají mnoho aplikací, včetně modelování fyzikálních procesů, studia biologických systémů a výzkumu umělé inteligence.

2.3 Generování bludišť

Existuje mnoho algoritmů pro generování bludišť, obvykle jsou řešení spjatá s konkrétní reprezentací. Vzhledem ke zvolené grafové reprezentaci budou prezentovány převážně grafové algoritmy.

S generováním bludiště souvisí poslední kategorie rozdělení bludišť – *záměr* (Focus), viz podkapitola 1.4. Obecně se dají stejná bludiště vygenerovat oběma *záměry*.

Dále popisované algoritmy pro generování bludišť nekladou žádná omezení na druhy bludišť. Řešení v podkapitole 2.3.7 předpokládá reprezentaci maticí, ostatní algoritmy využívají reprezentaci grafem.

Detailněji budou popsány algoritmy, které jsou využité v implementaci aplikace. K detailnímu popisu u implementovaných algoritmů bude vždy uveden pseudokód.

2.3.1 Prohledávání do hloubky

Algoritmus je také známý jako *rekurzivní backtracker*. Jedná se o randomizovanou verzi klasického prohledávání do hloubky (DFS). Hlavní myšlenkou tohoto algoritmu je začít se zaplněným bludištěm (prázdný graf). Počáteční aktuální buňka se zvolí náhodně a následně se jako aktuální buňka zvolí náhodný soused, který ještě nebyl navštíven. Takto se volí, dokud má aktuální buňka nenavštívené sousedy. Pokud už jsou všichni sousedé navštíveni, nastane návratová fáze a algoritmus se vrací k předchozí buňce tolikrát, dokud nenajde vhodnou, tj. takovou, která má nenavštívené sousedy. Algoritmus takto postupně přidává hrany do grafu, dokud není bludiště kompletně vygenerované, tj. dokud není každá buňka navštívena [2, 5]. Způsob implementace popsaného algoritmu je nastíněn Pseudokódem 2.1 a 2.2.

Pokud je algoritmus implementován s užitím rekurze, nikoliv zásobníku, může dojít k přetečení paměťového zásobníku.

Bludiště vygenerovaná tímto způsobem jsou dokonalá – neobsahují žádné smyčky. Z důvodu charakteru algoritmu vygenerovaná bludiště často obsahují dlouhé chodby a obecně neobsahují příliš křižovatek – prohledávání do hloubky se snaží jít co nejvíce přímočaře a větví se pouze ve fázi návratu z rekurze [12].

Pseudokód 2.1: Prohledávání do hloubky (rekurzivní verze)

```
DFS(G, v) begin /* G je graf, v je vrchol */
    označ v jako navštívený vrchol
    foreach soused in sousedé vrcholu v do begin
        if soused není označen jako navštívený then
            DFS(G, soused)
    end
end
```

Pseudokód 2.2: Prohledávání do hloubky (iterativní verze)

```
Nechť v je zvolený počáteční vrchol grafu
Nechť S je zásobník
S.push(v)
while S není prázdný do begin
    v := S.pop()
    if v není označen jako navštívený then
        označ v jako navštívený vrchol
        foreach soused in sousedé vrcholu v do begin
            S.push(soused)
        end
end
```

2.3.2 Kruskalův algoritmus

Algoritmus je randomizovanou verzí originálního Kruskalova algoritmu pro hledání minimální kostry grafu. Jedná se o výpočetně náročnější algoritmus. Na počátku je opět bludiště zaplněné zdmi – prázdný graf. Každá buňka (vrchol grafu) si vytvoří jednoprvkovou množinu obsahující pouze sebe sama. Následně se vybere náhodná dvojice sousedících vrcholů (buněk), pokud jsou množiny odpovídající jednotlivým vrcholům disjunktní, sjednotí se a do grafu se přidá hrana mezi vrcholy. Algoritmus toto opakuje tak dlouho, dokud nezkusí přidat každou možnou hranu grafu [2, 5]. Popisu algoritmu odpovídá Pseudokód 2.3

Implementace tohoto algoritmu je značně usnadněna použitím datové struktury disjunktních množin – tato datová struktura nabízí všechny potřebné operace pro tento algoritmus. Navíc může pomoci si graf udělat ohodnocený a ohodnocení hran volit náhodně, neboť algoritmus hledá minimální kostru ohodnoceného grafu.

Výsledná bludiště jsou opět dokonalá – neobsahují žádné smyčky – neboť algoritmus se původně používal pro nalezení kostry grafu [12].

Po několika jednoduchých modifikacích tohoto algoritmu je možné vytvářet nedokonalá bludiště, která jdou navíc parametrizovat několika hodnotami – biasy. Tyto biasy ovlivňují výslednou texturu bludiště – jedná se o horizontální a vertikální bias. Navíc lze také přidat cyklický bias určující, jak časté jsou smyčky v bludišti [7]. Pseudokód 2.4 znázorňuje běh tohoto modifikovaného algoritmu.

Pseudokód 2.3: Obyčejný Kruskalův algoritmus

```
Nechť G je graf
Nechť K je prázdná množina /* výsledná kostra */
foreach vrchol in G do begin
    MAKE_SET(vrchol)
end
foreach hrana (u, v) in G, seřazeno vzestupně vahami do begin
    if FIND_SET(u) ≠ FIND_SET(v) then
        K := K ∪ {(u, v)} ∪ {(v, u)}
        UNION(FIND_SET(u), FIND_SET(v))
end

/* funkce MAKE_SET vytvoří jednoprvkovou množinu s daným
   prvkem, funkce FIND_SET hledá kořenový prvek dané množiny
   (množina je reprezentována stromem) a funkce UNION provádí
    sjednocení dvou daných množin v jednu */
```

Pseudokód 2.4: Modifikovaný Kruskalův algoritmus

```
Nechť G je graf
Nechť K je prázdná množina /* výsledný graf */
foreach vrchol in G do begin
    MAKE_SET(vrchol)
end
pridej_nutne_hrany() /* spojí každé dva sousedící sloupce,
   resp.~řádky náhodnou hranou */
foreach hrana (u, v) in G, seřazeno vzestupně vahami do begin
    if FIND_SET(u) ≠ FIND_SET(v) then
        zkontroluj_a_pridej((u, v))
end
foreach hrana (u, v) in G - K do begin
    zkontroluj_a_pridej((u, v))
end

/* funkce zkontroluj_a_pridej(hrana) kontroluje, zda je hrana
    horizontální nebo vertikální a podle počítadel hran
   rozhoduje, zda danou hranu přidá či nikoliv -- přidání
   probíhá sjednocením množin */
```

2.3.3 Prim–Jarníkův algoritmus

Jedná se o randomizovanou verzi algoritmu pro hledání minimální kostry grafu. Algoritmus opět začíná se všemi zdmi postavenými – prázdný graf. Nejprve se vybere náhodná buňka a označí se za část bludiště. Dále se do listu zpracování přidají všichni sousedé této buňky a buňka se označí za navštívenou. Dokud není list zpracování prázdný, prochází se postupně všichni sousedé buněk. Pokud zeď odděluje buňky, které nebyly ještě navštíveny, přidají se do bludiště a zeď mezi nimi se zboří – přidá se v grafu hrana [2, 5].

Výsledná bludiště vypadají stylisticky podobně jako ta vygenerovaná Kruskalovým algoritmem, neboť oba algoritmy hledají minimální kostru grafu [12].

2.3.4 Wilsonův algoritmus

Algoritmus na začátku zvolí náhodné dvě buňky zaplněného grafu a pokusí se z jedné buňky náhodnou chůzí dostat do té druhé. Výsledek je zapisován do prázdného grafu přidáváním relevantních hran. Pokud na své cestě překříží algoritmus sám svou cestu, odebere z tvořené cesty smyčku a pokračuje náhodnou chůzí, dokud nenalezne cílovou buňku. Pokud na své cestě překříží algoritmus již existující jinou cestu, ukončí svojí náhodnou chůzi a přidá dosavadní cestu do celku – bludiště. Takto algoritmus spojuje náhodným výběrem směru všechny buňky, dokud není bludiště hotové [2, 5].

Všechny dosavadní algoritmy mají nějakou formu biasu – bludiště vygenerovaná náhodným průchodem do hloubky mají tendence k tomu mít dlouhé chodby a málo křižovatek, bludiště vygenerovaná Kruskalovým (resp. Prim–Jarníkovým) algoritmem mají tendence mít mnoho krátkých a slepých cest. Wilsonův algoritmus poskytuje bludiště, která nemá sklony k žádné z výše uvedených forem biasu. Generuje rovnoměrně rozdělená bludiště z dané množiny vrcholů (buněk) [12].

2.3.5 Aldous–Broderův algoritmus

Algoritmus využívá obdobné strategie jako Wilsonův algoritmus – náhodná chůze. Na začátku se zvolí náhodná buňka a označí se jako aktuální a navštívená. Dále se náhodně zvolí soused buňky, který ještě nebyl navštíven, a zboří se mezi nimi zeď. Zvolený soused je označen za aktuální buňku a algoritmus pokračuje, dokud nejsou všechny buňky navštíveny [2, 12].

Tento algoritmus opět poskytuje uniformní kostru grafu [5].

2.3.6 Hunt and kill

Algoritmus využívá stejnou strategii jako randomizovaný průchod do hloubky – na počátku se zvolí náhodná buňka, ze které se začne prohledávat do hloubky. Dojde-li algoritmus do slepé uličky, nebude se na rozdíl od průchodu do hloubky vracet z rekurze, ale zvolí si náhodnou novou buňku, ze které opět začne prohledávat do hloubky [2, 5].

Jelikož algoritmus vychází ze strategie průchodu do hloubky, vygenerovaná bludiště jsou opět dokonalá [12].

2.3.7 Buněčný automat

Buněčné automaty byly detailně popsány v podkapitole 2.2.

Určitá pravidla buněčného automatu mohou vést ke generování bludišť. Dva známé buněčné generátory bludišť jsou *Maze* a *Mazectric* [3]. Tyto buněčné automaty jdou popsat řetězcem pravidel – B3/S12345 a B3/S1234. Řetězec pravidel lze jednoduše číst tak, že čísla uvedená za písmenem B říkají, kolik musí buňka mít sousedů, aby se narodila (ožila). Čísla uvedená za písmenem S určují počet sousedů, kolik musí buňka mít, aby přežila. Řetězec pravidel B3/S12345 lze tedy přeložit takto – buňka ožije právě tehdy, když má tři živé sousedy; buňka přežije právě tehdy, když má jednoho, dva, tři, čtyři nebo pět živých sousedů. Při jiné konfiguraci buňka umírá.

2.4 Řešení bludišť

Řešení bludišť nemusí být na rozdíl od generování spjatá s reprezentací. Známý naivní postup pro vyřešení bludiště je držet se u zdi a zatáčet stále do stejného směru, je-li to možné. Tento algoritmus funguje pouze pro dokonalá bludiště, která mají začátek i konec na kraji bludiště [12, 10].

Dále budou popsány algoritmy pro řešení bludišť spojené s grafovou reprezentací, neboť se jedná o reprezentaci zvolenou v této práci.

2.4.1 Prohledávání do šířky

Původně bylo prohledávání do šířky navrženo pro hledání ve stromové struktuře. V podkapitole 2.1, Graf bylo zmíněno, že každý graf má kostru – faktor, který je navíc stromem. Lze tedy prohledávání do šířky užít v obecném grafu.

Algoritmus na začátku zvolí počátek bludiště jako aktuální buňku. Do fronty se uloží všichni sousedé aktuální buňky, kteří ještě nebyli navštíveni. Jako aktuální nová

buňka se zvolí první buňka ve frontě. Algoritmus takto pokračuje, dokud nejsou navštíveny všechny buňky [8]. Díky zapisování předchůdců jednotlivých sousedů při procházení grafu je později možné zpětně rekonstruovat nalezenou nejkratší cestu, viz Pseudokód 2.5.

Spustí-li se algoritmus z počátku bludiště a do fronty se v průběhu hledání dostane buňka koncová, je bludiště řešitelné. Pokud je navíc požadována konstrukce cesty ze začátku bludiště do konce, je zapotřebí si pamatovat konkrétní sousedy, kteří vkládali jednotlivé buňky do fronty. Pomocí tohoto listu předchůdců lze zpětně zkonstruovat celou cestu. Nalezená cesta je nejkratší.

Oproti negrafovým řešením má tento algoritmus hlavní výhodu, že vždy cestu najde za podmínky, že cesta existuje [10].

Pseudokód 2.5: Prohledávání do šířky

```
1  Nechť v je zvolený počáteční vrchol grafu
2  Nechť Q je fronta
3  Nechť P je pole předchůdců /* možnost rekonstrukce nejkratší
       cesty */
4  Q.enqueue(v)
5  P[v] := -1
6  while Q není prázdná do begin
7      v := Q.dequeue()
8      if v není označen jako navštívený then
9          označ v jako navštívený vrchol
10         foreach soused in sousedé vrcholu v do begin
11             Q.enqueue(soused)
12             P[soused] := v
13         end
14 end
```

2.4.2 Dijkstrův algoritmus

Dijkstrův algoritmus je vylepšením prohledávání do šířky o prioritní frontu [13].

Algoritmus si nejprve vytvoří pole předchůdců a vzdáleností. Inicializační hodnoty předchůdců se nastaví na nedefinováno, hodnoty vzdáleností na nekonečno. Jako aktuální buňka se zvolí počátek bludiště a zároveň se jeho vzdálenost nastaví na nulu. Do prioritní fronty se zařadí všichni sousedé aktuální buňky, nastaví se jejich předchůdce na aktuální buňku a jejich vzdálenost se aktualizuje na minimum z jejich dosavadní vzdálenosti a vzdálenosti aktuální buňky plus jedna. Jako nová aktuální buňka se volí první buňka ve frontě, tedy ta nejvýhodnější ve smyslu prioritní fronty. Buňky se v prioritní řadě řadí dle vzdálenosti [13]. Dijkstrův algoritmus je možné

implementovat pomocí A* algoritmu, proto zde není uveden pseudokód pro tento algoritmus.

Algoritmus je opět schopen kontroly řešitelnosti bludiště, tj. odpovědět na otázku, zda je vůbec začátek a konec bludiště spojen nějakou cestu. Pokud cesta existuje, povaha algoritmu zajišťuje, že se jedná o nejkratší cestu. Její zpětná rekonstrukce je jednoduchá díky poli předchůdců.

2.4.3 A* algoritmus

A* algoritmus[1] používá hladový princip pro nalezení optimální cesty, optimální ve smyslu nejkratší. A* algoritmus je vylepšením Dijkstrova algoritmu o heuristiku, která posuzuje, jak daleko je daný uzel grafu od cíle [9, 8].

Algoritmus si na začátku vytvoří pole předchůdců, vzdáleností (označováno jako *g skóre*), heuristických odhadů (označováno jako *h skóre*) a předpokladů délky cest (označováno jako *f skóre*). Počátek se zvolí jako aktuální buňka a jeho g skóre se nastaví na nulu, h skóre se vypočítá pomocí heuristické funkce – viz následující podkapitola 2.4.3, Heuristiky – a f skóre se nastaví na součet g skóre a h skóre. Pro každého ze sousedů aktuální buňky se vyhodnotí jejich g skóre jako součet g skóre aktuální buňky a heuristická vzdálenost daného souseda. Pokud se daný soused nenachází v prioritní frontě, přidá se do ní. Pokud již existující g skóre daného souseda bylo lepší, nestane se nic, neboť už v prioritní frontě se správnými hodnotami je. Pokud je g skóre daného souseda horší než nově spočítané, aktualizují se mu hodnoty na lepší. Následně se aktualizuje i f skóre, opět jako součet g skóre a h skóre. Jako nová aktuální buňka se volí první z fronty, tedy ta nejvýhodnější ve smyslu zmiňovaných skóre. V průběhu algoritmu se navíc zapisují předchůdci jednotlivých buněk [9, 8]. Popisu algoritmus odpovídá Pseudokód 2.6.

Zpětná rekonstrukce cesty je opět snadná díky listu předchůdců. Nalezená cesta je nejkratší, stejně jako u prohledávání do šířky a Dijkstrova algoritmu. Výhodou oproti Dijkstrově algoritmu je, že díky heuristické funkci algoritmus obecně konverguje k cíli rychleji.

[1] Bertram Raphael v roce 1968 dokázal, že algoritmus A2 – vylepšení algoritmu A1, což je vylepšení Dijkstrova algoritmu o heuristiku – je optimální pro konzistentní heuristiku. Nazval ho unixovým stylem A*, tedy A a všechny možné další verze.

Pseudokód 2.6: A* algoritmus

```
Nechť v je zvolený počáteční vrchol grafu
Nechť cíl je cílový vrchol grafu
Nechť Q je prioritní fronta
Nechť P je pole předchůdců /* možnost rekonstrukce nejkratší
    cesty */
Nechť Gs je pole odpovídající g_score, inicializace na ∞
Nechť Fs je pole odpovídající f_score, inicializace na ∞
Q.add(v)
P[v] := -1
Gs[v] := 0
Fs[v] := h(v) /* heuristická funkce */
while Q není prázdná do begin
    v := vrchol v Q s~nejmenší hodnotou Fs
    if v je cíl then
        return
    Q.remove(v)
    foreach soused in sousedé vrcholu v do begin
        skore := Gs[v] + d(v, soused)
        if skore < Gs[soused] then
            P[soused] = v
            Gs[soused] = skore
            Fs[soused] = skore + h(soused)
            if soused není v Q then
                Q.add(soused)
    end
end
```

Heuristiky

Správná volba heuristiky je pro A* algoritmus důležitá. Špatná volba může vést až k exponenciální složitosti. Optimální heuristická funkce vede na polynomiální složitost [9].

Heuristickou funkcí pro A* algoritmus se rozumí funkce na změření vzdálenosti mezi daným a cílovým uzlem.

Dále budou popsány tři nejčastěji užívané metriky pro A* algoritmus [9]. Užít se však mohou i jiné metriky.

Konstantní nula. A* algoritmus vznikl vylepšením Dijkstrova algoritmu heuristikou. Tato „metrika“ tedy dělá z A* algoritmu zpětně algoritmus Dijkstrův. Všechna h skóre budou nulová a záleží tedy pouze na g skóre – reálná vzdálenost uzlů [9].

Manhattanská metrika. Tato metrika představuje nejkratší vzdálenost na čtvercové mřížce. Následující Rovnice 2.1 popisuje tuto vzdálenost [11].

$$d_{manhattan} = |x_1 - x_2| + |y_1 - y_2| \tag{2.1}$$

kde x_1 a y_1 označují souřadnice prvního bodu, x_2 a y_2 označují souřadnice druhého bodu.

Euklidovská metrika. Eukleidovský prostor je metrickým prostorem, každé dva body v prostoru mají mezi sebou určitou vzdálenost. Tato vzdálenost je popsána Rovnicí 2.2 [11].

$$d_{euklid} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.2}$$

kde x_1 a y_1 označují souřadnice prvního bodu, x_2 a y_2 označují souřadnice druhého bodu.

Kosinová podobnost. Představuje míru podobnosti dvou vektorů na základě kosinu úhlu mezi nimi. Tuto podobnost popisuje Rovnice 2.3 [11].

$$d_{cosine} = \cos\theta = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{x_1 \cdot y_1 + x_2 \cdot y_2}{\sqrt{x_1^2 + y_1^2} \cdot \sqrt{x_2^2 + y_2^2}} \tag{2.3}$$

kde x_1 a y_1 označují souřadnice prvního bodu A, x_2 a y_2 označují souřadnice druhého bodu B a θ je úhel, který svírá přímka procházející počátkem souřadnic a bodem A s přímkou procházející počátkem souřadnic a bodem B.

3 Navržená a implementovaná řešení

Práce prohlubuje znalosti o neobvyklém druhu bludišť – v čase proměnlivá, dynamická bludiště. Bludiště *statická* se v čase nijak nevyvíjejí, naopak bludiště *dynamická* se v čase mohou měnit.

Časová proměnlivost u dynamických bludišť je zajištěná generováním bludiště za pomoci buněčných automatů.

3.1 Použité technologie

Pro implementaci byl zvolen jazyk C++ ve vybraném standardu C++20. Veškerá implementace logiky využívá standardní datové struktury jazyka, vlastní implementované datové struktury (graf) a vlastní implementace algoritmů.

Grafická část implementace využívá rozhraní OpenGL. Ke grafickému zpracování bylo využito několik knihoven:

- GLFW (Graphics Library Framework) – volně dostupná přenositelná knihovna pro OpenGL zajišťující založení okna a grafického kontextu pomocí volání služeb operačního systému. Webová stránka knihovny – `https://www.glfw.org`.
- GLEW (OpenGL Extension Wrangler Library) – volně dostupná přenositelná knihovna pro načítání rozšíření do OpenGL. Webová stránka knihovny – `https://glew.sourceforge.net`.
- Dear ImGui – volně dostupná přenositelná knihovna grafického uživatelského rozhraní pro C++. Dovoluje volbu backendu, např. OpenGL a GLFW. Repozitář knihovny – `https://github.com/ocornut/imgui`.

- stb – volně dostupná knihovna pro načítání a ukládání rastrových obrázků. V této práci byla využita pro export bludišť do obrázků. Repozitář knihovny – `https://github.com/nothings/stb`.

V neposlední řadě byl využit verzovací systém git a celý projekt je dostupný na adrese `https://github.com/SpeekeR99/BP_2022_2023_Zappe`.

3.2 Reprezentace bludišť

Vybranou reprezentací bludišť v této práci je reprezentace grafem. Důvodem je větší obecnost oproti reprezentací maticí, dovolující experimenty s různými kombinacemi druhů sousedností a pravidel.

Vrcholy jsou implementovány jako přepravky držící informace o tom, kde mají být na obrazovce vykresleny. Tato informace o poloze je dána souřadnicemi X a Y.

Buněčný automat implementovaný grafem pracuje s dvěma grafy (očekává se stejný počet uzlů u obou grafů). Jeden graf (G_1) reprezentuje grafickou podobu bludiště, druhý graf (G_2) reprezentuje možné sousedy daných uzlů pro kontrolu pravidel. Počet vrcholů obou grafů musí být stejný. Množiny hran mohou být odlišné.

$$G_1 = (V_1, E_1) \tag{3.1}$$
$$G_2 = (V_2, E_2) \tag{3.2}$$
$$V_1 = V_2 \tag{3.3}$$
$$E_1 \neq E_2 \tag{3.4}$$

Je tedy možné vykreslovat graf reprezentující ortogonální mřížku, zatímco se sousedé budou kontrolovat i diagonálně – 8–okolí. Lze tedy jednoduše docílit známého buněčného automatu – *Game of Life* od Johna Conwaye. Díky této změně k buněčným automatům je možné objevovat nové kombinace sousedností a pravidel, která by potencionálně mohla generovat i bludiště.

3.3 Generování bludišť

Podkapitola generování bludišť je rozdělena na dvě části. Statická bludiště jsou reprezentována grafem a využívají výše zmíněné algoritmy, viz podkapitola 2.3. Dynamická bludiště jsou reprezentována buněčnými automaty – výše zmíněná kombinace grafů – a používají jako základ ke generování bludišť informace z podkapitoly 2.3.7.

3.3.1 Generování statických bludišť

Nejprve byla implementována datová struktura graf. Implementace využívá spojového seznamu sousedů. Implementace grafu nabízí očekávané funkce – přidání hrany, odebrání hrany, kontrolu sousednosti.

Vybranými a implementovanými řešeními jsou randomizovaná verze průchodu do hloubky a modifikovaný Kruskalův algoritmus.

Jelikož implementaci nelimituje velikost ani tvar grafu, je možné generovat bludiště s různou *teselací*. Implementace je tedy schopná generovat mřížková bludiště různých tvarů buněk – typicky ortogonální mřížka, hexagonální (sigma) mřížka. Mřížka ovšem také není limitací a implementace si poradí i s takovými grafy, které mají různé stupně vrcholů.

Randomizované prohledávání do hloubky

Prohledávání do hloubky je popsáno a pseudokódem nastíněno v podkapitole 2.3.1. Implementace je řešena přes datovou strukturu zásobník a smyčku, aby nebyla zapotřebí rekurze – viz Pseudokód 2.2.

Řešitelnost bludiště je vždy zaručena, protože se při generování graf prochází do hloubky celý – každá buňka je spojená s libovolnou jinou buňkou. Bludiště vygenerované tímto algoritmem je dokonalé – obsahuje právě jednu cestu z jakékoliv buňky do jakékoliv jiné. Z pohledu řešitelnosti tedy nezáleží na volbě startu a cíle, neboť řešení (cesta) existovat musí.

Modifikovaný Kruskalův algoritmus

Obyčejný Kruskalův algoritmus je popsán v podkapitole 2.3.2. Modifikovaná verze algoritmu je převzata z autorovy semestrální práce z předmětu KIV/PRO, resp. ze článku [7]. Hlavní myšlenka této modifikace je, že algoritmus je schopen generovat i nedokonalá bludiště. Navíc se dá ovládat horizontální a vertikální bias – poměr horizontálních a vertikálních cest. Cyklický bias zajišťuje smyčky v bludišti, čímž vznikají nedokonalá bludiště. Implementace je řešena přes datovou strukturu disjunktních množin. Pseudokód 2.4 znázorňuje běh tohoto modifikovaného algoritmu.

Řešitelnost bludišť je opět vždy zajištěna tím, že se při generování bludiště prochází každý vrchol grafu – každá buňka je spojená s libovolnou jinou buňkou. Vygenerované bludiště může být dokonalé i nedokonalé, záleží na volbě parametru

cyklického biasu. Z pohledu řešitelnosti tedy nezáleží na volbě startu a cíle, neboť řešení (cesta) existovat musí.

Implementace modifikovaného Kruskalova algoritmu je vyladěna primárně pro ortogonální mřížkový graf. Pro hexagonální mřížku či jiné obecné grafy nemusí parametry správně fungovat. Článek [7] uvádí modifikace na základě řádků a sloupců, což není praktická definice pro obecné grafy, z tohoto důvodu je správná funkčnost zajištěna pouze pro ortogonální mřížkové grafy, kde pojmy řádek a sloupec mají jasně definovaný význam. Chování pro obecné grafy v této implementaci není definované.

3.3.2 Generování dynamických bludišť

Nejprve byla implementována třída reprezentující buněčný automat. Tato třída využívá výše zmíněnou datovou strukturu graf. Implementace buněčného automatu nabízí očekávanou funkci – evoluci, vygenerování dalšího kroku. Na konci každé evoluce je vynucen živý stav buněk, které reprezentují začátek a konec bludiště.

Implementace buněčného automatu při správném zadání obou grafů, jeden graf pro vykreslování, druhý graf pro kontrolu sousednosti, podporuje výše zmíněné řetězce pravidel v podkapitole 2.3.7. Jelikož řetězce pravidel z podkapitoly 2.3.7 počítají s klasickou maticovou reprezentací, správným zadáním obou grafů je myšleno následující – vykreslovací graf (G_1) je ortogonální mřížkový graf a graf sousednosti (G_2) je ortogonální mřížkový graf s diagonálami (8–okolí).

Řešitelnost bludiště již oproti algoritmům z podkapitoly 3.3.1 zaručena není vždy. V buněčném automatu v dané generaci nemusí cesta z počátku do cíle existovat. Když cesta neexistuje, uživatel je informován grafickým uživatelským rozhraním, že bludiště není řešitelné.

3.4 Řešení bludišť

Díky reprezentaci bludišť grafem se problém řešení bludiště převádí na problém hledání cesty v grafu. Proto jsou všechna zvolená a implementovaná řešení grafová. Konkrétně se jedná o prohledávání do šířky, Dijkstrův algoritmus a A* algoritmus s různými heuristikami.

Zvolené implementace fungují pro oba druhy bludišť, neboť statická bludiště jsou reprezentována grafem a dynamická bludiště jsou reprezentována buněčnými automaty, které jsou reprezentovány grafy.

3.4.1 Prohledávání do šířky

Implementace prohledávání do šířky je v souladu s popisem v podkapitole 2.4.1. Implementace navíc obsahuje zmiňovaný list předchůdců pro možnost zpětného rekonstruování cesty, viz Pseudokód 2.5.

Když je bludiště řešitelné, tak prohledávání do šířky vždy najde řešení. Nalezená cesta je navíc vždy nejkratší. Pro nalezení řešení je však často zapotřebí projít graf celý, algoritmus tedy zbytečně zachází i do slepých cest bludiště.

3.4.2 Dijkstrův algoritmus

Implementace Dijkstrova algoritmu využívá dále popisovanou implementaci A* algoritmu. Předávaná heuristika je konstantní nula, viz podkapitola 2.4.3.

Stejně jako prohledávání do šířky Dijkstrův algoritmus nalézá nejkratší cestu. Oproti průchodu do šířky však Dijsktrův algoritmus nalézá řešení obecně rychleji.

3.4.3 A* algoritmus

Implementace A* algoritmu využívá ohodnocovací funkce popsané v podkapitole 2.4.3 a prioritní frontu. Zpětná rekonstrukce cesty je opět zaručena listem předchůdců.

Nalezená cesta je opět nejkratší. Výhodou oproti Dijkstrově algoritmu je, že díky heuristické funkci algoritmus obecně konverguje k cíli rychleji.

3.4.4 Dynamická bludiště

Problém u dynamických bludišť může být s konvergencí buněčného automatu k řešitelnému bludišti, byly proto implementovány funkce, které nehledají přímo cestu, ale pouze se zabývají otázkou řešitelnosti – zdali je bludiště vůbec řešitelné. Tato funkce využívá obyčejný průchod do šířky bez listu předchůdců a snaží se pouze dostat z jednoho uzlu do druhého. Dalším problémem u dynamických bludišť je nutnost přepočítávání řešení v každé generaci. Navíc je otázkou, zda řešit bludiště jen jako celek (od začátku do konce), nebo uvažovat, že řešitel se mohl v každé generaci přesunout do jakékoliv jiné buňky. Zvoleným řešením tohoto problému je, že se bludiště řeší jako celek (od začátku bludiště do konce), zároveň se bludiště řeší z pozice řešitele, ten se může libovolně přesouvat. Navíc se omezuje pohyb řešitele v dynamických bludištích pouze na moment, kdy generace ručně pozastaví – tedy když zastaví vývoj buněčného automatu. Pokud se řešitel přesunul na libovolnou buňku a opět spustil evoluci buněčného automatu, může se stát, že

buňka v následujících generacích zemře – z tohoto důvodu je implementován algoritmus pro kontrolu živosti buňky na které řešitel stojí. Zemře-li buňka pod řešitelem, je hráč přesunut na nejbližší živou buňku.

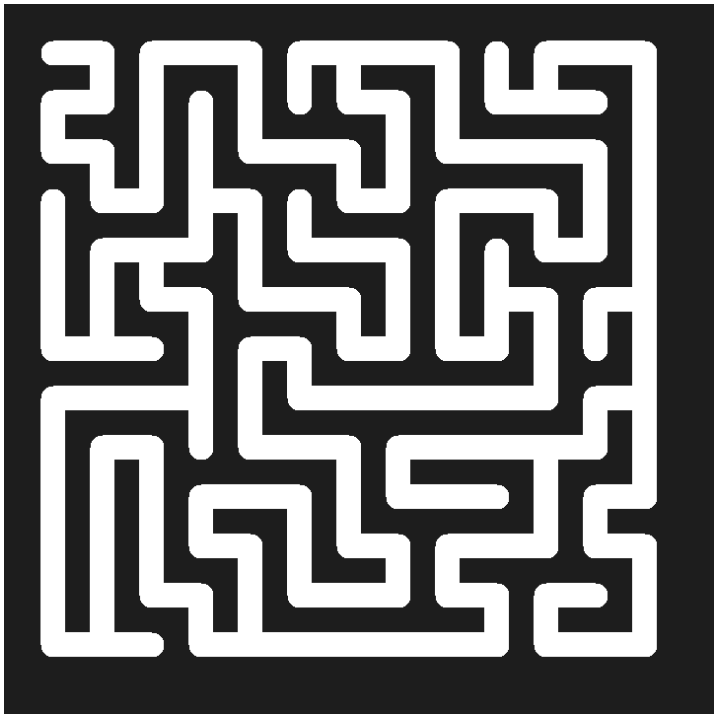
3.5 Vizualizace

Jak již bylo zmíněno v podkapitole 3.2, každý vrchol grafu udržuje informaci o své poloze v souřadnicích X a Y. Poloha těchto vrcholů tak může být volena pseudonáhodně nebo například pravidelně pro vytvoření různých mřížek.

Grafické rozhraní využívá výše zmíněné technologie a knihovny, viz podkapitola 3.1.

3.5.1 Vykreslení bludiště

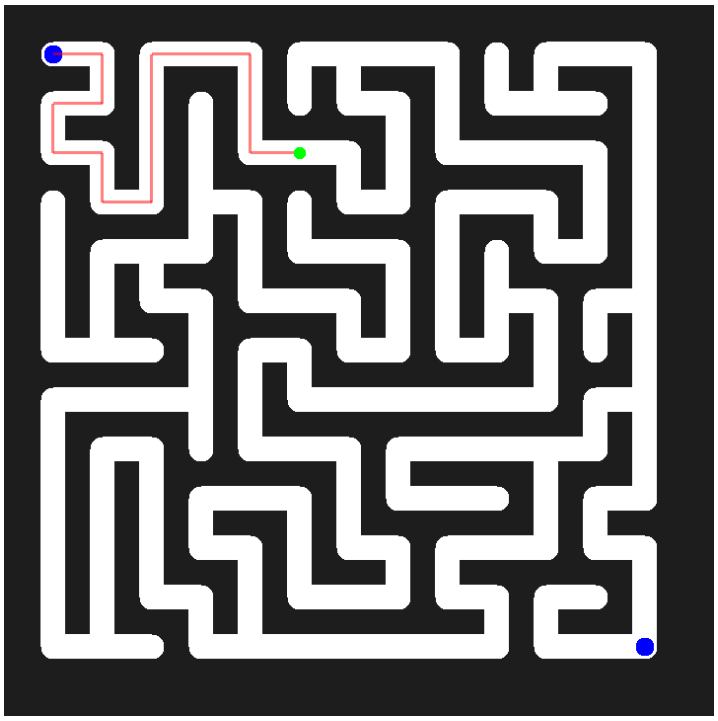
Při vykreslování bludiště se využívá znalosti poloh jednotlivých vrcholů grafu a existujících hran mezi nimi. Vrcholy jsou graficky reprezentovány kruhy, hrany jsou reprezentovány čarami. Hlavní myšlenka spočívá ve vykreslení cest na pozadí odlišné barvy. Ve výchozím nastavení je pozadí tmavé a cesty bílé. Při vhodně zvolených poloměrech a šířkách čar budí tmavé pozadí dojem zdí. Výsledek je možné si prohlédnout na Obrázku 3.1.



Obrázek 3.1: Výsledek kreslení bludiště (vygenerováno pomocí DFS)

3.5.2 Ovládání uživatelem

Implementace byla vylepšena přidáním možnosti pro uživatele si bludiště projít. Zelený kruh v levém horním rohu (začátek bludiště je vždy vlevo nahoře) reprezentuje hráče. Modrý kruh v pravém dolním rohu reprezentuje konec. Hráč – zelený kruh – následuje kurzor myši. Při jakémkoliv pohybu myší se nalezne nejbližší buňka ke kurzoru. Pokud je nejbližší buňka ke kurzoru spojena hranou s buňkou, na které se nachází hráč, hráč se přemístí. Navíc je za hráčem vykreslována červená čára reprezentující doposud prošlou cestu – trajektorie. Zmíněná dodatečná vykreslení jsou vidět na Obrázku 3.2



Obrázek 3.2: Vykreslené bludiště s vyznačeným začátkem a koncem (modré kruhy), řešitelem (zelený kruh) a vykreslenou doposud prošlou trajektorií (červená čára) (vygenerováno pomocí DFS)

3.5.3 Grafické uživatelské rozhraní

Jako „backend“ pro knihovnu Dear ImGui byl zvolen OpenGL a GLFW, neboť v práci jsou již tyto technologie užity.

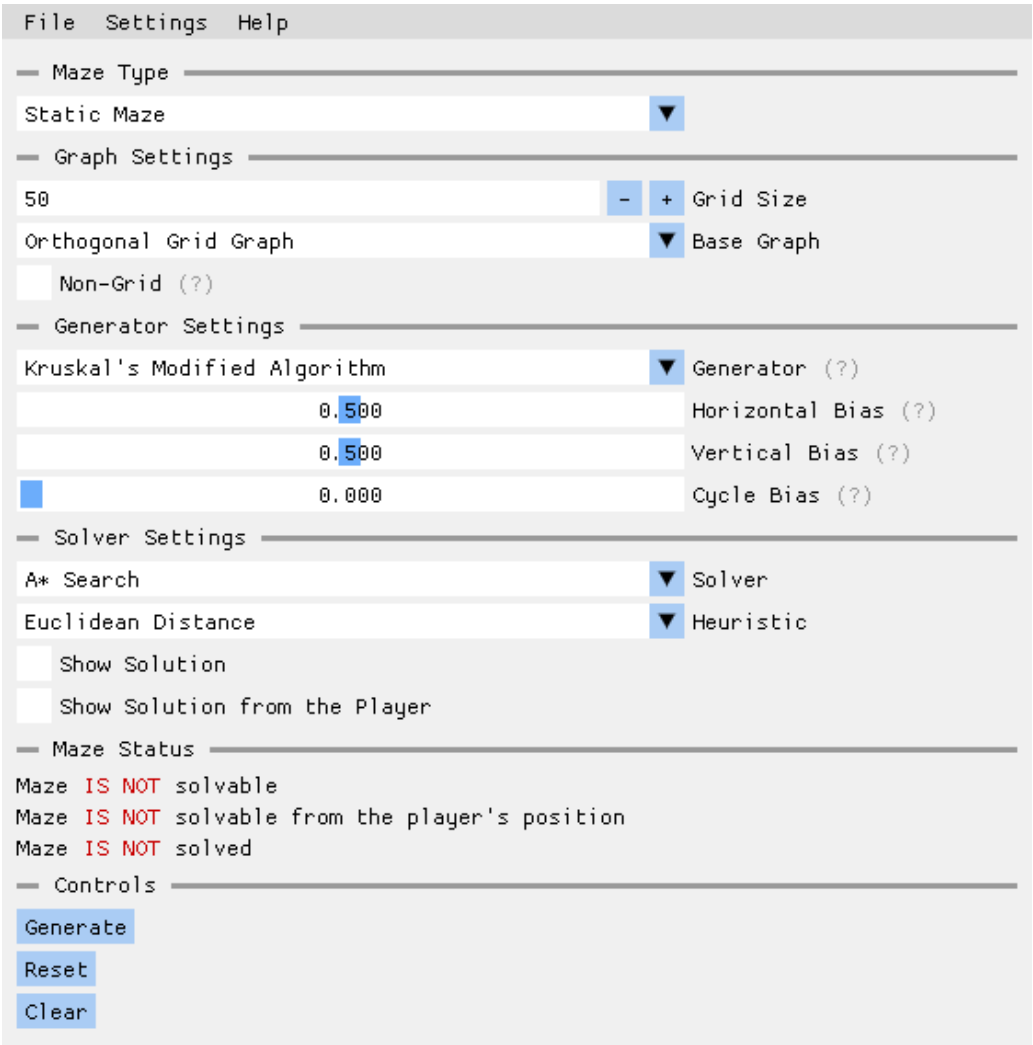
Uživatelské rozhraní bylo rozvrženo do dvou částí – levý ovládací panel, pravá interaktivní část obsahující vykreslené bludiště s možností ručního řešení. Velikost pravé části okna závisí pouze na výšce okna, ta definuje i šířku této části. Pravá část

pro vykreslení bludiště má vždy tvar čtverce. Na zbylé části okna vlevo je vykreslen ovládací panel pomocí knihovny Dear ImGui. Panel obsahuje hlavní menu s položkami:

- File – zde se nachází možnosti pro export obsahu okna do obrázku ve formátu .png a možnost vypnout aplikaci.
- Settings – zde je možnost změny grafického nastavení, která nabízí změnu rozlišení, velikosti fontu a různých barev. Zmíněná nastavení jsou v dialogovém vyskakovacím okně.
- Help – obsahuje pouze položku „About“, která pouze uvádí autora práce.

Levá ovládací část mimo menu obsahuje možnosti přímo související s vykreslovaným bludištěm. Nabízeny jsou možnosti změny druhu bludiště, možnost výběru parametrů pro generování a řešení bludiště. Nabízené druhy jsou statický a dynamický druh bludiště. Parametry pro generování souvisí jak s druhem bludiště, tak s vybraným algoritmem na generování. Parametry řešení bludiště nezávisí na druhu ani na generátoru. V neposlední řadě se v levé části obrazovky nacházejí ovládací tlačítka pro vygenerování zvoleného bludiště, pro obnovení hráčovy pozice a pro vymazání pravé grafické části. Ovládací levá část popisovaného uživatelského rozhraní je vidět na Obrázku 3.3.

Detailní uživatelská příručka je k nalezení v Příloze A.



Obrázek 3.3: Grafické uživatelské rozhraní – ovládací panel

Experimenty a výsledky

4

Veškeré experimenty byly provedeny na osobním počítači s operačním systémem Windows 10 Home. Počítač obsahuje procesor AMD Ryzen 5 4600H 3.00 GHz, 16 GB RAM a grafickou kartu NVIDIA GeForce GTX 1660 Ti.

Jak již bylo zmíněno v podkapitole 3.5.2, předpokladem je, že začátek bludiště je vždy v levém horním rohu a konec bludiště je vždy v pravém dolním rohu.

Animované verze sérií obrázků, které budou k vidění napříč touto kapitolou jsou dostupné na adrese `https://github.com/SpeekeR99/BP_2022_2023_Zappe/tree/master/doc/gifs`.

4.1 Generování bludišť

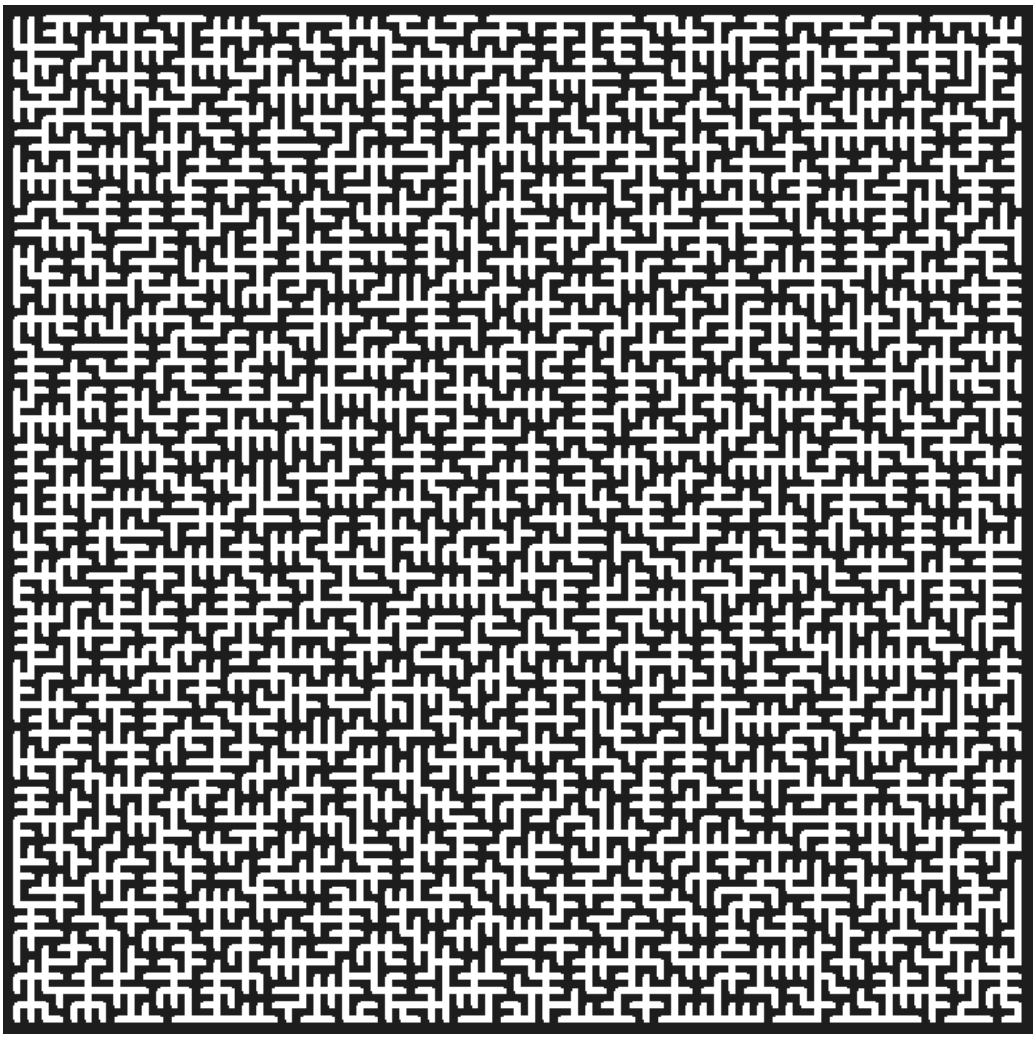
Jedním z prvních volitelných parametrů je *velikost mřížky*. Čím menší je tato velikost určující relativní velikost buňky, tím více bude buněk ve výsledném grafu. Naopak čím větší je hodnota velikost mřížky, tím větší buňky grafu budou a tím méně se buněk do grafu vejde. Výchozí hodnota je 50. Krajní hodnota 100 je na Obrázku 4.1 a druhá krajní hodnota 10 je na Obrázku 4.2. Na Obrázku 4.3 je vidět bludiště s mřížkovou velikostí 50 (výchozí hodnota).

Dále uváděné experimenty jsou rozdělené do dvou částí, neboť mřížková velikost je jedinou společnou částí obou vykreslovaných druhů bludišť.

4.1.1 Statická bludiště

U statických bludišť je možné zvolit algoritmus pro generování a základ grafu, ze kterého se bludiště generuje.

Obrázek 4.1: Bludiště s hodnotou mřížkové velikosti 100 (vygenerováno pomocí Kruskalova algoritmu s výchozími hodnotami)



Obrázek 4.2: Bludiště s hodnotou mřížkové velikosti 10 (vygenerováno pomocí Kruskalova algoritmu s výchozími hodnotami)

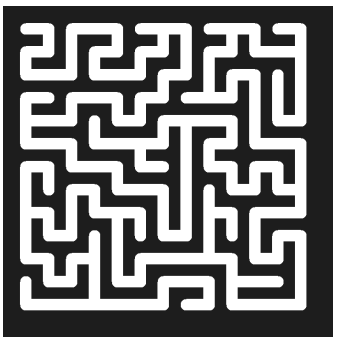
Algoritmy pro generování bludišť

Následující příklady mají nastavenou velikost mřížky na 50 (výchozí hodnota) a všechna bludiště jsou generovaná na základu ortogonálního mřížkového grafu.

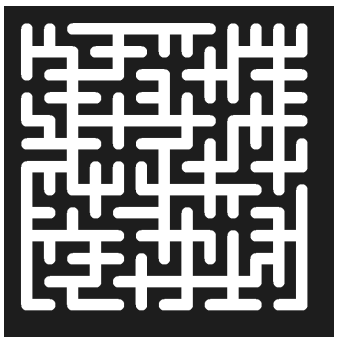
Příklad vygenerovaného bludiště pomocí průchodu do hloubky je k nahlédnutí na Obrázku 4.3. Z obrázku je patrné, že vygenerované bludiště je dokonalé, viz podkapitola 1.4, Směrování. Je vidět, že tento způsob generování má sklony k vysokému faktoru run, viz podkapitola 1.4, Textura. S velice malou pravděpodobností může tento algoritmus vygenerovat i jednocestné bludiště (labyrint), viz podkapitola 1.4, Směrování. Algoritmus by však musel projít každou buňkou od začátku až do konce bez návratové fáze (Backtracking).

Na Obrázku 4.4 je vidět příklad bludiště vygenerovaného s užitím modifikovaného Kruskalova algoritmu. Lze vypozorovat, že oproti předchozímu algoritmu má tento způsob generování naopak sklony k nízkému faktoru run, viz podkapitola 1.4, Textura. Jestli je bludiště dokonalé závisí na cyklickém biasu, je-li nastaven na hodnotu 0, bude bludiště dokonalé, je-li nastaven na jakoukoliv jinou hodnotu (až 1), bylo by bludiště nedokonalé. Dokonalé bludiště vygenerované pomocí základního Kruskalova algoritmu je vidět na Obrázku 4.4. Nedokonalé bludiště se zvýšeným horizontálním a cyklickým biasem je na Obrázku 4.5.

Okrajové případy modifikovaného Kruskalova algoritmu jsou zachyceny v Příloze C.



Obrázek 4.3: Bludiště vygenerované randomizovaným průchodem do hloubky



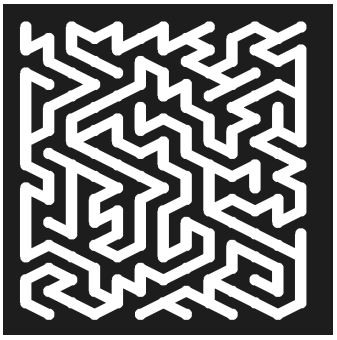
Obrázek 4.4: Bludiště vygenerované Kruskalovým algoritmem (horizontální a vertikální bias 0.5, cyklický bias 0)

Obrázek 4.5: Bludiště vygenerované Kruskalovým algoritmem (horizontální bias 0.75 (vertikální bias 0.25), cyklický bias 0.25)

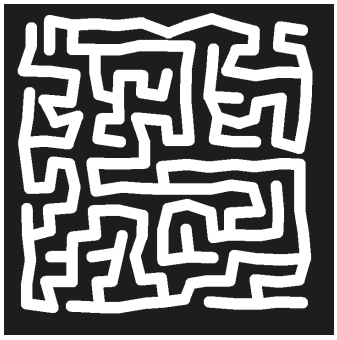
Grafové základy bludišť

Následující příklady mají nastavenou velikost mřížky na 50 (výchozí hodnota) a všechna bludiště jsou generovaná pomocí randomizovaného průchodu do hloubky.

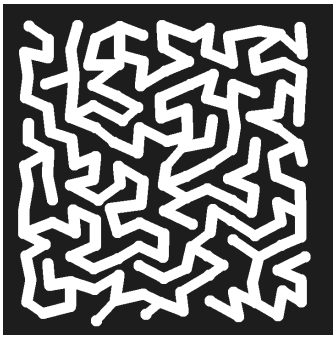
Implementovanými grafy jsou ortogonální mřížkový graf (obdoba maticové reprezentace), hexagonální mřížkový graf a ortogonální mřížkový graf s diagonálami (8–okolí pro graf sousednosti u buněčných automatů). Příklad ortogonálního mřížkového grafu je vidět na Obrázku 4.3, hexagonální mřížkový graf je vyobrazen na Obrázku 4.6. Všechny grafy nabízejí zaškrtávací políčko nazvané *Non–grid*, které způsobí, že se každý vrchol grafu posune o náhodně vygenerované číslo na obou osách. Číslo je generované pomocí normálního rozdělení – grafy tak vypadají nahodileji, ale stále přirozeně. „Nemřížkové“ verze obou grafů jsou k prohlédnutí na Obrázku 4.7 a na Obrázku 4.8



Obrázek 4.6: Hexagonální mřížkový graf jako základ pro bludiště vygenerované pomocí DFS



Obrázek 4.7: Ortogonální „nemřížková“ verze grafu (vygenerováno pomocí DFS)



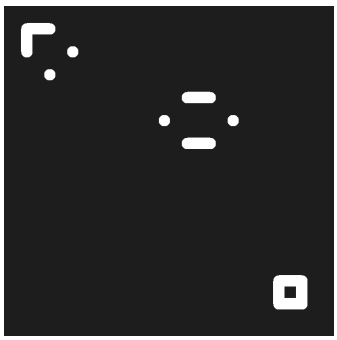
Obrázek 4.8: Hexagonální „nemřížková“ verze grafu (vygenerováno pomocí DFS)

4.1.2 Dynamická bludiště

Dynamická bludiště nabízejí možnost volby grafů (základní graf, graf sousednosti) a pravidel pro evoluci buněčného automatu. Navíc nabízejí změnu hodnoty velikosti strany čtverce inicializovaných buněk, čtverec začíná vlevo nahoře (začátek bludiště). Popsaná hodnota bude dále v práci pro jednoduchost označována jako *inicializační konstanta*. Výchozí hodnota je -1, což je symbolická konstanta reprezentující maximální velikost – tedy inicializují se všechny buňky. Inicializací se rozumí náhodné nastavení nějakých buněk na živé (výchozím stavem je pro buňku stav reprezentující mrtvou buňku). Maximální hodnota závisí jak na velikosti mřížky, tak na rozlišení monitoru. Na osobním počítači s výchozí hodnotou velikosti mřížky symbolizuje konstanta -1 hodnotu 13.

Game of Life

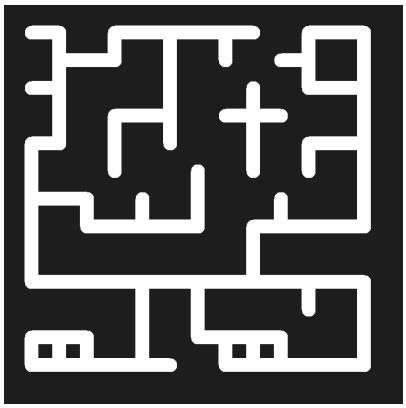
Nejedná se o bludiště, ale lze tímto experimentem ověřit funkčnost buněčného automatu, který je reprezentován grafy. Základní graf je nastaven na ortogonální mřížkový, graf sousednosti je nastaven na ortogonální mřížkový s diagonálami (8–okolí). Řetězec pravidel je nastaven na hodnotu B3/S23, jak fungují řetězce pravidel bylo vysvětleno v podkapitole 2.3.7. Na Obrázku 4.9 je vidět, že po ustálení buněčného automaty jsou k nalezení klasické statické útvary z Game of Life – *loďka* (vlevo nahoře), *včelí úl* (uprostřed) a *blok* (vpravo dole).



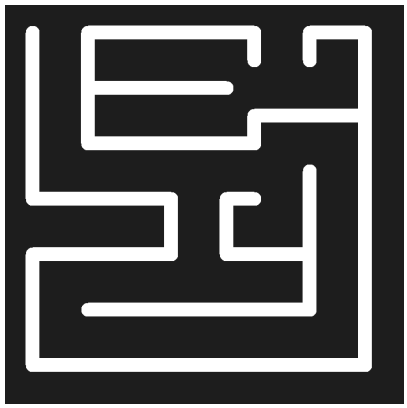
Obrázek 4.9: Buněčný automat reprezentující Game of Life (B3/S23 na ortogonální mřížce s 8–okolní sousedností)

Maze a Mazectric

Jedná se o zmiňovaná pravidla z podkapitoly 2.3.7 – B3/S12345 a B3/S1234. Vykreslovací (základní) graf byl nastaven na ortogonální mřížkový a graf sousednosti je opět nastaven na ortogonální mřížkový s diagonálami (8–okolí) – pro napodobení reprezentace maticí. Vygenerovaná bludiště jsou často neřešitelná, neboť začátek a konec bludiště nejsou spojená žádnou cestou. Obrázek 4.10 a Obrázek 4.11 ukazují případy, kdy generování bylo úspěšné. Obrázky zachycují finální stav buněčného automatu, po kterém se již grafy nijak nevyvíjejí. Z obrázků je vidět, že oba řetězce pravidel mají sklony k vytváření řídkých bludišť, viz podkapitola 1.4, Směrování. Dále je vidět, že pravidla B3/S12345 umožňují vznik smyček, tedy výsledná bludiště mohou být nedokonalá. Oproti tomu pravidla B3/S1234 vytvářejí pouze dokonalá bludiště.

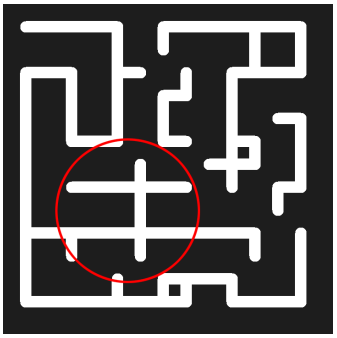


Obrázek 4.10: Řetězec pravidel Maze (B3/S12345)

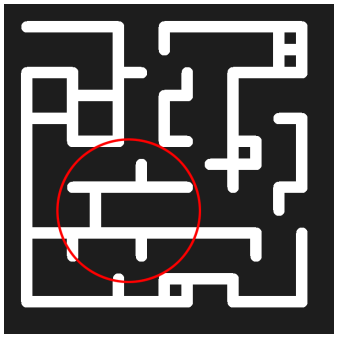


Obrázek 4.11: Řetězec pravidel Mazectric (B3/S1234)

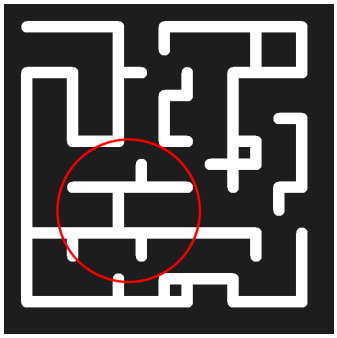
Přidáním sedmičky do pravidel pro zrození – B37/S12345 a B37/S1234 – vznikají v podlouhlých chodbách napříč bludištěm oscilátory (buňky opakující nějaký vzor s určitou periodou) připomínající mosty mezi dlouhými chodbami. Následující série šesti obrázků[1] (Obrázek 4.12 – 4.17) ukazuje tento fenomén s periodou opakování 6. Měnící se „mosty“ jsou označené červenou kružnicí.



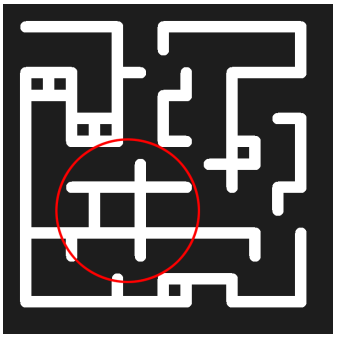
Obrázek 4.12: B37/S12345 v čase t_1



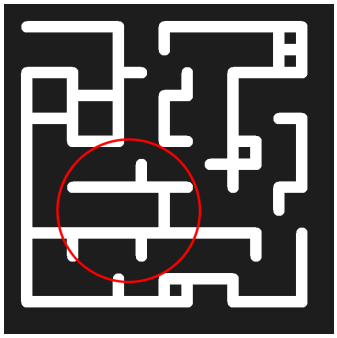
Obrázek 4.13: B37/S12345 v čase t_2



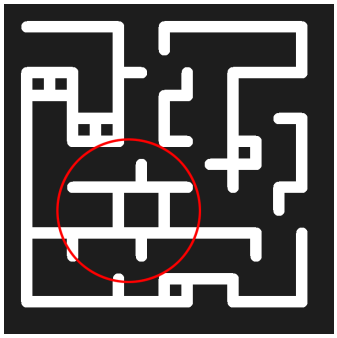
Obrázek 4.14: B37/S12345 v čase t_3



Obrázek 4.15: B37/S12345 v čase t_4



Obrázek 4.16: B37/S12345 v čase t_5

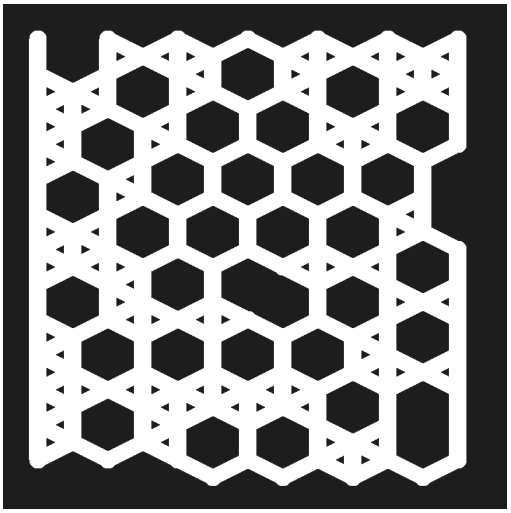


Obrázek 4.17: B37/S12345 v čase t_6

[1] Animace je k nahlédnutí v repozitáři projektu, viz úvod kapitoly 4.

Vlastní navržené kombinace grafů a pravidel

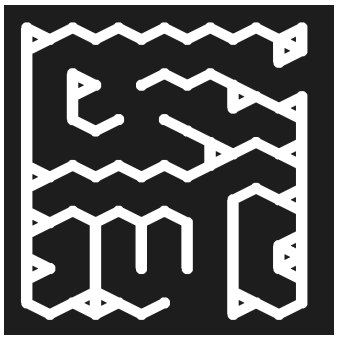
Plástve. Experimentálně byl objeven fenomén generování „pláství“. Nutností je nastavení obou grafů na hexagonální mřížku. Vyzkoumané řetězce pravidel jsou B24/S12345, B24/S1234, B24/S2345. Všechny tyto kombinace generují podobné výsledky. Inicializační konstanta je nastavena buď na původní hodnotu -1 (inicializace celého grafu), nebo na 7. Při volbě hodnoty inicializační konstanty 7 se plástve postupně generacemi rozrůstají z levého horního okraje přes celý zbytek okna. Z pohledu řešitelnosti se jedná o příliš jednoduché bludiště, neboť cest ze začátku do cíle existuje několik, jedná se tedy o nedokonalé bludiště. Na Obrázku 4.18 byl užit řetězec pravidel B24/S2345 s hodnotou inicializační konstanty -1.



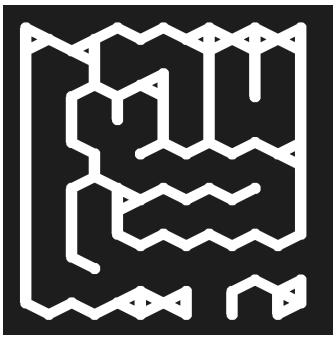
Obrázek 4.18: Bludiště připomínající svými cestami plástve včelího úlu

Hexagonální mřížkový graf s 8–okolím. Zajímavější bludiště jsou generována kombinací hexagonálního mřížkového základu s grafem sousednosti 8–okolí (ortogonální mřížkový graf s diagonálami). Nalezená pravidla jsou B23/S12345 a B3/S1234. První řetězec pravidel poskytuje bludiště více zaplněné cestami, často však lze tato bludiště řešit pouhou chůzí po okraji bludiště. Druhý řetězec pravidel často generuje řídká bludiště, viz podkapitola 1.4, Směrování. Řešení mohou být komplexnější než u prvních zmíněných pravidel. Následující Obrázek 4.19 a Obrázek 4.20 ukazují buněčné automaty ve finální podobě s inicializační konstantou 7. Obrázek 4.21 navíc ukazuje případ, kdy druhý řetězec selže ve vygenerování bludiště (za předpokladu, že pravý dolní roh značí konec bludiště).

Obrázek 4.19: Hexagonální mřížkový graf s 8–okolní sousedností s použitím pravidel B23/S12345

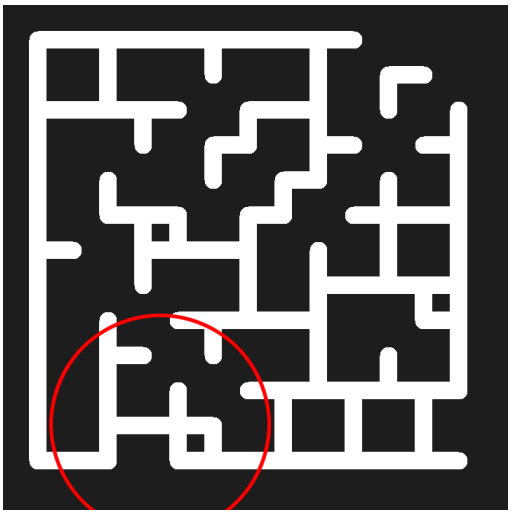


Obrázek 4.20: Hexagonální mřížkový graf s 8–okolní sousedností s použitím pravidel B3/S1234

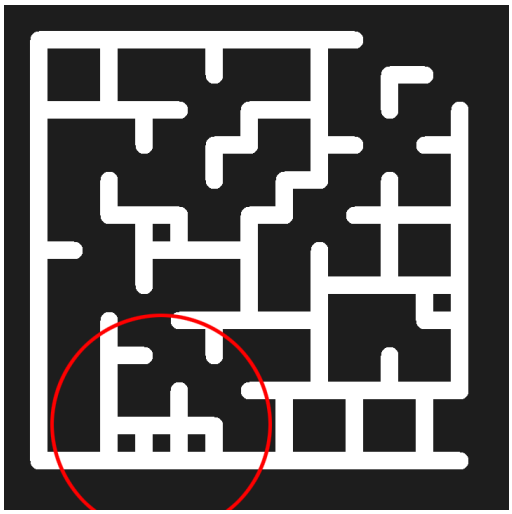


Obrázek 4.21: Hexagonální mřížkový graf s 8–okolní sousedností s použitím pravidel B3/S1234

Ortogonální mřížkový graf s hexagonální sousedností. Bludiště vygenerovaná s pomocí ortogonálního mřížkového základního grafu a hexagonálního mřížkového sousednostního grafu v kombinaci s řetězci pravidel B3/S234 a B2/S234 často generují bludiště, která jsou řešitelná jen v určitých fázích finálních oscilátorů. Pro následující obrázky byla použita konstanta inicializace -1. Na sérii čtyř obrázků[2] (Obrázek 4.22 – 4.25) je vidět oscilátor (označený červenou kružnicí) umožňující cestu od začátku do cíle jen v určitých generacích, konkrétně v časech t_1 a t_2. Na Obrázku 4.26 je vidět obecné bludiště bez oscilátorů, které tato kombinace také dokáže generovat. Ze všech obrázků je mimo jiné vidět, že vygenerovaná bludiště jsou nedokonalá a řídká, viz podkapitola 1.4.

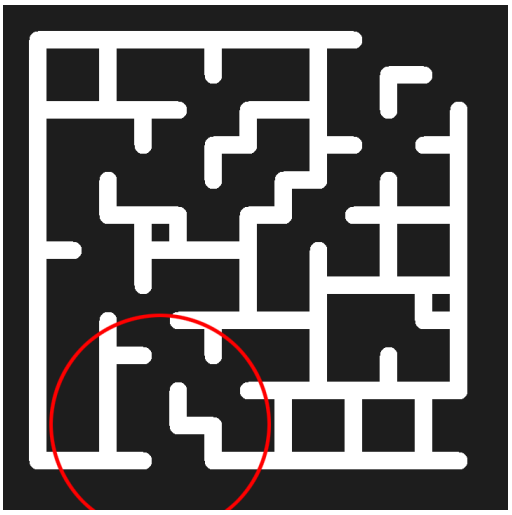


Obrázek 4.22: B3/S234 v čase t_1

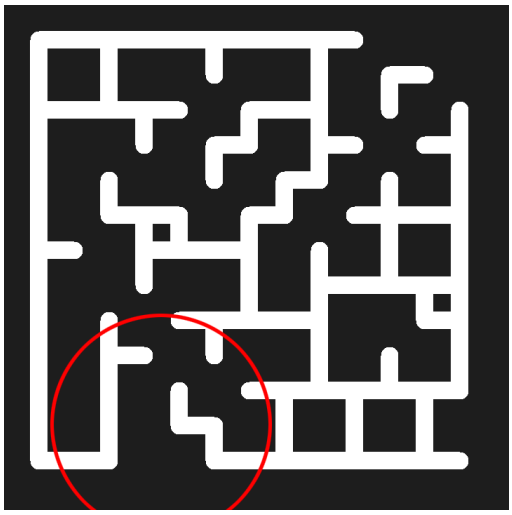


Obrázek 4.23: B3/S234 v čase t_2

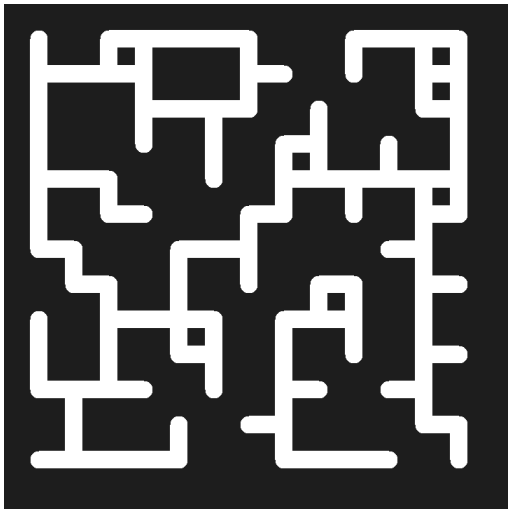
[2]Animace je k nahlédnutí v repozitáři projektu, viz úvod kapitoly 4.



Obrázek 4.24: B3/S234 v čase t_3



Obrázek 4.25: B3/S234 v čase t_4



Obrázek 4.26: Ortogonální mřížkový graf se sousedností v hexagonální mřížce s pravidly B2/S234

Rychlost generování dynamických bludišť

Následující experiment spočíval v měření počtu generací, než je daný buněčný automat řešitelný jako bludiště. Nějaká pravidla nemusí bludiště generovat vůbec, zastavovací podmínka tedy byla nastavena na 10 sekund nebo 1 000 generací buněčného automatu. Pokud daný buněčný automat do 10 sekund, resp. do 1 000 generací, vygeneroval bludiště, byl pokus považován za úspěšný. Bylo prováděno 100 experimentů pro každou kombinaci grafů, pravidel a velikostí inicializačních konstant. Byla testována každá výše uvedená kombinace grafů a pravidel

z podkapitoly 4.1.2. Každá kombinace byla stokrát testována pro každou možnou hodnotu inicializační konstanty při výchozím rozlišení – tj. konstanty od 0 do 13 (13 je totéž jako symbolická konstanta -1). Velikost čtverce 0 a 1 logicky nedávají smysl, neboť neinicializovat nic, resp. jen začátek a konec, nemůže vést k žádnému bludišti. Jelikož bylo testováno 11 kombinací na 13 různých velikostech inicializační konstanty, je nutné pro kompaktnost tabulky těchto třináct hodnot průměrovat pomocí váženého průměru. Výsledný vážený průměr W je určen Rovnicí 4.1. Standardní odchylky σ byly spočítány pomocí Rovnice 4.2, výsledná odchylka uvedená v Tabulce 4.1 je opět spočítána jako vážený průměr jednotlivých odchylek dle Rovnice 4.1. Tabulka 4.1 zobrazuje vážené průměry výsledných časů v jednotkách počtu generací. Tabulka všech hodnot je k nahlédnutí v Příloze D (Tabulka D.1).

$$W = \frac{\sum_{i=0}^{N} w_i \cdot x_i}{\sum_{i=0}^{N} w_i} \tag{4.1}$$

kde w_i značí počet vygenerovaných bludišť a x_i je průměrný počet generací. Iteruje se přes velikost inicializační konstanty – i nabývá hodnot od 0 do 13 ($N = 13$).

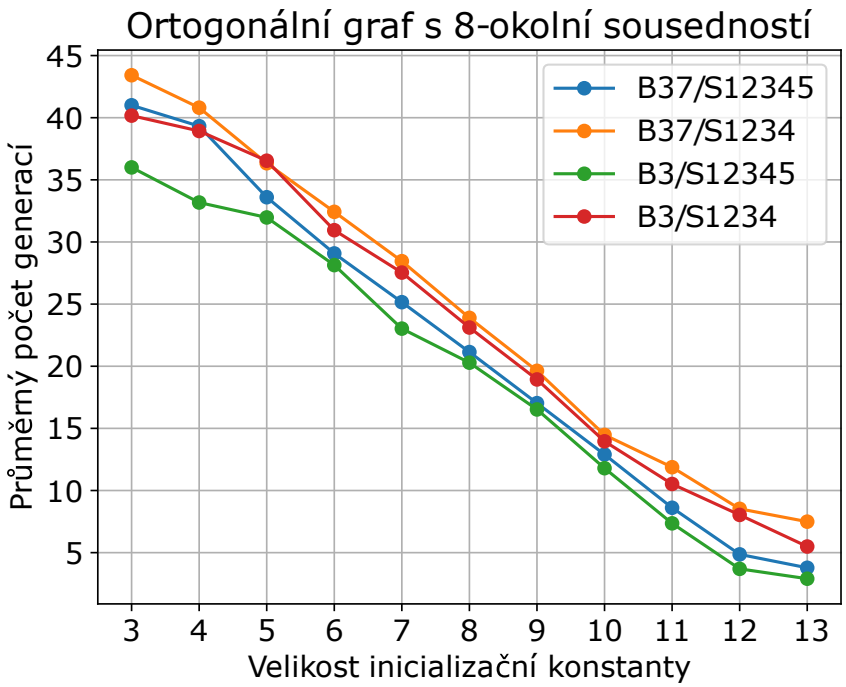
$$\sigma = \sqrt{\frac{\sum_{i=0}^{N}(x_i - \overline{x})^2}{N}} \tag{4.2}$$

kde x_i reprezentuje jednotlivé naměřené hodnoty počtu generací, $\overline{x}$ značí průměr počtu generací a N je počet vygenerovaných bludišť. Iteruje se přes počet vygenerovaných bludišť.

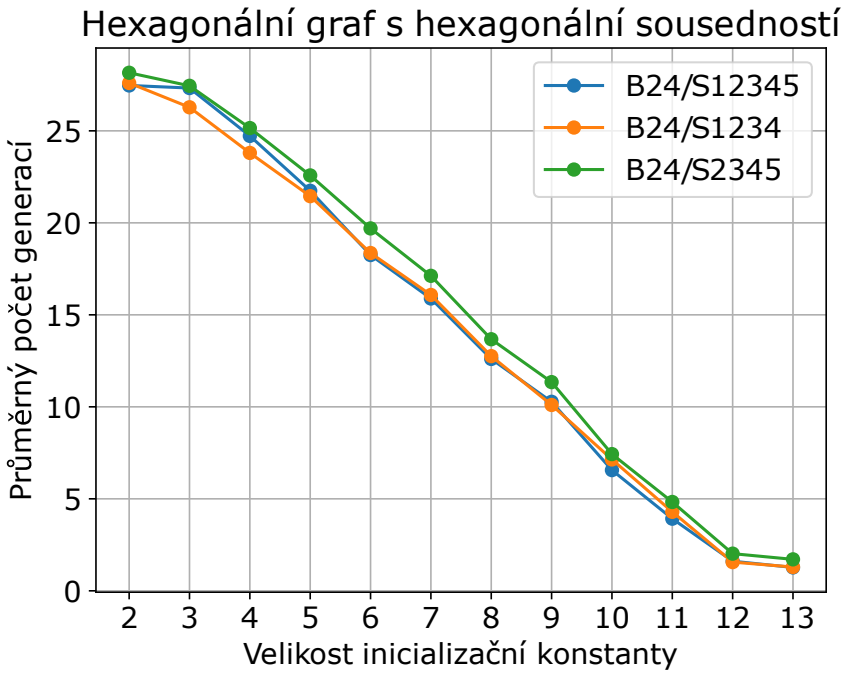
Jako nejrychlejší generátor bludišť se může z Tabulky 4.1 zdát kombinace ortogonálního grafu s hexagonální sousedností s pravidly B3/S234. Proto jsou dále vykresleny grafy (podle skupin grafových kombinací) pro lepší porovnání hodnot. Z Obrázku 4.30 (případně z Tabulky 4.2) je vidět, že výše zmíněná kombinace není spolehlivý generátor bludišť, proto je jeho vážený průměr rychlosti tak nízký, neboť váhu reprezentuje počet vygenerovaných bludišť.

Tabulka 4.1: Počet generací buněčného automatu potřebných pro vytvoření řešitelného bludiště

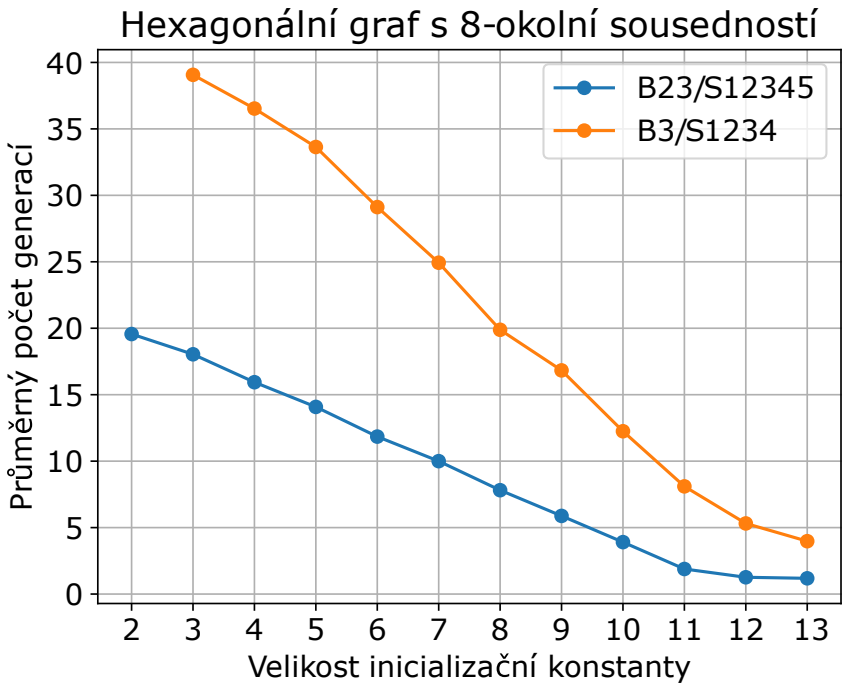
Základní graf	Graf sousednosti	Řetězec pravidel	Vážený průměr počtu generací	Standardní odchylka počtu generací
Ortogonální	8–okolí	B3/S12345	15.45	3.65
Ortogonální	8–okolí	B3/S1234	21.50	4.35
Ortogonální	8–okolí	B37/S12345	18.88	4.47
Ortogonální	8–okolí	B37/S1234	22.60	4.84
Hexagonální	Hexagonální	B24/S12345	14.07	2.05
Hexagonální	Hexagonální	B24/S1234	14.06	1.83
Hexagonální	Hexagonální	B24/S2345	14.68	2.40
Hexagonální	8–okolí	B23/S12345	9.22	0.80
Hexagonální	8–okolí	B3/S1234	18.55	3.84
Ortogonální	Hexagonální	B3/S234	4.82	3.14
Ortogonální	Hexagonální	B2/S234	19.23	3.30



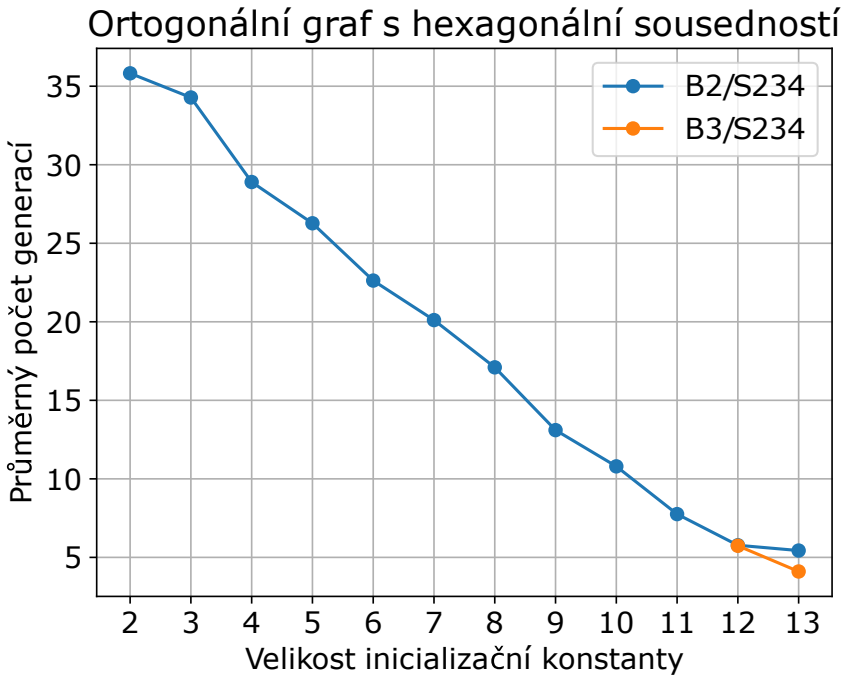
Obrázek 4.27: Graf závislosti průměrného počtu generací pro vytvoření bludiště na inicializační konstantě pro ortogonální graf s 8–okolní sousedností



Obrázek 4.28: Graf závislosti průměrného počtu generací pro vytvoření bludiště na inicializační konstantě pro hexagonální graf s hexagonální sousedností



Obrázek 4.29: Graf závislosti průměrného počtu generací pro vytvoření bludiště na inicializační konstantě pro hexagonální graf s 8–okolní sousedností



Obrázek 4.30: Graf závislosti průměrného počtu generací pro vytvoření bludiště na inicializační konstantě pro ortogonální graf s hexagonální sousedností

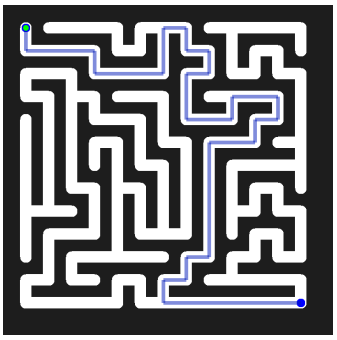
Z výše vyobrazených grafů je vidět obecný trend rychlejší konvergence k řešitelnému bludišti v závislosti na hodnotách inicializační konstanty. Čím větší čtverec je inicializován (ideálně celý buněčný automat), tím menší počet generací je zapotřebí k vytvoření řešitelného bludiště.

4.2 Řešení bludišť

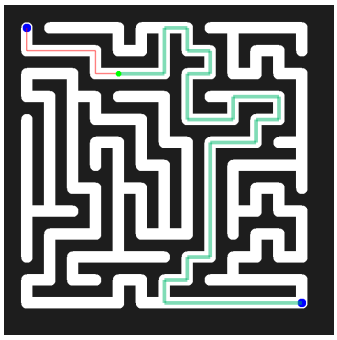
Řešení na rozdíl od generování nemusí být rozděleno do dvou sekcí na základě druhu bludiště.

4.2.1 Algoritmy pro řešení bludišť

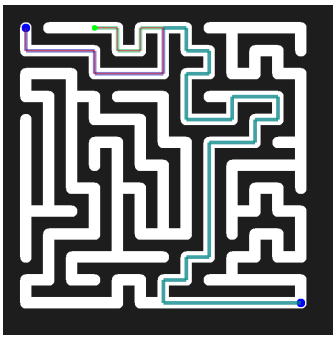
Jak již bylo zmíněno v Sekci 2.4, řešení bludiště je implementováno jak z počátku do konce, tak z pozice řešitele do konce. Ve výchozím nastavení je celkové řešení vykreslováno průhledně modrou barvou, řešení z pozice řešitele je pak vyznačeno barvou průhledně zelenou. Na Obrázcích 4.31 – 4.33 jsou tato vykreslená řešení vidět. U dokonalých bludišť nezáleží na volbě algoritmu, neboť cesta existuje pouze jedna, a všechny implementované algoritmy tedy nalézají stejné řešení.



Obrázek 4.31: Vykreslené řešení od začátku do konce

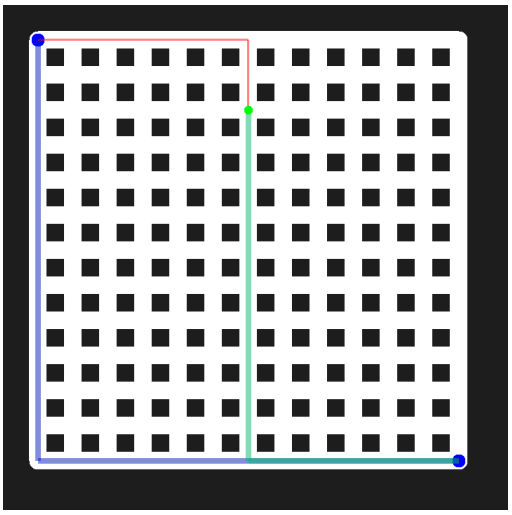


Obrázek 4.32: Vykreslené řešení od řešitele do konce

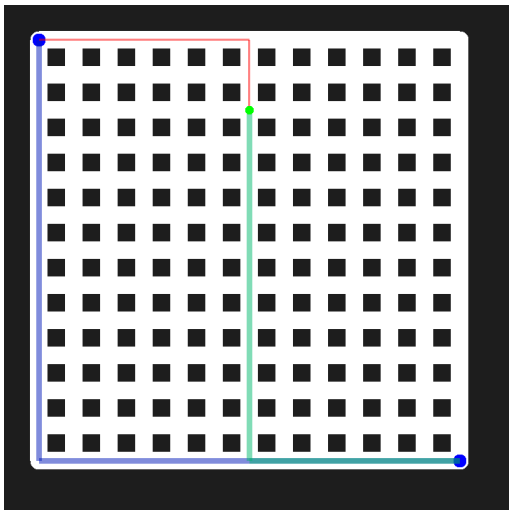


Obrázek 4.33: Vykreslená obě řešení (od začátku do konce, od řešitele do konce)

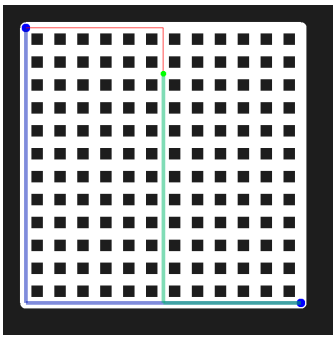
Úplný graf na ortogonální mřížce. Následující experiment zobrazuje výsledné nalezené cesty všech implementovaných řešení. Pro vygenerování zaplněné ortogonální mřížky byl užit modifikovaný Kruskalův algoritmus s cyklickým biasem nastaveným na hodnotu 1. Z Obrázků 4.34 – 4.38 je vidět, že prohledávání do šířky a Dijkstrův algoritmus volí nejkratší cestu. Dle očekávání A* algoritmus s užitím Manhattanské metriky řeší bludiště, jako by se jednalo o *šachovnici* a řešitel se hýbal jako *věž*. Euklidovská metrika se dle očekávání snaží řešení nalézt na diagonále – tedy nejkratší možná vzdálenost ve 2D prostoru. Kosinová podobnost bude více rozebrána ještě později u hexagonální mřížky.



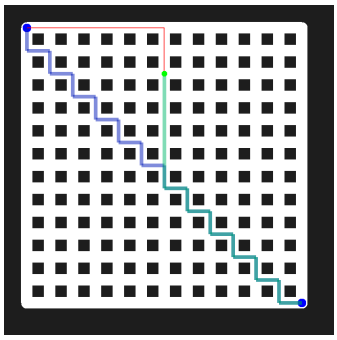
Obrázek 4.34: Řešení nalezené pomocí průchodu do šířky



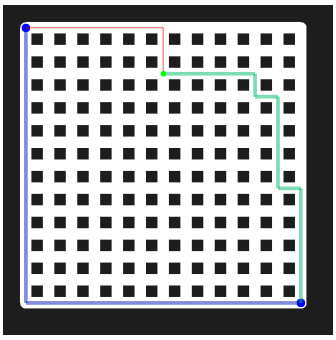
Obrázek 4.35: Řešení nalezené pomocí Dijkstrova algoritmu



Obrázek 4.36: Řešení nalezené pomocí A* algoritmu (použitá heuristika – Manhattanská metrika)

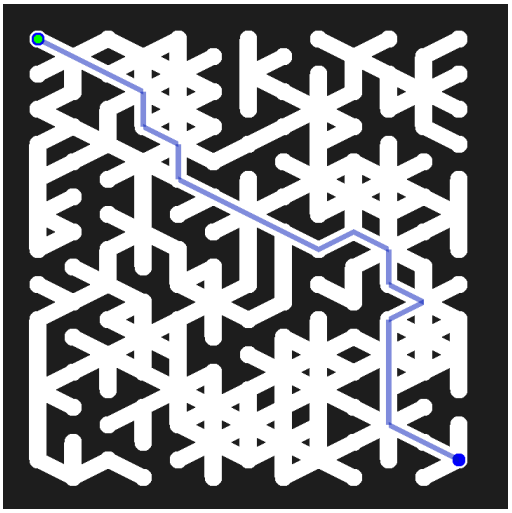


Obrázek 4.37: Řešení nalezené pomocí A* algoritmu (použitá heuristika – Euklidovská metrika)

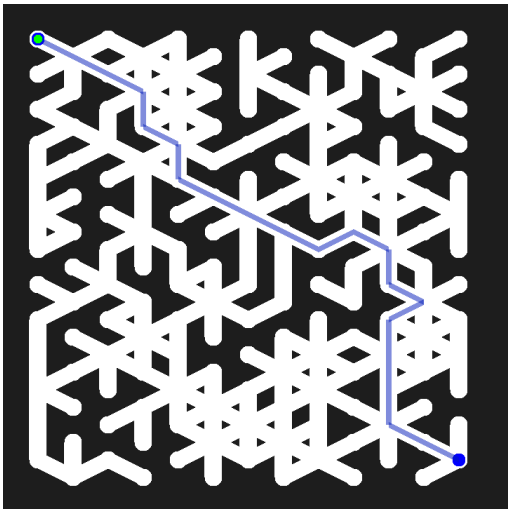


Obrázek 4.38: Řešení nalezené pomocí A* algoritmu (použitá heuristika – kosinová podobnost)

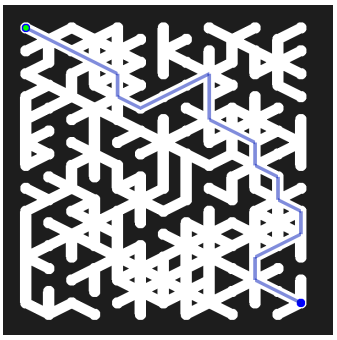
Řešení hexagonálních mřížek. Následující ukázka je vygenerována pomocí modifikovaného Kruskalova algoritmu na hexagonálním mřížkovém základu. Biasy byly nastaveny na následující hodnoty: horizontální bias = 0.8 (vertikální bias = 0.2) a cyklický bias = 0.3. Vyobrazená řešení na následujících obrázcích (Obrázek 4.39 – 4.43) jsou vygenerována opět pomocí všech možných nabízených algoritmů a heuristik. Z obrázků je patrné, že kosinová podobnost, resp. Dijkstrův algoritmus a prohledávání do šířky, jsou optimální řešící algoritmy pro bludiště založená na hexagonální mřížce.



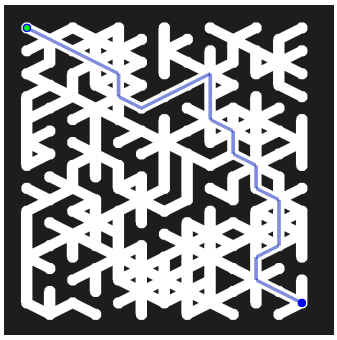
Obrázek 4.39: Řešení nalezené pomocí průchodu do šířky



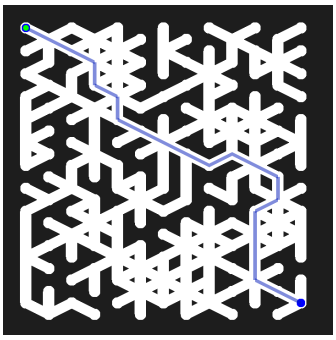
Obrázek 4.40: Řešení nalezené pomocí Dijkstrova algoritmu



Obrázek 4.41: Řešení nalezené pomocí A* algoritmu (použitá heuristika – Manhattanská metrika)



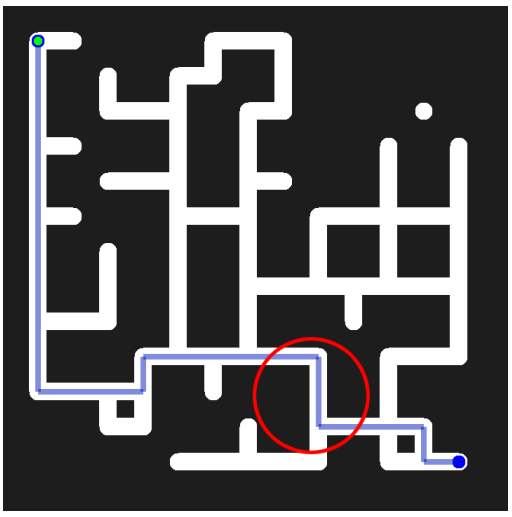
Obrázek 4.42: Řešení nalezené pomocí A* algoritmu (použitá heuristika – Euklidovská metrika)



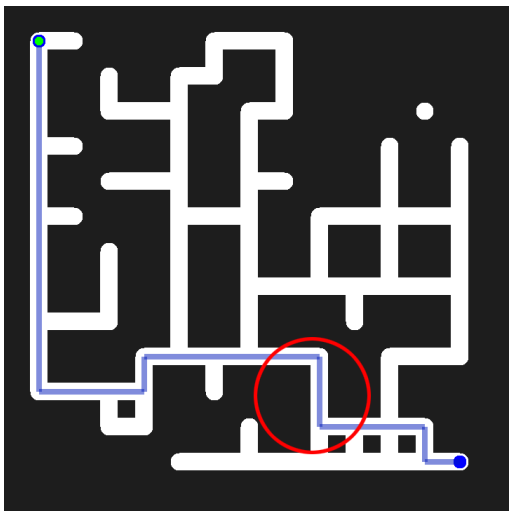
Obrázek 4.43: Řešení nalezené pomocí A* algoritmu (použitá heuristika – kosinová podobnost)

4.2.2 Řešitelnost dynamických bludišť

Oscilátory. Jak již bylo zmíněno v podkapitole 4.1.2, Kombinace ortogonálního mřížkového grafu s hexagonální sousedností, určité kombinace mohou vytvářet oscilátory v cestě řešení. Série Obrázků[3] 4.44 – 4.47 zobrazuje případ, kdy kombinace ortogonálního mřížkového grafu s hexagonální sousedností za užití řetězce pravidel B3/S234 vytvořila v cestě řešení oscilátor (označený červenou kružnicí) s periodou opakování 4.

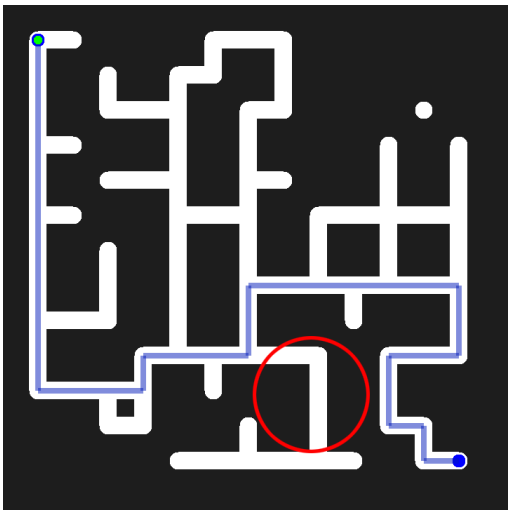


Obrázek 4.44: Řešení v čase t_1

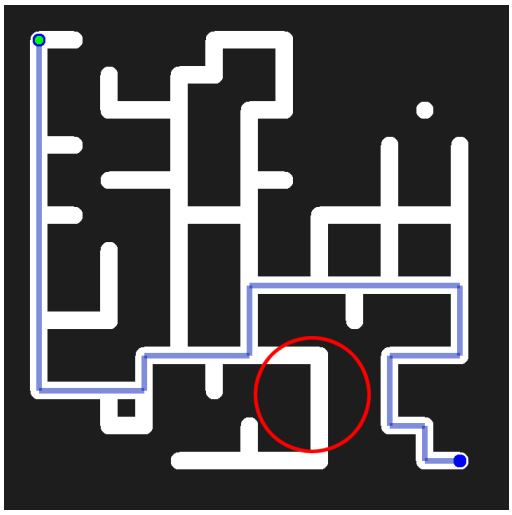


Obrázek 4.45: Řešení v čase t_2

[3] Animace je k nahlédnutí v repozitáři projektu, viz úvod kapitoly 4.



Obrázek 4.46: Řešení v čase t_3



Obrázek 4.47: Řešení v čase t_4

Pravděpodobnost vygenerování řešitelného bludiště. Tento experiment volně navazuje na experiment z podkapitoly 4.1.2. Experiment spočívá v měření počtu řešitelných bludišť vygenerovaných pomocí buněčných automatů. Opět byla nastavena zastavovací podmínka na 10 sekund čekání nebo 1 000 generací buněčného automatu. Bylo provedeno 100 experimentů pro každou kombinaci grafů, pravidel a velikostí inicializačních konstant. Testovány byly všechny kombinace uvedené v podkapitole 4.1.2. Opět byly otestovány všechny hodnoty inicializační konstanty při výchozím rozlišení – tj. hodnoty od 0 do 13 (13 je totéž jako symbolická konstanta -1). Následující Tabulka 4.2 zobrazuje pravděpodobnosti, že daná kombinace grafů a pravidel vygeneruje řešitelné bludiště. Sloupec označený jako P_{-1} zobrazuje průměrnou pravděpodobnost pro hodnotu inicializační konstanty -1, výpočet této pravděpodobnosti P_{-1} je zobrazen v Rovnici 4.3. Následující sloupec ukazuje průměr pravděpodobností, že daná kombinace grafů a pravidel generuje řešitelné bludiště napříč všemi možnými hodnotami inicializační konstanty (-1 až 12, resp. 0 až 13). Tabulka D.1 v Příloze D obsahuje všechny hodnoty užité k výpočtům.

$$P_{-1} = \frac{x_{-1}}{N} \tag{4.3}$$

kde x_{-1} reprezentuje počet vygenerovaných bludišť a N značí počet pokusů (= 100).

Tabulka 4.2: Pravděpodobnost daných buněčných automatů, že vygenerují řešitelné bludiště

Základní graf	Graf sousednosti	Řetězec pravidel	P_{-1}	Průměr P
Ortogonální	8–okolí	B3/S12345	0.51	0.31
Ortogonální	8–okolí	B3/S1234	0.65	0.40
Ortogonální	8–okolí	B37/S12345	0.55	0.45
Ortogonální	8–okolí	B37/S1234	0.77	0.50
Hexagonální	Hexagonální	B24/S12345	1.00	0.84
Hexagonální	Hexagonální	B24/S1234	0.99	0.84
Hexagonální	Hexagonální	B24/S2345	1.00	0.83
Hexagonální	8–okolí	B23/S12345	0.98	0.83
Hexagonální	8–okolí	B3/S1234	0.88	0.55
Ortogonální	Hexagonální	B3/S234	0.59	0.08
Ortogonální	Hexagonální	B2/S234	0.46	0.38

Z Tabulky 4.2 je patrné, že nejspolehlivěji fungují kombinace se základem grafu na hexagonální mřížce. Výsledná bludiště však často nejsou zábavná pro řešitele, neboť se často dají vyřešit pouhou chůzí po okraji bludiště. Nejzajímavější volbou je nejméně spolehlivý generátor – ortogonální mřížkový základ se sousedností v hexagonální mřížce s pravidly B2/S234. Pravděpodobnost vygenerování řešitelného bludiště je sice pouze 46 %, ale výsledné bludiště se jeví nahodile a často není řešení zjevné na první pohled. Ukázka takového bludiště již byla v podkapitole 4.1.2, Kombinace ortogonálního mřížkového grafu s hexagonální sousedností – Obrázek 4.26.

Závěr 5

Práce zavedla nové dělení bludišť – statická a dynamická bludiště. Pro oba druhy bludišť bylo navrženo více možností generování i průchodu. Oba druhy bludišť jsou reprezentovány pomocí abstraktní datové struktury graf.

Práce předvedla navržené řešení reprezentace a implementace buněčných automatů pomocí grafů. Experimentálně byly nalezeny kombinace grafů a řetězců pravidel, které spolehlivě generují bludiště.

Implementace by se dala do budoucna vylepšit více možnostmi pro generování a řešení bludišť. Dalo by se implementovat více algoritmů pro generování statických bludišť a více algoritmů pro průchod grafu. Dalším vylepšením by mohlo být zavedení více grafových základů, případně rozšíření pro Voroného diagramy. Bylo by tak možné prozkoumávat větší množství kombinací dynamických bludišť.

Uživatelská příručka

Repozitář projektu je veřejně dostupný na githubu na adrese `https://github.com/SpeekeR99/BP_2022_2023_Zappe`.

Předpoklady

Pro správný překlad a běh programu je zapotřebí mít:

- CMake verze 3.0 a vyšší (doporučeno 3.22)
- C++ překladač (např. clang, gcc nebo msvc)
- Standardní C++ knihovny standardu C++14 a novější (doporučeno C++20)

Nástroj CMake si lze stáhnout z oficiálních stránek `https://cmake.org`. Standardní knihovny jazyka C++ jsou součástí překladače.

Naklonování repozitáře

Jelikož repozitář obsahuje tzv. *podmoduly*, je zapotřebí projekt správně naklonovat.

Nejjednodušší možností je naklonovat vše pomocí jednoho příkazu. Ukázka tohoto postupu pro operační systém Windows je vidět na Výpisu A.1. Stejný postup platí i pro operační systém Linux, viz Výpis A.2.

Výpis A.1: Ukázkový výpis při klonování repozitáře (Windows)

```
C:\Users\Uzivatel>git clone --recursive
    https://github.com/SpeekeR99/BP_2022_2023_Zappe.git
... (zde by se měl objevit dlouhý výpis)

C:\Users\Uzivatel>
```

Výpis A.2: Ukázkový výpis při klonování repozitáře (Linux)

```
uzivatel@pocitac:~$ git clone --recursive
    https://github.com/SpeekeR99/BP_2022_2023_Zappe.git
... (zde by se měl objevit dlouhý výpis)
uzivatel@pocitac:~$
```

Překlad

Po úspěšném stažení projektu je nutné aplikaci přeložit. Pro tento účel slouží nástroj CMake spolu s překladačem jazyka C++. V adresáři projektu si např. vytvořte adresář s názvem *build*. Výpis A.3 ukazuje postup jak si aplikaci v tomto adresáři sestavit pod operačním systémem Windows. Pro Linux je postup ukázán ve Výpisu A.4.

Nezapomeňte na přepínač **Release** u obou operačních systémů! Z hlediska optimalizací je velice důležité sestavit aplikaci v tomto módu.

Výpis A.3: Ukázkový výpis při překladu projektu (Windows)

```
C:\Users\Uzivatel\BP_2022_2023_Zappe\build>cmake ../
... (zde by se měl objevit dlouhý výpis)
-- Configuring done
-- Generating done
-- Build files have been written to:
    C:/Users/Uzivatel/BP_2022_2023_Zappe/build

C:\Users\Uzivatel\BP_2022_2023_Zappe\build>cmake --build . --config
    Release
... (zde by se měl objevit dlouhý výpis)

C:\Users\Uzivatel\BP_2022_2023_Zappe\build>
```

Výpis A.4: Ukázkový výpis při překladu projektu (Linux)

```
uzivatel@pocitac:~\BP_2022_2023_Zappe\build$ cmake
    -DCMAKE_BUILD_TYPE=Release ../
... (zde by se měl objevit dlouhý výpis)
-- Configuring done
-- Generating done
-- Build files have been written to:
    /home/uzivatel/BP_2022_2023_Zappe/build
uzivatel@pocitac:~\BP_2022_2023_Zappe\build$ make
... (zde by se měl objevit dlouhý výpis)
uzivatel@pocitac:~\BP_2022_2023_Zappe\build$
```

Spuštění

Po překladu dle výše uvedeného návodu by se spustitelný soubor pro operačním systémem Windows měl nacházet ve složce BP_2022_2023_Zappe/build/Release; spustitelný soubor pro operační systém Linux by se měl nacházet ve složce BP_2022_2023_Zappe/build.

Aplikace nepředpokládá žádné vstupní parametry z příkazové řádky. Příklad úspěšného spuštění předvádí Výpisy A.5 a A.6. Po uvedeném výpisu by se navíc mělo otevřít okno s grafickým uživatelským rozhraním.

Výpis A.5: Ukázkový výpis po spuštění aplikace (Windows)

```
C:\Users\Uzivatel\BP_2022_2023_Zappe\build\Release>BP_2022_Zappe.exe
OpenGL version: 3.0.13596 Compatibility Profile Context
    20.10.32.09 27.20.11032.9001
GLEW version: 2.2.0
```

Výpis A.6: Ukázkový výpis po spuštění aplikace (Linux)

```
uzivatel@pocitac:~\BP_2022_2023_Zappe\build$ ./BP_2022_Zappe
OpenGL version: 3.3 (Compatibility Profile) Mesa 22.0.5
GLEW version: 2.2.0
```

Obsluha aplikace

Po spuštění aplikace se otevře okno s grafickým uživatelským rozhraním. Uživatelské rozhraní je rozvržené do dvou částí – levý ovládací panel a pravá interaktivní část obsahující vykreslené bludiště s možností ručního řešení. Levá ovládací část mimo hlavní menu obsahuje možnosti nastavení přímo související s vykreslovaným bludištěm.

Hlavní menu

Hlavní menu pod položkou *File* nabízí možnost exportovat si snímek celého okna, pouze bludiště (s hráčem, řešením, apod.) nebo pouze bludiště bez přidané grafiky. Další záložkou je nastavení (*Settings*), které obsahuje grafická nastavení okna. Ve vyskakovacím okně jsou možnosti pro úpravu velikosti okna a fontu, dále je nabízena změna barevných schémat včetně změn jednotlivých barev. Poslední záložka hlavního menu *Help* obsahuje pouze položku *About*, která pouze uvádí autora práce a účel aplikace.

Levý ovládací panel

Prvním rozbalovacím seznamem je výběr druhu bludiště – statické nebo dynamické. Na základě tohoto výběru se upravují zbylé možnosti v reálném čase.

Další sekcí jsou grafová nastavení. Pro statická bludiště lze vybrat velikost mřížky, typ základu grafu a zaškrtávací pole pro „nemřížkové“ vykreslení. Dynamická bludiště navíc nabízejí možnost výběru druhého grafu – grafu sousednosti.

Následující sekce se věnuje generování, zde jsou největší rozdíly mezi dvěma zmiňovanými druhy bludišť. Statická bludiště nabízejí pouze rozbalovací seznam pro výběr algoritmu. Nabízené algoritmy jsou průchod do hloubky a modifikovaný Kruskalův algoritmus, který navíc nabízí tři dodatečné posuvníky pro nastavení biasů. Dynamická bludiště nabízejí nastavení týkající se buněčného automatu. Lze zadat vlastní řetězec pravidel, který je kontrolován regulárním výrazem – musí být dodržen formát B⟨čísla⟩/S⟨čísla⟩. Navíc lze nastavit hodnotu velikosti strany čtverce inicializovaných buněk. V neposlední řadě jsou nabízena nastavení rychlosti vývoje buněčného automatu – posuvníkem lze ovlivnit samotnou rychlost evoluce a zaškrtávacím polem lze pozastavit vývoj kompletně.

Poslední nastavovací sekce je věnována řešení bludišť. Tato sekce je zcela stejná pro oba druhy bludišť. Rozbalovacím seznamem lze vybrat jeden ze tří algoritmů – průchod do šířky, Dijkstrův algoritmus a A* algoritmus, který navíc nabízí v dodatečném rozbalovacím seznamu možnost volby heuristické funkce. Po výběru algoritmu je možné si pomocí zaškrtávacího pole nechat vykreslit řešení od začátku do cíle, nebo řešení od pozice hráče (řešitele) do cíle.

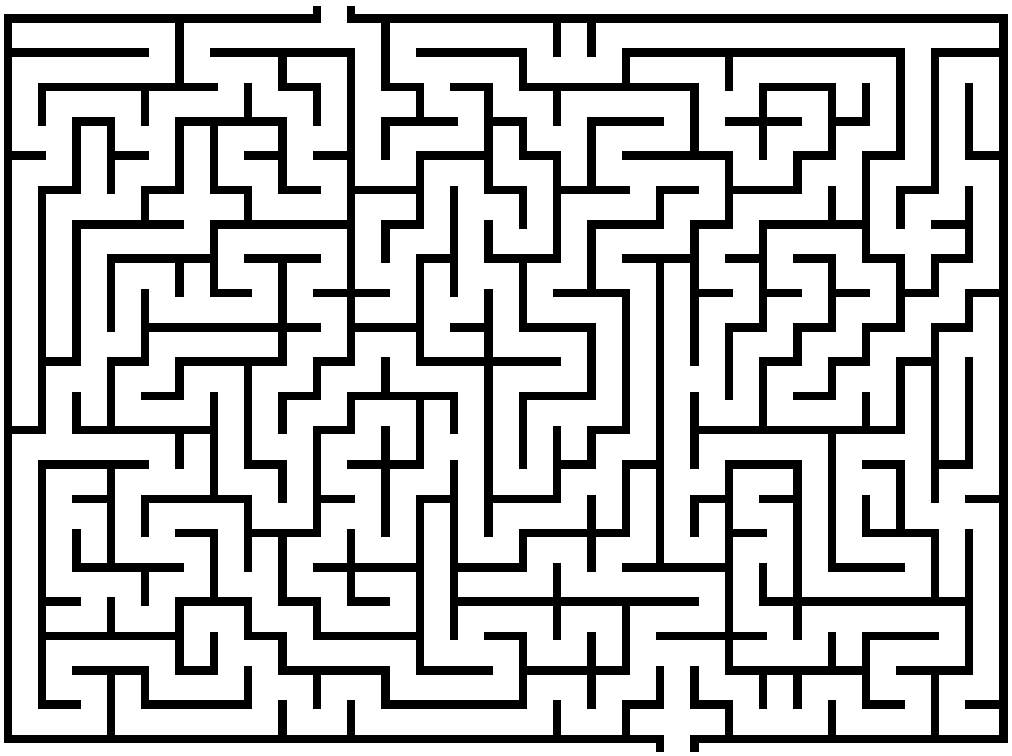
Další dvě sekce obsahují pouze stavové informace a tlačítka pro ovládání. Stavové informace zobrazují, zda je bludiště řešitelné jako celek, zda je bludiště řešitelné od pozice hráče a zda hráč bludiště vyřešil. Ovládací tlačítka jsou tři – tlačítko pro vygenerování zvoleného bludiště, tlačítko pro resetování bludiště do původního stavu (resetování pozice hráče a uražené cesty, resp. resetování buněčného automatu do stavu inicializace) a tlačítko pro vymazání pravé interaktivní části.

Pravá interaktivní část

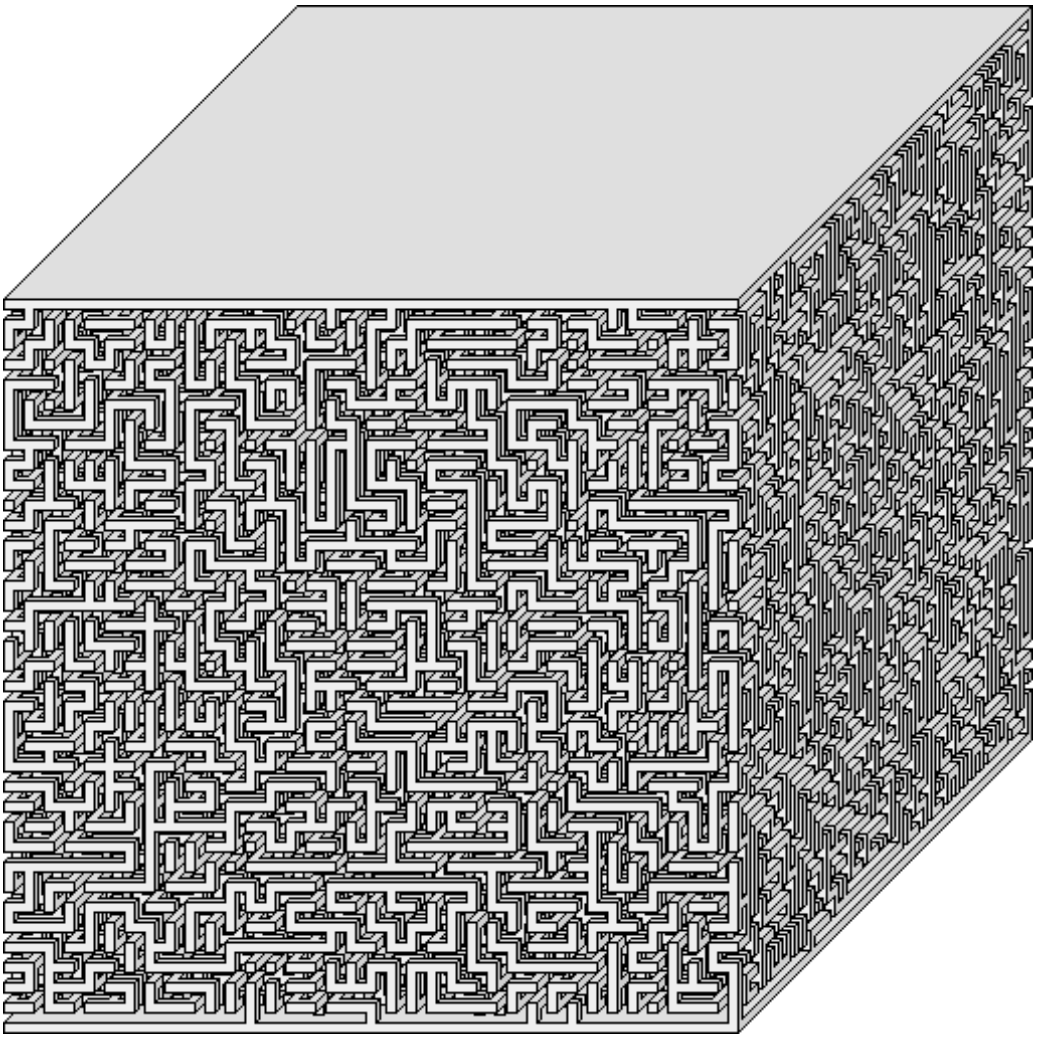
Levý horní roh reprezentuje začátek bludiště (modrý kruh) a pravý dolní roh představuje konec (cíl) bludiště (taktéž modrý kruh). Pohybem myši je možné ovládat hráče (zelený kruh). Hráč se posouvá na nejbližší buňku ke kurzoru, pokud stávající buňka hráče a nejbližší buňka ke kurzoru jsou spojeny hranou. Navíc je u dynamických bludišť zakázán pohyb hráče při vývoji buněčného automatu – je proto nutné si vývoj pozastavit, když chcete bludiště procházet.

Obrázky druhů bludišť

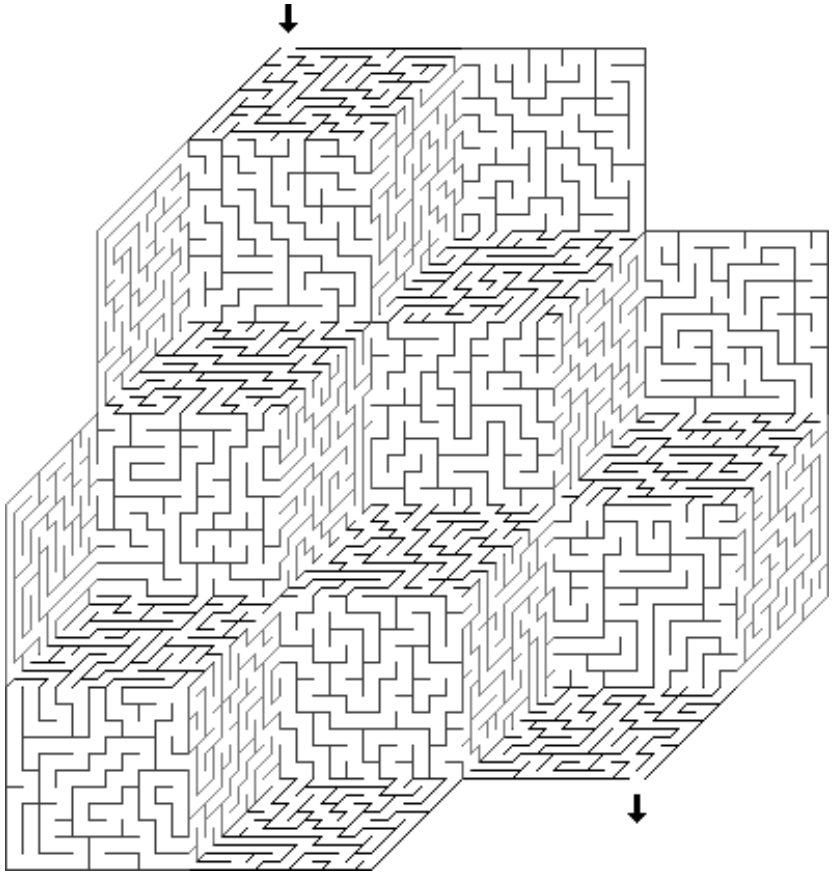
B



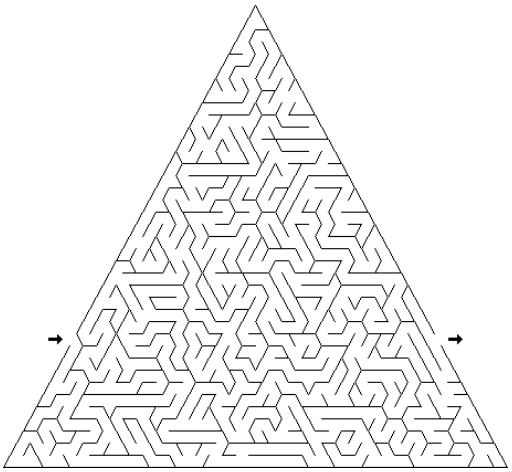
Obrázek B.1: Dokonalé 2D bludiště s ortogonální mřížkovou teselací (převzato z [12])



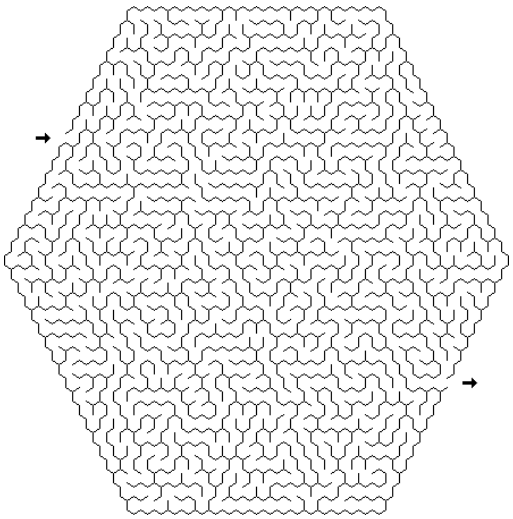
Obrázek B.2: 3D bludiště (převzato z [12])



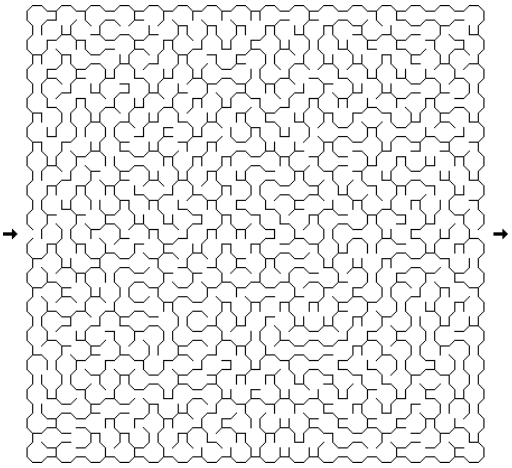
Obrázek B.3: Bludiště vytvořené v neeuklidovském prostoru (převzato z [12])



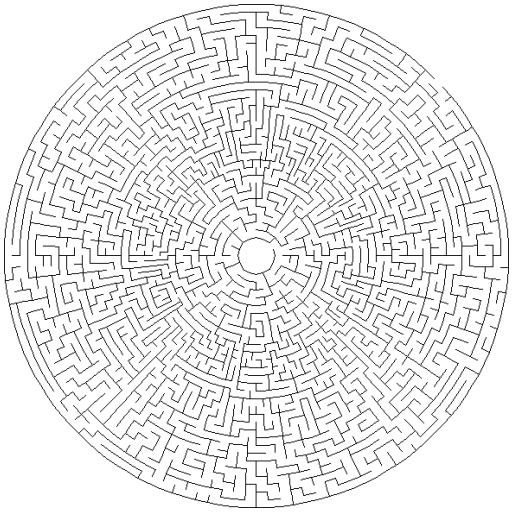
Obrázek B.4: Delta teselace (převzato z [12])



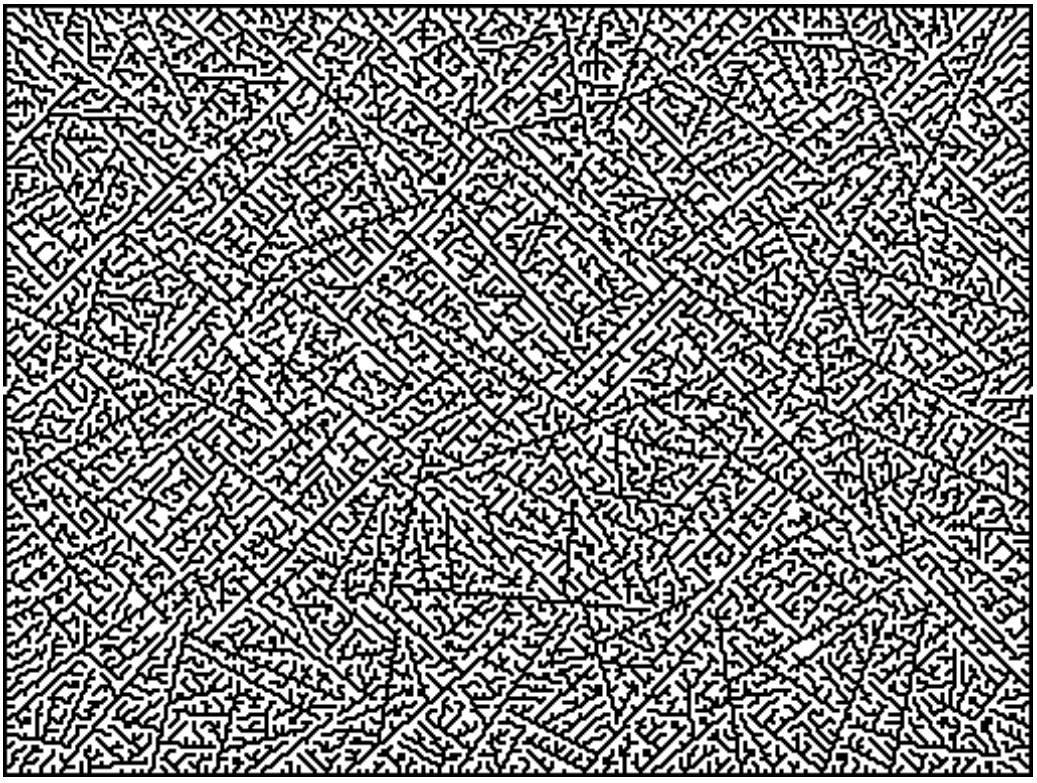
Obrázek B.5: Sigma teselace (převzato z [12])



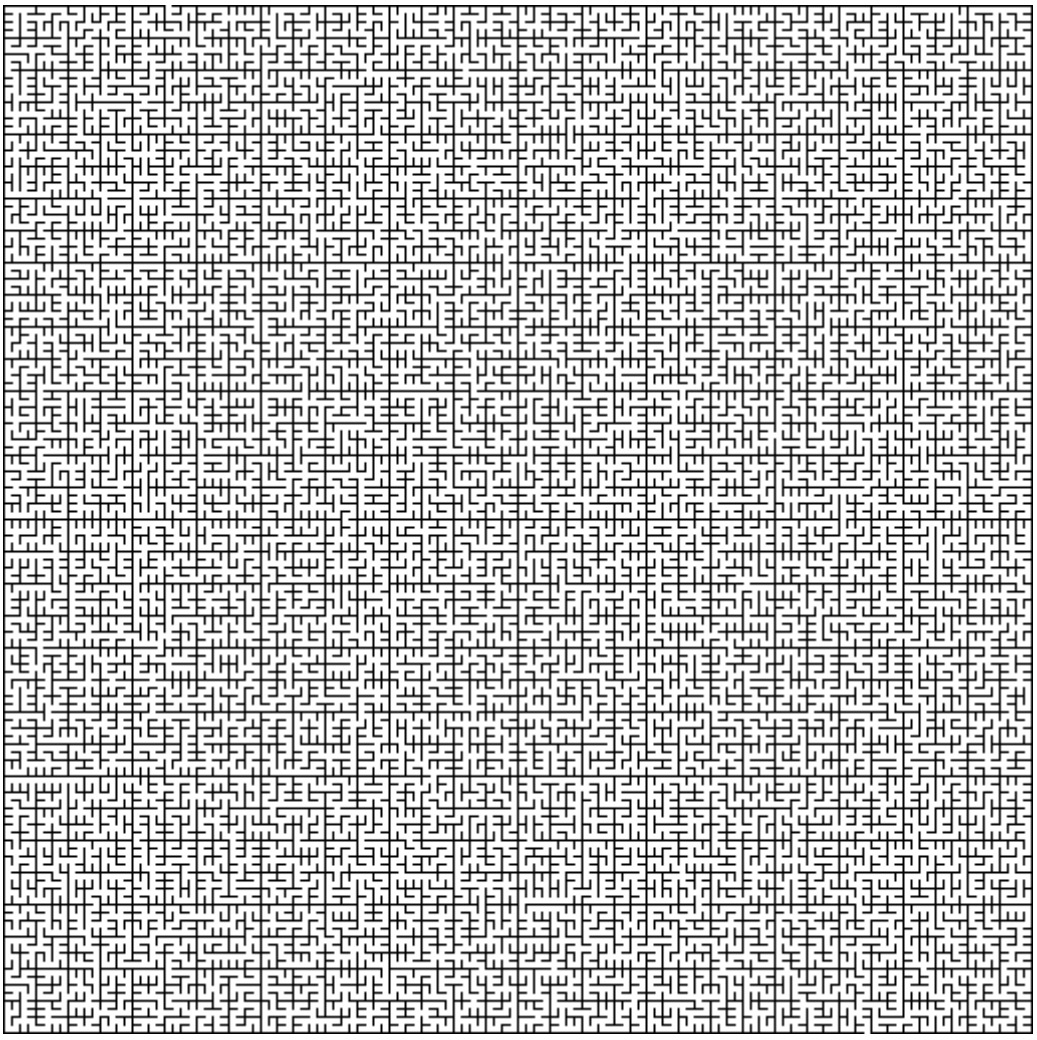
Obrázek B.6: Upsilon teselace (převzato z [12])



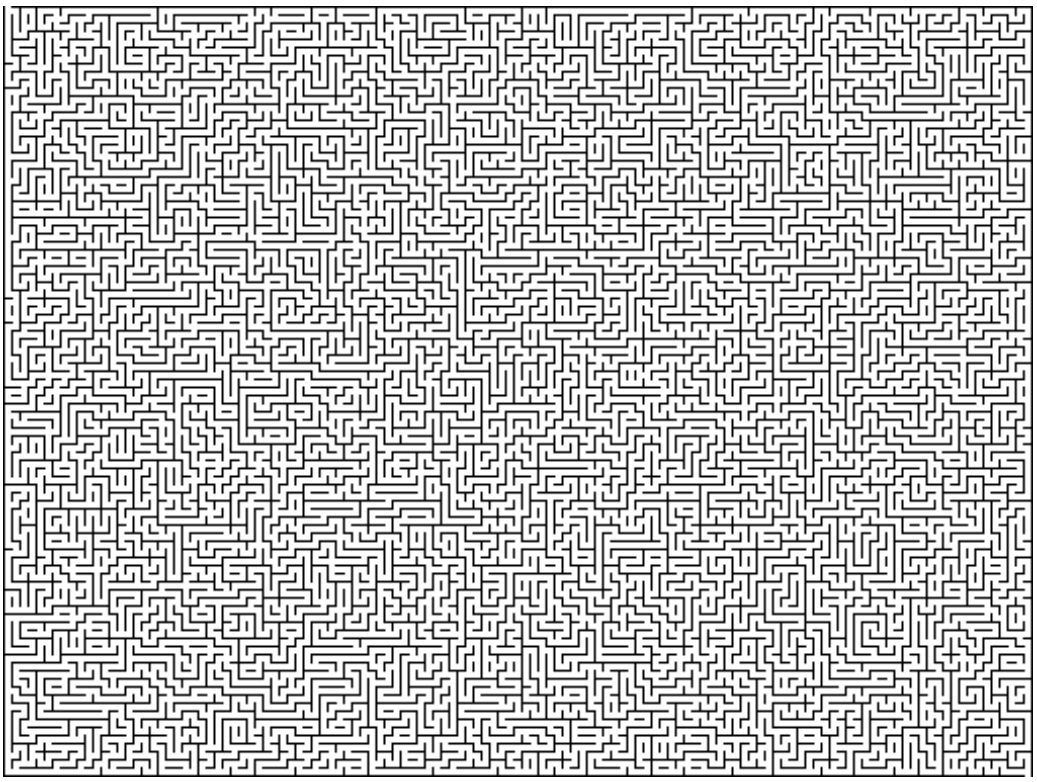
Obrázek B.7: Theta teselace (převzato z [12])



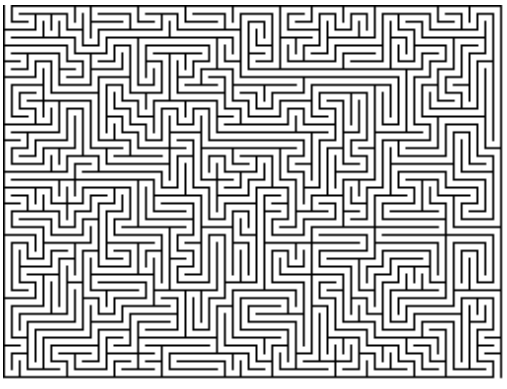
Obrázek B.8: Crack bludiště – atypická teselace (převzato z [12])



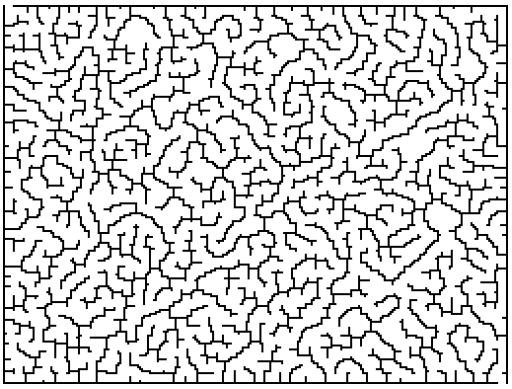
Obrázek B.9: Fraktálová teselace (převzato z [12])



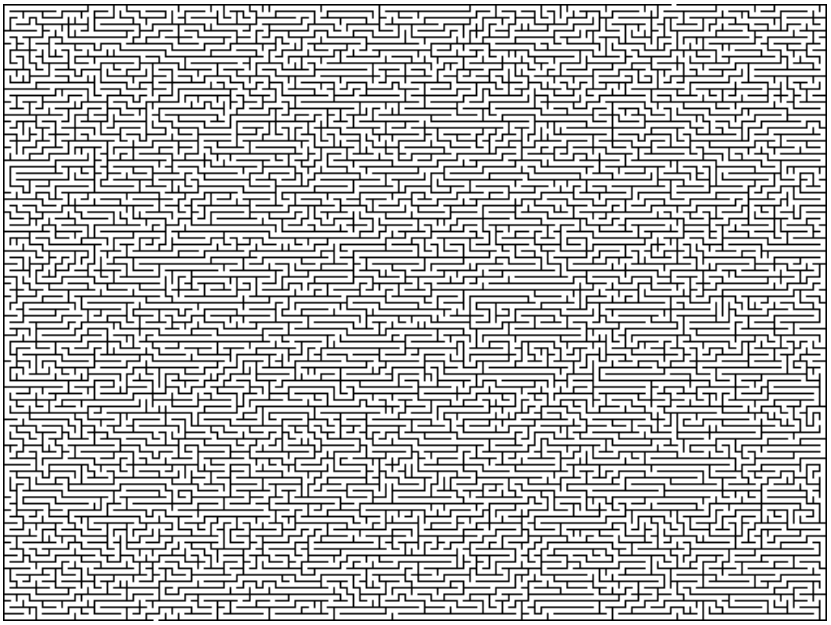
Obrázek B.10: Spletené bludiště (převzato z [12])



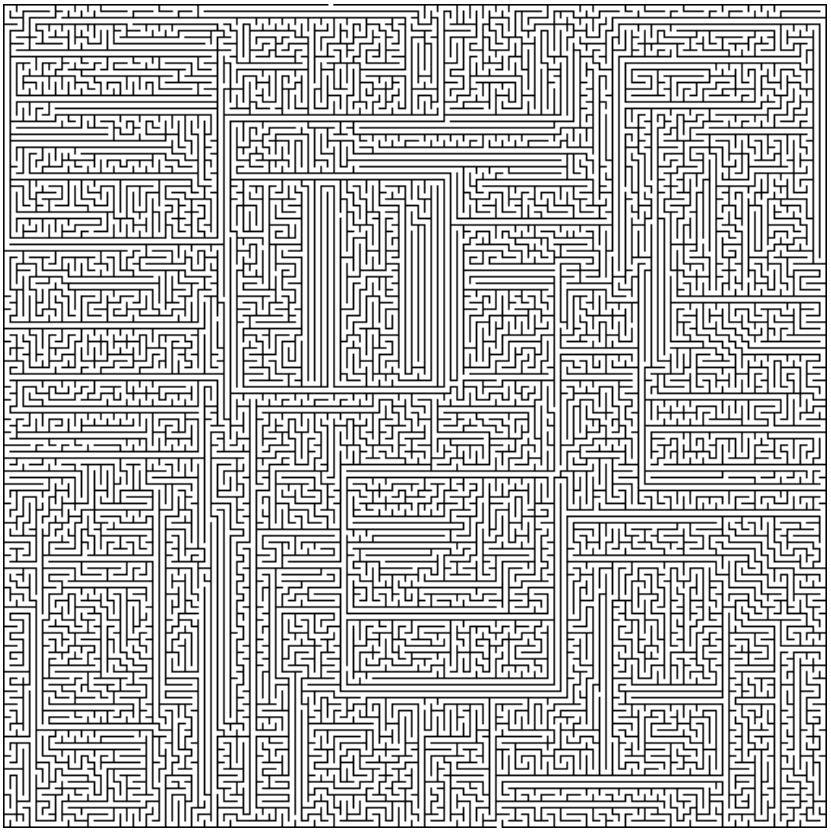
Obrázek B.11: Jednocestné bludiště (převzato z [12])



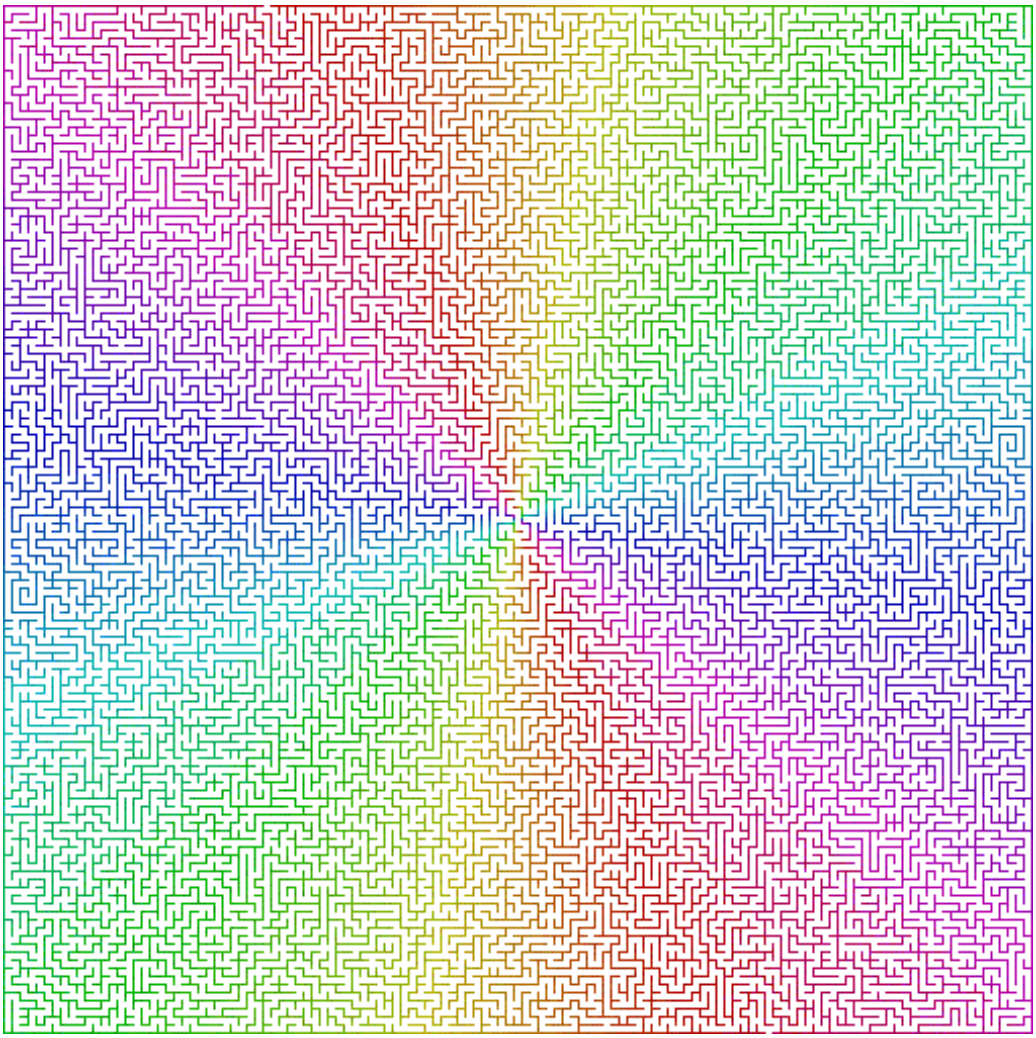
Obrázek B.12: Řídké bludiště (převzato z [12])



Obrázek B.13: Bludiště s horizontálním biasem (převzato z [12])



Obrázek B.14: Bludiště s vyšším faktorem run (převzato z [12])

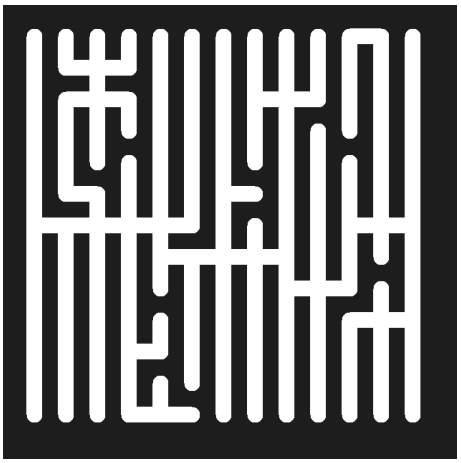


Obrázek B.15: Bludiště se středovou symetrií (převzato z [12])

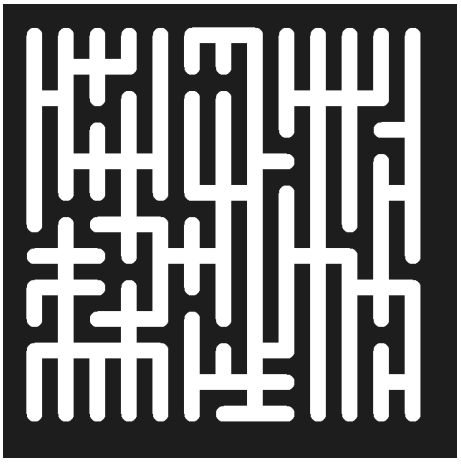
Modifikovaný Kruskalův algoritmus

C

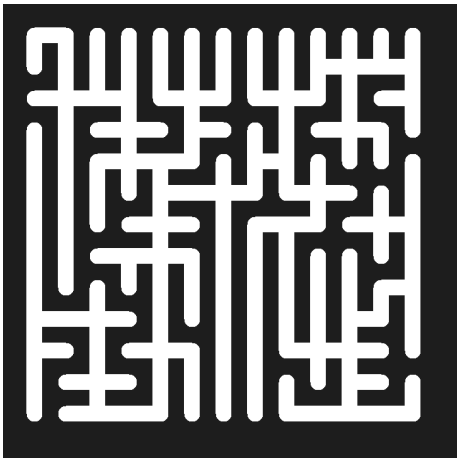
Obrázek C.1: Horizontální bias 0, vertikální bias 1



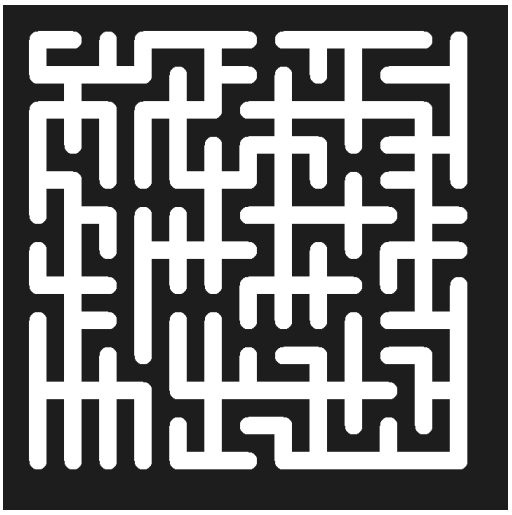
Obrázek C.2: Horizontální bias 0.1, vertikální bias 0.9



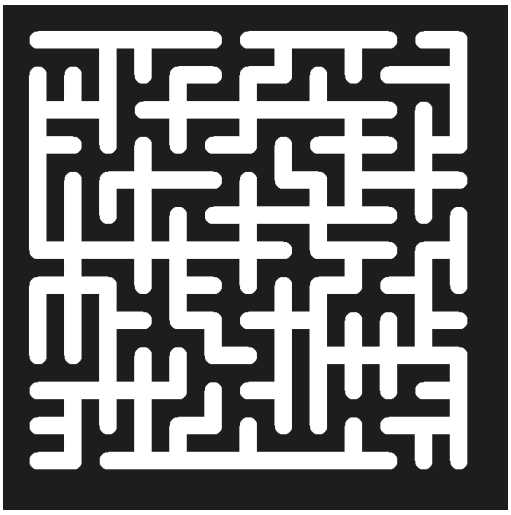
Obrázek C.3: Horizontální bias 0.2, vertikální bias 0.8



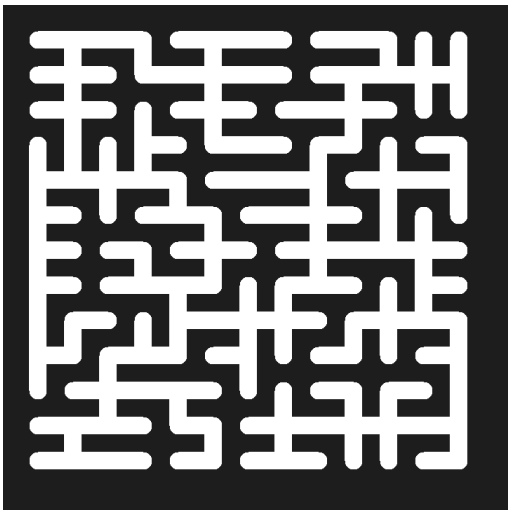
Obrázek C.4: Horizontální bias 0.3, vertikální bias 0.7



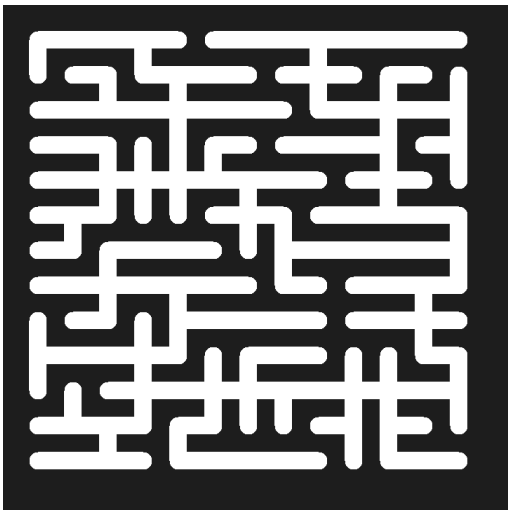
Obrázek C.5: Horizontální bias 0.4, vertikální bias 0.6



Obrázek C.6: Horizontální bias 0.5, vertikální bias 0.5

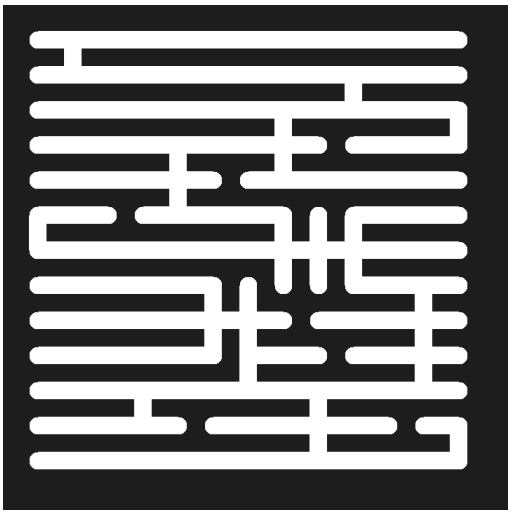


Obrázek C.7: Horizontální bias 0.6, vertikální bias 0.4

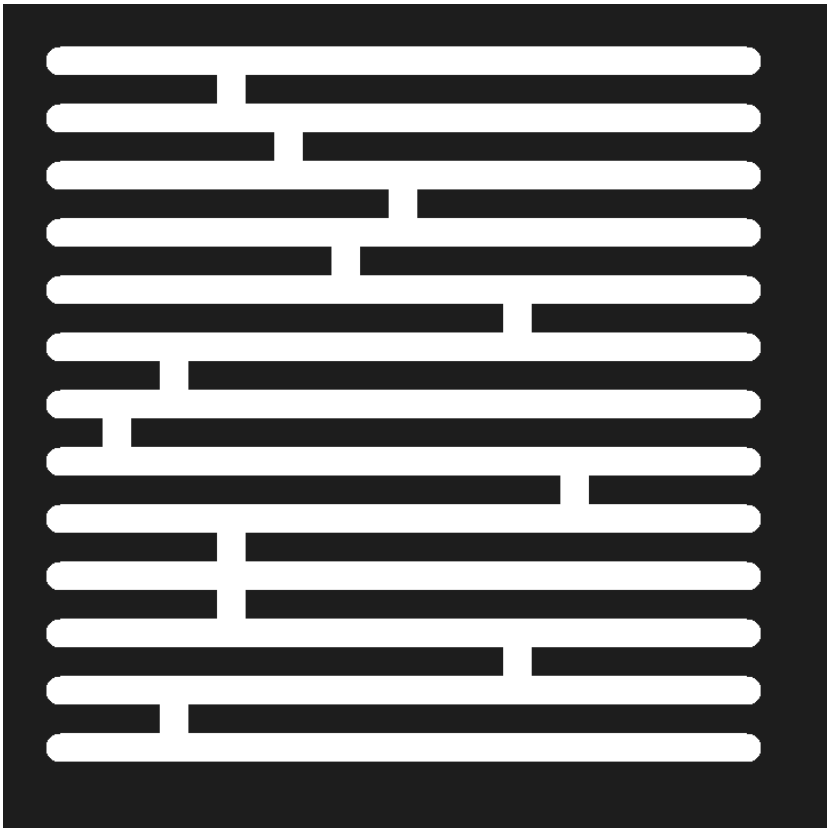


Obrázek C.8: Horizontální bias 0.7, vertikální bias 0.3

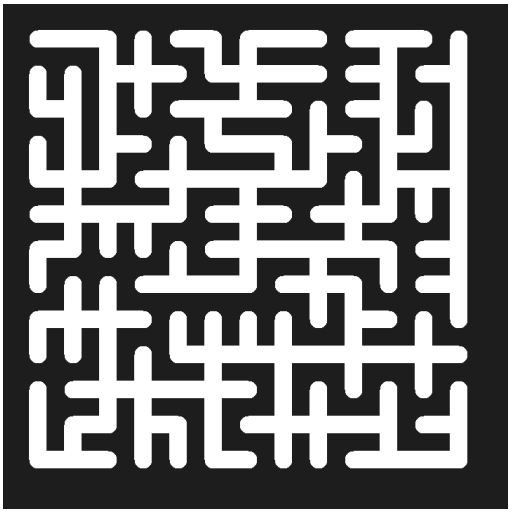
Obrázek C.9: Horizontální bias 0.8, vertikální bias 0.2



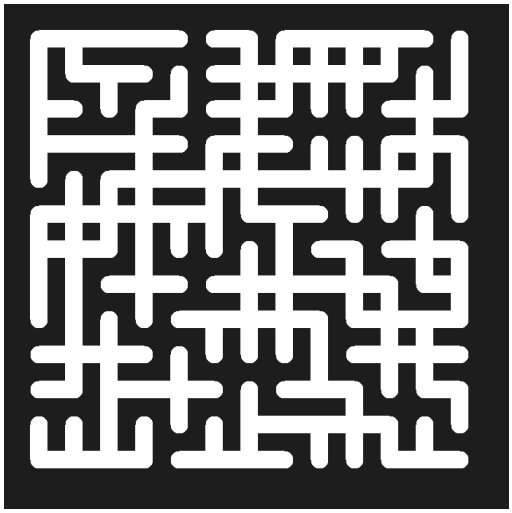
Obrázek C.10: Horizontální bias 0.9, vertikální bias 0.1



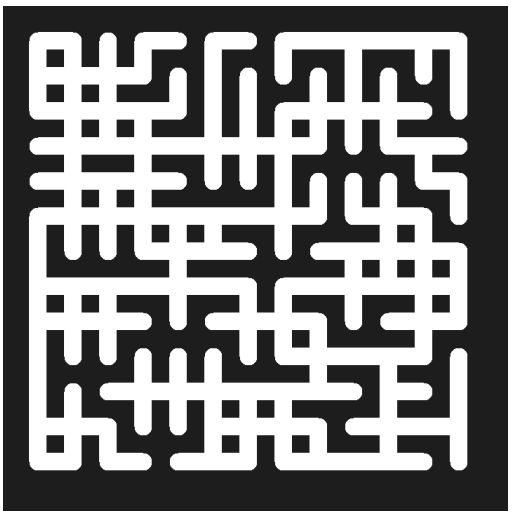
Obrázek C.11: Horizontální bias 1, vertikální bias 0



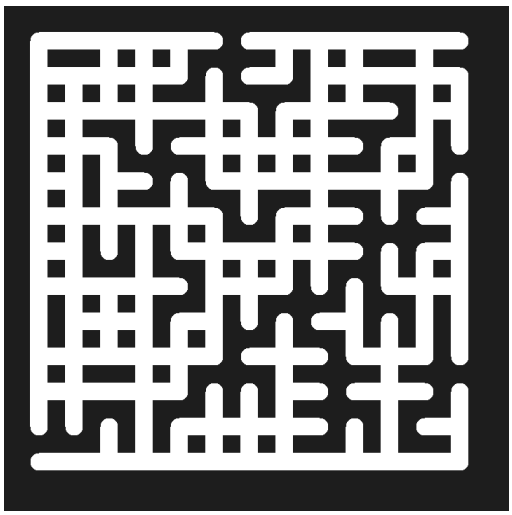
Obrázek C.12: Cyklický bias 0



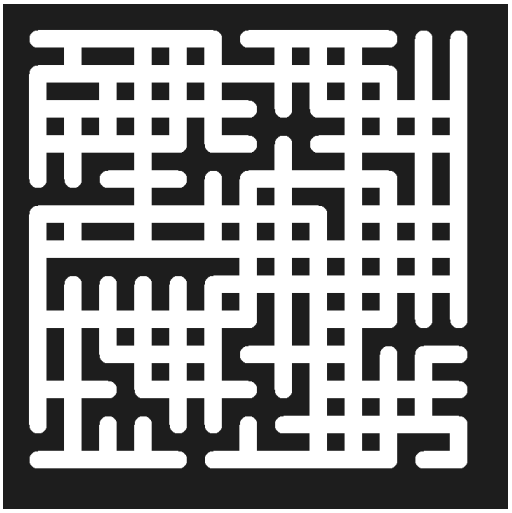
Obrázek C.13: Cyklický bias 0.1



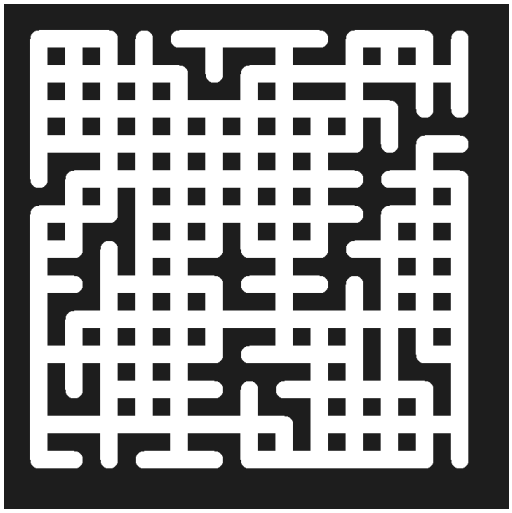
Obrázek C.14: Cyklický bias 0.2



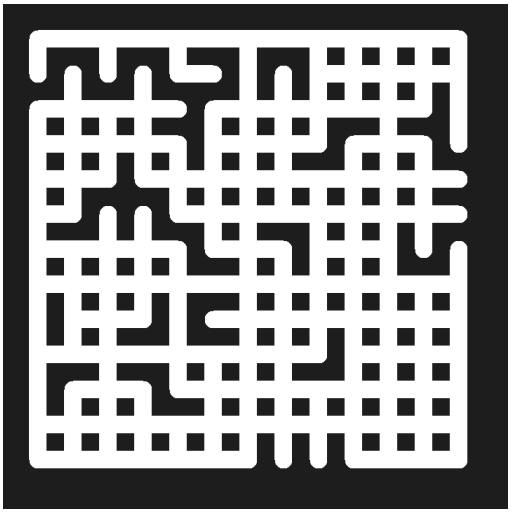
Obrázek C.15: Cyklický bias 0.3



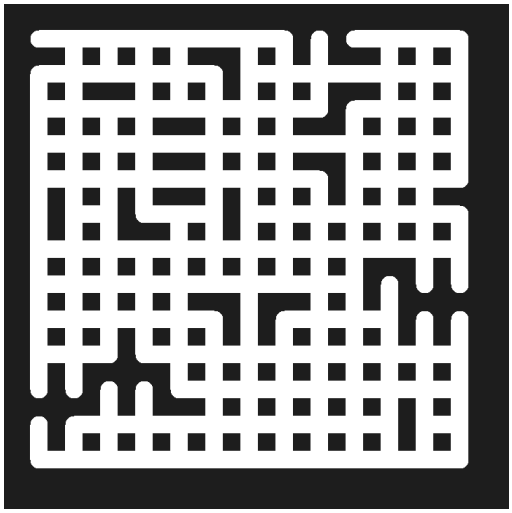
Obrázek C.16: Cyklický bias 0.4



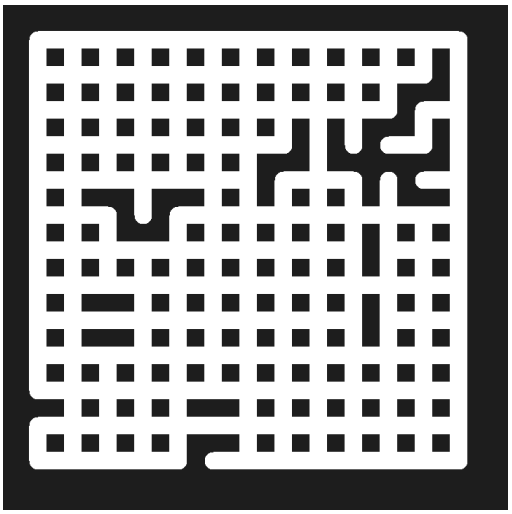
Obrázek C.17: Cyklický bias 0.5



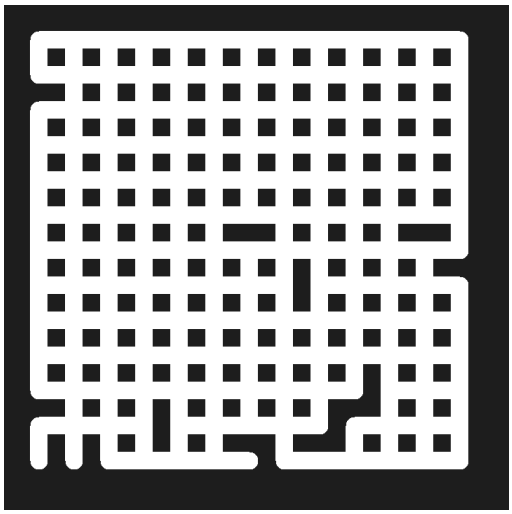
Obrázek C.18: Cyklický bias 0.6



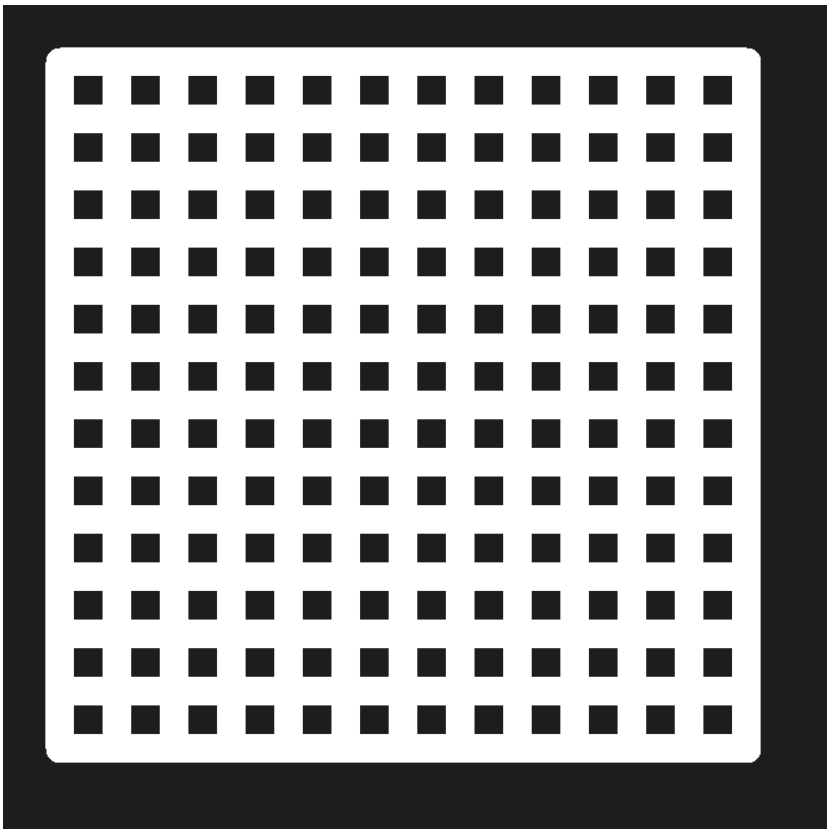
Obrázek C.19: Cyklický bias 0.7



Obrázek C.20: Cyklický bias 0.8



Obrázek C.21: Cyklický bias 0.9



Obrázek C.22: Cyklický bias 1

Tabulka výsledků z experimentu měření počtu generací a řešitelnosti

D

Tabulka D.1: Tabulka všech naměřených hodnot

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Ortogonální 8–okolí B3/S12345			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	1	36	0
4	23	33.1739	4.64994
5	36	31.9722	5.95579
6	29	28.1379	5.16436
7	36	23.0278	4.34285
8	49	20.2857	4.25705
9	44	16.5227	4.53516
10	54	11.7963	3.83637
11	56	7.35714	2.72179
12	56	3.69643	2.04345
-1	51	2.90196	1.51146

(tabulka pokračuje na další stránce)

Tabulka D.1 *(pokračování z předchozí stránky)*

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Ortogonální 8–okolí B3/S1234			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	24	40.1667	4.43158
4	42	38.9286	4.32266
5	52	36.5385	5.66182
6	58	30.9483	5.49113
7	54	27.537	5.61649
8	52	23.1154	4.81845
9	48	18.9375	4.44131
10	49	13.9592	3.61944
11	59	10.5254	3.82833
12	55	8.03636	2.95397
-1	65	5.49231	2.9723
Ortogonální 8–okolí B37/S12345			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	2	41	0
4	50	39.32	6.23679
5	50	33.6	7.09084
6	65	29.0769	5.46403
7	71	25.1549	5.2612
8	64	21.1406	4.17158
9	66	17.0303	5.33703
10	59	12.8983	3.8475
11	72	8.61111	3.03935
12	72	4.86111	2.70445
-1	55	3.78182	2.57746

(tabulka pokračuje na další stránce)

Tabulka D.1 *(pokračování z předchozí stránky)*

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Ortogonální 8–okolí B37/S1234			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	24	43.4167	2.89995
4	67	40.806	5.43685
5	60	36.3333	5.34062
6	66	32.4242	5.90085
7	66	28.4545	4.84981
8	66	23.8939	5.60338
9	65	19.6308	5.2347
10	61	14.4754	3.58323
11	64	11.875	4.75493
12	80	8.525	4.00617
-1	77	7.49351	4.57158
Hexagonální Hexagonální B24/S12345			
0	0	nan	nan
1	0	nan	nan
2	90	27.4667	1.18509
3	92	27.3261	3.34959
4	98	24.7245	3.36786
5	97	21.7423	2.46743
6	100	18.25	2.2644
7	98	15.8878	2.33379
8	98	12.602	2.14638
9	99	10.2727	2.05882
10	100	6.56	1.80732
11	100	3.92	1.86376
12	100	1.61	1.05731
-1	100	1.27	0.785557

(tabulka pokračuje na další stránce)

Tabulka D.1 *(pokračování z předchozí stránky)*

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Hexagonální Hexagonální B24/S1234			
0	0	nan	nan
1	0	nan	nan
2	90	27.6	1.49666
3	93	26.2796	1.85707
4	100	23.8	2.0199
5	100	21.45	2.08507
6	100	18.36	2.41462
7	99	16.0808	2.29482
8	98	12.7551	1.93286
9	99	10.101	1.81183
10	100	7.14	2.31093
11	99	4.29293	1.84912
12	100	1.56	1.05186
-1	99	1.29293	0.781769
Hexagonální Hexagonální B24/S2345			
0	0	nan	nan
1	0	nan	nan
2	70	28.1571	1.34839
3	94	27.4468	2.55
4	99	25.1515	3.59956
5	97	22.5773	2.94917
6	99	19.697	3.13164
7	100	17.12	2.68805
8	98	13.6735	2.76559
9	99	11.3434	2.45016
10	100	7.43	2.4218
11	99	4.82828	2.387
12	100	2.02	1.00975
-1	100	1.71	1.25136

(tabulka pokračuje na další stránce)

Tabulka D.1 *(pokračování z předchozí stránky)*

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Hexagonální 8–okolí B23/S12345			
0	0	nan	nan
1	0	nan	nan
2	87	19.5632	0.495987
3	100	18.04	0.904655
4	99	15.9394	0.874073
5	99	14.0808	0.981411
6	98	11.8469	0.87309
7	98	10	0.92582
8	97	7.81443	0.841328
9	100	5.88	0.908625
10	97	3.90722	0.825901
11	96	1.88542	0.888231
12	99	1.26263	0.523891
-1	98	1.18367	0.559645
Hexagonální 8–okolí B3/S1234			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	30	39.0667	4.85066
4	49	36.5306	4.57635
5	66	33.6364	5.14822
6	78	29.1154	5.48887
7	72	24.9306	4.55926
8	69	19.8841	3.54898
9	76	16.8289	3.7254
10	76	12.25	3.92738
11	88	8.10227	3.17318
12	80	5.3125	2.62961
-1	88	3.97727	2.08881

(tabulka pokračuje na další stránce)

Tabulka D.1 *(pokračování z předchozí stránky)*

Velikost inicializační konstanty	Počet vygenerovaných bludišť	Průměrný počet generací	Standardní odchylka počtu generací
Ortogonální Hexagonální B3/S234			
0	0	nan	nan
1	0	nan	nan
2	0	nan	nan
3	0	nan	nan
4	0	nan	nan
5	0	nan	nan
6	0	nan	nan
7	0	nan	nan
8	0	nan	nan
9	0	nan	nan
10	0	nan	nan
11	0	nan	nan
12	46	5.73913	3.33255
-1	59	4.1017	2.98978
Ortogonální Hexagonální B2/S234			
0	0	nan	nan
1	0	nan	nan
2	67	35.8209	1.18354
3	57	34.2807	5.1801
4	30	28.9	5.37494
5	33	26.2727	4.34339
6	24	22.625	4.0601
7	46	20.1087	3.11538
8	40	17.1	3.1607
9	39	13.1026	3.31117
10	44	10.7955	3.20213
11	45	7.75556	2.64286
12	57	5.77193	3.03209
-1	46	5.43478	2.90915

Bibliografie

[1] Andrew Adamatzky. *Game of life cellular automata*. Sv. 1. Springer, 2010. ISBN: 978-1-84996-216-2.

[2] V. Bellot et al. „How to generate perfect mazes?“ In: *Information Sciences* 572 (2021), s. 444–459. ISSN: 0020-0255. DOI: 10.1016/j.ins.2021.03.022. URL: https://doi.org/10.1016/j.ins.2021.03.022.

[3] David Eppstein. *Cellular Automata rules lexicon*. 2001. URL: https://web.archive.org/web/20071117041941/http://www.mirwoj.opus.chelm.pl/ca/rullex_life.html.

[4] Jiří Demel. *Grafy a jejich aplikace*. Libčice nad Vltavou: Jiří Demel, 2015. ISBN: 9788026076841.

[5] P. Gabrovšek. *Analysis of Maze Generating Algorithms*. 2019. URL: http://ipsitransactions.org/journals/papers/tir/2019jan/p5.pdf.

[6] Alan Gibbons. *Algorithmic graph theory*. New York, NY, USA: Cambridge University Press, 1994. ISBN: 0521288819.

[7] MAHYUS IHSAN, Dedi Suhaimi, MARWAN RAMLI, Siti Yuni a Ikhsan Maulidi. „Non-perfect maze generation using Kruskal algorithm“. In: *Jurnal Natural* 21.1 (ún. 2021), s. 11. ISSN: 1411-8513. DOI: 10.24815/jn.v21i1.18840. URL: http://doi.org/10.24815/jn.v21i1.18840.

[8] Navin Kumar a Sandeep Kaur. „A Review of Various Maze Solving Algorithms Based on Graph Theory“. In: *International Journal for Scientific Research and Development* 6.1 (břez. 2019), s. 431–434. URL: https://www.researchgate.net/publication/331481380_A_Review_of_Various_Maze_Solving_Algorithms_Based_on_Graph_Theory.

[9] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010. ISBN: 978-0134610993.

[10] Adil M.J. Sadik, Maruf A. Dhali, Hasib M.A.B. Farid, Tafhim U. Rashid a A. Syeed. „A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory“. In: 1 (2010), s. 52–56. DOI: 10.1109/AICI.2010.18. URL: https://doi.org/10.1109/AICI.2010.18.

[11] Fred E. Szabo. *The Linear Algebra Survival Guide*. Boston: Academic Press, 2015. ISBN: 978-0-12-409520-5. DOI: https://doi.org/10.1016/B978-0-12-409520-5.50021-7. URL: https://www.sciencedirect.com/science/article/pii/B9780124095205500217.

[12] Walter D. Pullen. *Think Labyrinth!* 1996. URL: http://www.astrolog.org/labyrnth.htm.

[13] Huijuan Wang, Yuan Yu a Quanbo Yuan. „Application of Dijkstra algorithm in robot path-planning“. In: (2011), s. 1067–1069. DOI: 10.1109/MACE.2011.5987118. URL: https://doi.org/10.1109/MACE.2011.5987118.

Seznam obrázků

Seznam tabulek

Seznam výpisů