Generování bludišť

Projekt rešerše do předmětu GAL

16. ledna 2016

Autor: Vojtěch Havlena (xhavle03)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Pod pojmem bludiště lze chápat systém spletitých cest a slepých uliček [13]. Řešením bludiště je potom nalezení cesty z počátečního do koncového bodu. Často se pojem bludiště nahrazuje pojmem labyrint, nicméně labyrint ve většině případů označuje jednu klikatou cestu z počátečního do koncového bodu [11]. Labyrintům se v této práci dále věnovat nebudeme. Tato práce se zabývá algoritmy pro generování bludišť, zejména se jedná o algoritmy založené na teorii grafů.

Pravděpodobně primárním využitím bludišť je pobavení lidí. Řešení bludišť může být určeno pro lidi různých věkových kategorií. S rovinnými bludišti se můžeme setkat například v různých časopisech pro pobavení čtenářů. Bludiště se také často využívají v počítačových hrách, kde náhodně vytvořené prostředí, ve kterém se hráč pohybuje, lze považovat za určitý typ bludiště. Bludiště se také dají využít pro testování inteligence a orientačních schopností různých živočichů.

1.1 Prvky bludiště a typy bludišť

Základními stavebními prvky bludišť jsou zeď a buňka. Buňkou rozumíme základní element v bludišti. Zdi od sebe oddělují sousední buňky, mezi kterými neexistuje přímé propojení. Podle počtu propojení se sousedními buňkami dělíme buňky na: izolované (0 propojení), slepý konec (1 propojení) buňka součástí cesty (2 propojení) a křižovatka (3 a více propojení). Cestou v bludišti pak rozumíme souvislou posloupnost propojených buněk.

Bludiště se mohou rozdělovat podle několika hledisek. Prvním takovým dělením je dělení podle typu dimenze. Základními typy jsou dvoudimenzionální a třídimenzionální bludiště [12]. U více dimenzionálních bludišť nastává problém s vizualizací takového bludiště. Proto se často více dimenzionální bludiště zobrazují jako dvou, popř. třídimenzionální bludiště spojená portály[1]. V této práci se však budeme zabývat zejména dvoudimenzionálními bludišti.

Dále je možné uvažovat dělení podle vnitřní struktury. Zde rozlišujeme tzv. perfektní bludiště, pro které platí, že mezi libovolnými dvěma body vede vždy jen jedna cesta. Pokud bludiště obsahuje cykly, ale neobsahuje slepé konce, jedná se o spletené bludiště. V případě, že spletené bludiště obsahuje slepé konce, hovoříme o něm jako o částečně spleteném bludišti. Nakonec, pokud bludiště obsahuje oddělené části, nazývá se řídké [12].

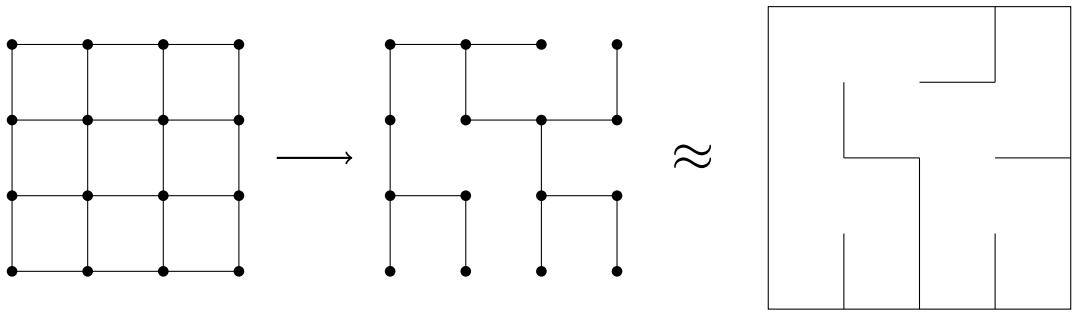
Posledním zmíněným dělením, které zohledňuje především vzhled bludiště, je dělení podle typu mozaiky [12]. Toto dělení se zaměřuje na vzhled buněk. Základním a jediným typem, který je v práci uvažován, jsou ortogonální bludiště, tedy bludiště se čtvercovými buňkami. Dále je možné uvažovat trojúhelníkové buňky (delta bludiště), šestiúhelníkové buňky (sigma bludiště) a jiné.

V další části práce se již budeme věnovat samotným algoritmům pro generování různých typů bludišť. Také se budeme zabývat souvislostí mezi generováním bludišť a teorií grafů.

[1] Lze si představit jako teleporty, které přenášejí procházející objekt mezi bludišti.

2 Generování založené na teorii grafů

V této kapitole se budeme zabývat generováním bludišť s využitím teorie grafů. V případě, že budeme uvažovat ortogonální (pravoúhlá) bludiště, je možné reprezentovat strukturu bludiště pomocí rovinného neorientovaného grafu (mřížky) [8]. Každý uzel grafu reprezentuje jednu buňku bludiště. Hrany grafu potom představují přechody mezi sousedními buňkami v bludišti. Příklad této reprezentace je na obr. 1. Hlavní výhodou této reprezentace je abstrakce od grafického zobrazení bludiště (velikost, tvar buněk apod.).



Obrázek 1: Neorientovaný graf reprezentující bludiště o rozměru 4×4 bez zdí. Vynecháním některých hran vznikne graf odpovídající bludišti na posledním obrázku.

Pro generování bludišť založené na teorii grafů se vychází z generování perfektního bludiště. Pro zopakování, perfektní bludiště je bludiště, ve kterém mezi libovolnými dvěma buňkami existuje právě jedna cesta. Tato definice odpovídá definici stromu v teorii grafů. Využijeme-li reprezentaci bludiště popsanou výše, generování perfektního bludiště odpovídá hledání kostry grafu. V další části se tedy budeme věnovat modifikacím algoritmů pro hledání kostry grafu tak, aby byly schopny generovat perfektní bludiště.

2.1 Randomizovaný Primův algoritmus

První představený algoritmus použitelný pro generování perfektního bludiště je randomizovaný Primův algoritmus [3]. Jak již ale bylo zmíněno dříve, bludiště reprezentujeme pomocí neorientovaného neohodnoceného grafu. Proto si graf reprezentující bludiště můžeme představit tak, že každé hraně je přiřazena konstantní váha 1 a Primův algoritmus tedy může hrany do minimální kostry vybírat náhodně.

Varianta tohoto algoritmu používaná pro generování bludišť využívá rozdělování buněk (uzlů) do tří množin – uvnitř (U), pomezí (P) a mimo (M). Množina U obsahuje buňky, které již jsou součástí bludiště. Množina M obsahuje buňky, které ještě nejsou součástí bludiště a nakonec množina P obsahuje buňky, které ještě nejsou součástí bludiště, ale mají alespoň jednoho souseda v množině U.

Na začátku algoritmu se po inicializaci vybere počáteční buňka a ta se přesune do množiny U. Všichni její sousedé se přesunou do množiny P. Následně se opakovaně vybírají buňky z množiny P, dokud tato množina není prázdná. Každá vybraná

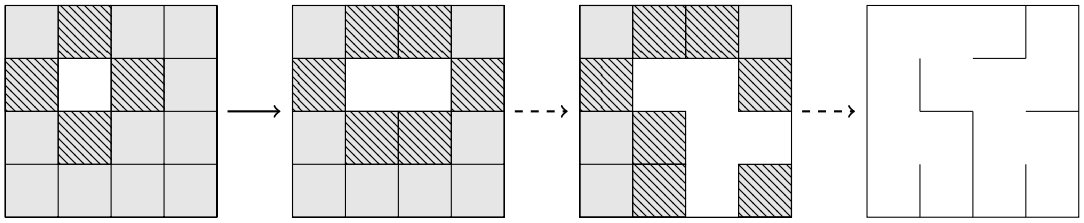
buňka se přesune do množiny U, poznačí se možnost přechodu v bludišti mezi vybranou buňkou a sousední buňkou, která je již v množině U (přidá se hrana mezi odpovídajícími uzly do grafu bludiště). Následně se aktualizuje obsah množin P a M [8, 3]. Příklad generování bludiště je na obr. 2. Zápis v pseudokódu je následující:

Algoritmus 1: Modifikovaný randomizovaný Primův algoritmus

Vstup: Graf $G = (V, E)$ reprezentující bludiště bez vnitřních zdí.
Výstup: Perfektní bludiště

1: Do množiny M ulož všechny buňky bludiště ($M \leftarrow V$), $P \leftarrow \emptyset$, $U \leftarrow \emptyset$.
2: Náhodně vyber počáteční buňku bludiště s a přidej ji do množiny U.
3: Přidej do množiny P všechny sousední buňky buňky s.
4: **while** $P \neq \emptyset$ **do**
5: Vyber náhodně buňku c_P z množiny P
6: Vyber náhodně buňku $c_U \in U$ takovou, že c_U je sousední buňka c_P.
7: $U \leftarrow U \cup \{c_P\}$
8: Přesuň všechny sousední buňky c_P, které jsou v množině M z množiny M do P.
9: Poznač spojení v bludišti mezi buňkami c_P a c_U.
10: $P \leftarrow P \setminus \{c_P\}$.
11: **end**

Tento algoritmus lze upravit tak, aby generoval i bludiště obsahující cykly (tedy ne již perfektní bludiště). Modifikace spočívá v tom, že se náhodně vybraná buňka c_P z množiny P s určitou pravděpodobností z množiny P neodebere (úprava řádku 10). Opětovným výběrem této buňky potom vznikne v bludišti cyklus [8].

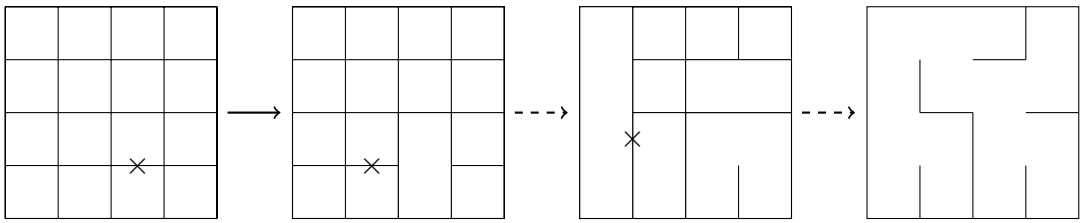


Obrázek 2: Příklad generování bludiště randomizovaným Primovým algoritmem. Bílou barvou jsou označeny buňky v množině U, plnou šedou barvou buňky v množině M a čárkovaně buňky v P.

2.2 Randomizovaný Kruskalův algoritmus

Další algoritmus, použitelný pro generování perfektního bludiště, je randomizovaný Kruskalův algoritmus [3]. Vzhledem k tomu, že graf reprezentující bludiště je neohodnocený neorientovaný graf, rozšíříme si tento graf o konstantní ohodnocení 1 každé hraně (podobně jako v případě randomizovaného Primova algoritmu). Jediná modifikace oproti původnímu Kruskalově algoritmu je, že hrany se z množiny dosud nepoužitých hran vybírají náhodně.

V prvním kroku algoritmu se provede vytvoření množin pro každou buňku (resp. uzel v grafu). V dalším kroku se opakovaně náhodně vybírá zeď (resp. hrana v grafu) z množiny zdí, dokud množina zdí není prázdná. Pokud jsou obě buňky, které jsou touto zdí odděleny, v různých množinách, obě tyto množiny se sjednotí a poznačí se cesta mezi těmito buňkami. V případě, že buňky jsou ve stejných množinách, není možné zeď odstranit. Vybraná zeď je posléze odstraněna z množiny zdí. Příklad generování bludiště je na obr. 3.



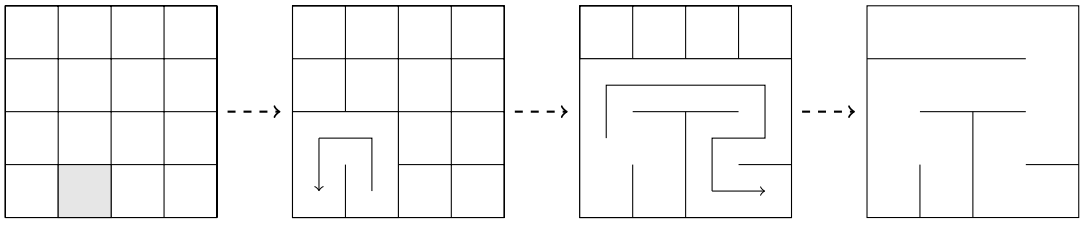
Obrázek 3: Příklad generování bludiště randomizovaným Kruskalovým algoritmem.

Podobně jako v případě randomizovaného Primova algoritmu, i tento algoritmus lze modifikovat pro generování bludišť obsahující cykly. Modifikace spočívá v tom, že pokud nějakou zeď nemůžeme odebrat, tak ji s určitou pravděpodobností přesto odebereme. Tím pádem nám vzniknou cykly [8].

2.3 Prohledávání do hloubky

Dalším algoritmem, který lze použít pro generování kostry grafu a tedy i perfektního bludiště, je prohledávání do hloubky (DFS) [8, 3]. Vstupem algoritmu DFS je graf reprezentující bludiště bez vnitřních stěn. Vzhledem k tomu, že zmíněný graf je souvislým grafem, je i les prohledávání do hloubky souvislým grafem a tedy stromem.

Jediný rozdíl oproti klasickému algoritmu DFS je způsob volby buněk (uzlů) pro další procházení. Aby bludiště vypadalo co nejvíce nepravidelně, je počáteční buňka zvolena náhodně. Stejně tak sousední buňky jsou v proceduře DFS-VISIT [6] vybírány v náhodném pořadí. Nevýhodou tohoto algoritmu je použití rekurze, což zejména pro generování velkých bludišť způsobuje problémy. Proto se pro generování častěji využívá modifikace DFS nepoužívající rekurzi, ale využívající vlastní zásobník. Příklad generování bludiště je uveden na obr. 4.



Obrázek 4: Příklad generování bludiště algoritmem DFS. Šedou barvou je označena počáteční buňka.

Výhodou tohoto algoritmu je lineární časová složitost (vzhledem k reprezentaci grafu bludiště). Nevýhodou je fakt, že ne všechny perfektní bludiště je algoritmem DFS možné vygenerovat. Například bludiště na obr. 1 není možné tímto algoritmem vygenerovat.

2.4 Aldous–Broderův algoritmus

Další algoritmus, který lze použít pro generování perfektního bludiště, úzce souvisí s pojmem rovnoměrná kostra grafu (uniform spanning tree). Rovnoměrná kostra grafu je kostra grafu, která byla náhodně (a se stejnou pravděpodobností) vybrána ze všech možných koster daného grafu [8]. Pro hledání rovnoměrných koster lze využít právě Aldous–Broderův algoritmus. Rovnoměrné kostry mají využití např. v kombinatorice nebo ve fyzice (konkrétně v oblasti kvantové teorie pole) [5].

Vstupem algoritmu je neorientovaný souvislý graf. Výstupem je rovnoměrná kostra zadaného grafu. Principem algoritmu je náhodné procházení grafu. Na začátku algoritmu se náhodně zvolí libovolný uzel grafu u a tento uzel označ jako navštívený. V dalším kroku náhodně vyber jeden sousední uzel v uzlu u, který je s uzlem u spojen hranou. Pokud uzel v ještě nebyl navštíven, přidej hranu (u, v) do kostry grafu. Nakonec se přesuň do uzlu v, tedy $u \leftarrow v$. Následně pokračuj na krok výběru jednoho sousedního uzlu a tento výběr opakuj dokud nejsou všechny uzly grafu označeny jako navštívené [2, 8].

Popsaný algoritmus lze snadno modifikovat pro generování bludiště [8]. Vstupem algoritmu bude graf reprezentující bludiště bez vnitřních zdí. Algoritmus pro generování popsaný v pseudokódu je potom následující.

Algoritmus 2: Aldous–Broderův algoritmus

Vstup: Graf $G = (V, E)$ reprezentující bludiště bez vnitřních zdí.
Výstup: Perfektní bludiště.

1: Náhodně vyber buňku bludiště u.
2: Inicializuj množinu navštívených buněk $Vis \leftarrow \emptyset$.
3: **repeat**
4: $\quad Vis \leftarrow Vis \cup \{u\}$
5: $\quad$ Náhodně vyber sousední buňku u' buňky u.
6: $\quad$ **if** $u' \notin Vis$ **then**
7: $\quad\quad$ Poznač spojení mezi buňkami u a u'.
8: $\quad$ **end**
9: $\quad u \leftarrow u'$
10: **until** $|Vis| = |V|$;

Výhodou Aldous–Broderova algoritmu je jeho jednoduchost a možnost generovat libovolné perfektní bludiště. Nevýhodou je jeho značná neefektivnost.

Kromě Aldous–Broderova algoritmu lze pro hledání rovnoměrných koster grafu využít také Wilsonův algoritmus. Wilsonův algoritmus provede v daném grafu náhodnou procházku a odstraní cykly vzniklé při této procházce. Na rozdíl od Aldous–Broderova algoritmu má ale Wilsonův algoritmus zaručenou polynomiální časovou složitost [5].

2.5 Algoritmus Hunt and Kill

Poslední představený algoritmus, založený na teorii grafů, který lze použít pro generování perfektního bludiště, je algoritmus Hunt and Kill [3]. Algoritmus Hunt and Kill se dá považovat za jistou modifikaci prohledávání do hloubky.

Algoritmus provádí průchod grafem s náhodným výběrem nenavštíveného sousedního uzlu, do kterého se bude pokračovat. V případě, že žádný nenavštívený sousední uzel neexistuje, na rozdíl od prohledávání do hloubky (backtrackingu), kde se provede návrat k předcházejícímu uzlu, v případě algoritmu Hund and Kill se provede návrat do libovolného již navštíveného uzlu, který obsahuje alespoň jednoho nenavštíveného souseda. Z tohoto uzlu se potom pokračuje v průchodu grafu.

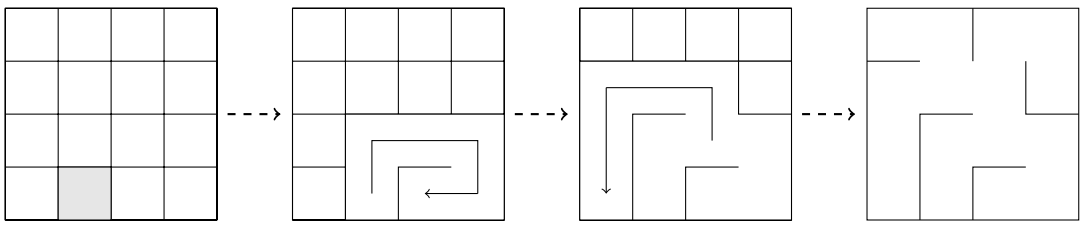
Algoritmus v pseudokódu přizpůsobený pro generování bludišť je potom zapsán následovně.

Algoritmus 3: Algoritmus Hunt and Kill

Vstup: Graf $G = (V, E)$ reprezentující bludiště bez vnitřních zdí.
Výstup: Perfektní bludiště.

1: Náhodně vyber buňku bludiště u.
2: Inicializuj množinu navštívených buněk $Vis \leftarrow \{u\}$.
3: **while** $|Vis| \neq |V|$ **do**
4: **if** *u má nenavštívené sousední buňky* **then**
5: Náhodně vyber buňku v z množiny nenavštívených sousedních buněk u.
6: Poznač spojení mezi buňkami u a v.
7: $u \leftarrow v$
8: $Vis \leftarrow Vis \cup \{u\}$
9: **else**
10: Náhodně vyber buňku u z množiny již navštívených buněk, které mají alespoň jednu nenavštívenou sousední buňku.
11: **end**
12: **end**

Výhodou tohoto algoritmu oproti algoritmu DFS je možnost generovat bludiště, které pomocí DFS vygenerovat nelze. Např. bludiště na obr. 5 není možné algoritmem DFS vygenerovat. Nicméně na druhou stranu existují i perfektní bludiště, které není možné algoritmem Hunt and Kill vygenerovat.



Obrázek 5: Příklad generování bludiště algoritmem Hunt and Kill. Šedou barvou je označena počáteční buňka.

2.6 Další algoritmy pro generování bludiště a jejich aplikace

Mimo výše uvedené algoritmy, které patří k nejvyužívanějším algoritmům pro generování bludiště, lze bludiště generovat i dalšími algoritmy, které jsou založeny na grafových algoritmech. Jedním z nich je např. algoritmus, který z grafu bludiště bez vnitřních zdí generuje binární strom, který potom reprezentuje výsledné bludiště [8]. Jiným přístupem může být algoritmus založený na růstu kolonií bakterií [8]. Mimo zmíněné grafové algoritmy modifikované pro generování bludišť, lze využít i jiných přístupů, které jsou upraveny pro konkrétní aplikaci.

Generování bludišť v počítačových hrách

Tvorba prostředí v počítačových hrách je jedním z hlavních využití generování bludišť. Z toho důvodu se využívají algoritmy, které jsou pro tuto aplikaci přizpůsobeny. V případě generování bludišť pro počítačové hry se výsledné bludiště charakterizuje pomocí atributů jako: počet pokojů, chodeb, křižovatek, slepých konců apod. Proto se tato bludiště negenerují pomocí algoritmů popsaných v předchozích podkapitolách, ale využívá se přístupů, které právě tyto atributy umí zohlednit. Jedním z takových přístupů je generování pomocí celulárních automatů [10]. Celulární automat je určitý matematický nebo fyzikální model, který se používá k modelování systémů. Pravidla pro celulární automat, který posléze generuje bludiště, jsou vyvinuty pomocí genetického algoritmu, který zohledňuje požadované vlastnosti kladené na výsledné bludiště. V případě, že je množina pravidel celulárního automatu již vyvinuta, může být dále prováděno generování bludišť s podobnými vlastnostmi v reálném čase (podrobněji viz [10]).

Generování třírozměrných bludišť

Doposud jsme se věnovali algoritmům, které generují dvoudimenzionální bludiště. Lze však nelézt i algoritmy, které generují více dimenzionální bludiště, konkrétně třídimenzionální bludiště [1]. Bludiště je v tomto případě reprezentováno krychlí. Celá krychle je potom rozdělena na 27 základních stavební prvků, ze kterých se výsledné bludiště skládá. Stavebními prvky jsou krychle, které obsahují dutiny ve tvaru velkého písmene T. Toto bludiště představuje hlavolam, jehož cílem je dopravit míček přes spletitou cestu z počátečního do cílového bodu, pomocí otáčení celé krychle. Pro vytváření takového bludiště je potom využit genetický algoritmus (podrobněji viz [1]).

Využití generování bludišť ve steganografii

Kromě využití generování bludišť v počítačových hrách, lze generování využít také např. v kryptografii, konkrétněji ve steganografii. Steganografie je metoda, která ukrývá tajnou zprávu v nějakém nosiči. Zpráva je ukryta tak, aby pokud možno vůbec nebylo odhaleno, že dochází k nějaké výměně zpráv. Nosičem zpráv může být např. obrázek, zvuk nebo třeba více tradiční neviditelný inkoust. Jiným nosičem může být perfektní bludiště, ve kterém je zpráva zakódována [7].

Hlavní myšlenkou zakódování informace v bludišti je výběr určitých buněk na cestě z počáteční do koncové buňky a do těchto buněk informaci vložit. Tyto buňky, vhodné pro uložení informace, se nazývají vložitelné (embeddable). Buňka a na vybrané cestě je vložitelná, pokud má ve svém 4-okolí právě 2 buňky, které nepatří do vybrané cesty (O-buňky) a navíc tyto buňky z okolí nejsou sousedními buňkami nějaké vložitelné buňky, která se vyskytuje před a. O-buňky každé vložitelné buňky jsou podle určitého pevného klíče označeny jako „1“ nebo „0“, viz obrázek 6 (např. pokud má vložitelná buňka O-sousedy vlevo a vpravo, levá O-buňka je označena jako „1“, pravá jako „0“, apod.). Pokud se má v určité vložitelné buňce uložit bit „1“, je probourána zeď do O-buňky, která je označená jako „1“, pokud se má uložit „0“, je probourána zeď do O-buňky označené jako „0“.

u	0	0	1	
	×	×	×	0
	1	1		
		1	×	0
				v

Obrázek 6: Vybraná cesta z buňky u do buňky v, s vyznačením vložitelných buněk (označeno symbolem $\times$) a O-buněk označených symboly „1“, „0“.

Celý algoritmus potom funguje tak, že nejprve se algoritmem Hunt and Kill vygeneruje perfektní bludiště, zvolí se několik počátečních a jedna koncová buňka a naleznou se cesty v bludišti z všech počátečních do koncové buňky. Ostatní cesty se z bludiště vymažou (zůstane tedy bludiště obsahující pouze nalezené cesty). Jednotlivé cesty se seřadí podle počáteční buňky a pro každou cestu se naleznou vložitelné buňky. Pro každou cestu, vybíranou v sestupném pořadí, se do vložitelných buněk této cesty zakóduje zpráva (probouráním do příslušné O-buňky, podle toho, jaký bit je uložen). V posledním kroku se vygeneruje zbytek bludiště tak, aby bylo perfektní [7].

Pro zpětné získání zakódované zprávy musí příjemce znát polohu počátečních a koncové buňky cest, na kterých je zpráva zakódována. Bludiště je perfektní, mezi dvěma buňkami vede tedy právě jedna cesta. Poté může příjemce určit vložitelné buňky na těchto cestách a podle toho, kam je probouraná zeď z vložitelné buňky určit i informaci zakódovanou v této buňce. Průměrná kapacita pro uložení informace závisí na velikosti bludiště a počtu vybraných cest. Např. pro bludiště o rozměru 64×64 a čtyři cesty pro uložení zprávy, je průměrná kapacita pro uložení informace 216 bitů (podrobnější informace viz [7]).

3 Grafické generování bludiště

Při generování bludišť se často může stát, že na podobu výsledného bludiště jsou kladeny jisté omezující podmínky, popřípadě se má generování provést na základě zadané šablony. V tom případě je možné pro generování bludišť využít grafické generování (picturesque generation). Grafické generování umožňuje generovat bludiště podle černobílého rastrového obrázku tak, že cesta, která představuje řešení bludiště, vede přes všechny černé pixely [9].

Vstupem algoritmu pro grafické generování bludiště je tedy černobílý rastrový obrázek. Navíc se předpokládá, že černé pixely tvoří souvislou oblast. Množina černých pixelů je nazývána jako popředí, množina bílých pixelů jako pozadí. Pro reprezentaci bludiště se využívá grafová reprezentace popsaná v předchozí kapitole. Každý pixel vstupního obrázku tedy představuje vrchol grafu a všechny dvojice vrcholů, představující sousední pixely, jsou spojeny hranou.

V případě, že by pro generování bludiště byl využit některý z algoritmů popsaných v předchozí kapitole, nebylo by zaručeno, že řešení bludiště vede přes všechny vrcholy odpovídající černým pixelům. Tento problém lze také formulovat tak, že pokud si nazveme podgraf indukovaný uzly odpovídající černým pixelům jako přední podgraf (foreground subgraph), tak cesta představující řešení bludiště tvoří v předním podgrafu Hamiltonovskou cestu.

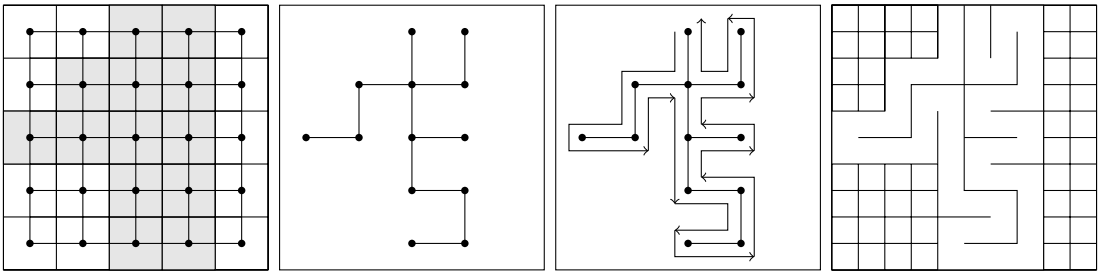
Nicméně může se stát, že vstupní obrázek takovou Hamiltonovskou cestu nemá. Navíc nalezení Hamiltonovské cesty v zadaném grafu je NP-úplný problém. Proto se provádí modifikace rozlišení vstupního obrázku tak, že modifikovaný obrázek již Hamiltonovskou cestu obsahuje. Celý postup generování je detailněji popsán v následujících podkapitolách (podrobnější informace viz [9]).

3.1 Generování popředí

Na vstupu je opět uvažován černobílý obrázek o rozměru $m \times n$. K tomuto obrázku vytvoříme příslušný neorientovaný graf podle popisu v předchozí části. V dalším kroku náhodně vybereme kostru předního podgrafu. K tomu lze využít randomizované algoritmy popsané v předchozí kapitole. Následně se celá nalezená kostra projde rovinným způsobem. Cesta, která vznikla, tvoří Hamiltonovskou cestu, která odpovídá řešení výsledného bludiště. Nicméně vytvořená cesta není Hamiltonovskou cestou v grafu, který odpovídá původnímu obrázku o rozměru $m \times n$. Vytvořená cesta je Hamiltonovskou v grafu, který odpovídá obrázku, který z původního vznikl rozdělením každého pixelu na 4 pixely. Modifikovaný obrázek má tedy rozměr $2m \times 2n$. Počáteční a koncová buňka bludiště odpovídají počátečnímu a koncovému uzlu Hamiltonovské cesty [9]. Proces generování popředí je znázorněn na obrázku 7.

3.2 Generování pozadí

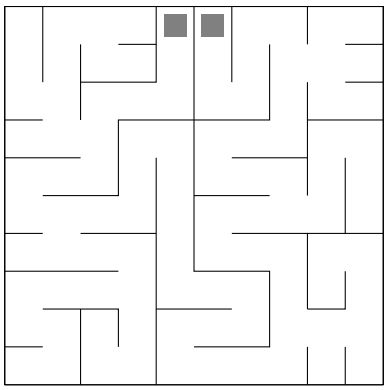
V případě, že již bylo provedeno generování popředí v grafu reprezentující modifikovaný obrázek o rozlišení $2m \times 2n$, je nutné provést generování zbytku bludiště. Za tím účelem uvažujeme podgraf vzniklý odstraněním hran z předního podgrafu, které



Obrázek 7: Proces generování popředí. Na prvním obrázku je vstupní rastrový obrázek s jeho grafovou reprezentací. Na druhém obrázku je kostra předního podgrafu. Na třetím je znázorněn rovinný průchod kostry, který odpovídá hledané Hamiltonovské cestě. Poslední obrázek znázorňuje řešení bludiště v modifikovaném obrázku.

nejsou součástí Hamiltonovské cesty. V dalším kroku vybereme libovolnou kostru tohoto podgrafu (lze využít algoritmy popsané v předchozí kapitole). Tato kostra již potom reprezentuje výsledné bludiště. Příklad výsledného bludiště je na obrázku 8. Celková složitost celého algoritmu záleží na použitém algoritmu pro hledání kostry grafu, což lze provést v polynomiálním čase. Tedy i celý algoritmus, při použití efektivních algoritmů pro hledání kostry grafu, má polynomiální časovou složitost.

V určitých případech generování rozsáhlých bludišť je možné pouhým okem rozlišit popředí od pozadí bez vyřešení bludiště, proto se používá jistá heuristika [9]. Tato heuristika spočívá v propojení dvou sousedních slepých konců bludiště, protože právě tyto slepé konce opticky rozdělují popředí a pozadí.



Obrázek 8: Příklad výsledné podoby vygenerovaného bludiště s vyznačenými počátečními a koncovými buňkami.

4 Generování řídkých bludišť využitím simulovaného žíhání

V této části práce se budeme zabývat vlastní modifikací předchozích algoritmů za účelem generování potenciálně řídkých čtvercových bludišť. Tedy algoritmus s určitou pravděpodobností generuje řídké bludiště a s určitou pravděpodobností perfektní bludiště. Problém lze také formulovat následovně: Ke dvěma zadaným buňkám (počáteční, koncová) na opačných stranách bludiště vygenerujeme potenciálně řídké bludiště, ve kterém existuje cesta mezi zadanými buňkami. Navíc do každé dostatečně velké oddělené části řídkého bludiště umístíme falešný cíl (tj. neexistuje do něj cesta z počáteční buňky). Taková bludiště s falešnými cíli lze po určitých modifikacích využít jako jednoduché hry, kdy více hráčů soupeří, kdo mezi falešnými cílovými buňkami rychleji nalezne pravou cílovou buňku, do které vede cesta z počáteční buňky. Celý tento algoritmus, včetně vizualizace vygenerovaného bludiště byl implementován v jazyce JavaScript.

Pro reprezentaci bludiště budeme podobně jako v předchozích kapitolách využívat grafovou reprezentaci. Vzhledem k tomu, že generujeme potenciálně řídké bludiště, graf bludiště nemusí být souvislý (může tedy obsahovat několik komponent). Nicméně musí být zaručeno, že uzly reprezentující počáteční a cílovou buňku (dále jen počáteční a cílový uzel), jsou ve stejné komponentě souvislosti. V každé komponentě se potom samostatně vygeneruje část celkového bludiště. Nejprve se budeme věnovat rozdělení grafu reprezentující bludiště do komponent souvislosti.

4.1 Rozdělení grafu do komponent souvislosti

Nejprve budeme vytvářet souvislou komponentu grafu, která obsahuje počáteční a cílový uzel. Navíc požadujeme, aby se souvislá komponenta vytvářela náhodně. Pro tento účel lze využít algoritmus DFS[2] z počátečního uzlu s náhodným výběrem sousedního uzlu a ukončením běhu algoritmu po nalezení koncového uzlu. Uzly na cestě z počátečního do koncového uzlu potom tvoří uzly hledané komponenty. Vzhledem k tomu, že poč. a koncový uzel jsou na opačných stranách bludiště, komponenta, která tyto uzly obsahuje, rozdělila graf do několika komponent.

V případě hledání komponenty u velkých bludišť se ale popsaný přístup ukázal jako značně neefektivní. Proto zavedeme následující heuristiku. Dále předpokládejme, že počáteční buňka je na vrchní straně a cílová na spodní straně bludiště. Vzhledem k tomu, že generujeme čtvercová bludiště, můžeme vygenerované bludiště na konci otočit tak, aby počáteční buňka byla na levé resp. pravé straně bludiště. Potom algoritmus DFS omezíme tak, že mezi sousední uzly nezahrnujeme uzly buněk, které jsou nad aktuální buňkou. Takto modifikovaný algoritmus DFS tedy může z každé buňky provést krok pouze do levé, pravé nebo dolní buňky (ne však do horní buňky). Navíc tímto modifikovaným algoritmem jsme schopni vytvořit stejné komponenty souvislosti jako nemodifikovaným algoritmem. Z důvodu efektivnosti je také využívána verze nevyužívající rekurzi.

[2]Přesněji řečeno se jedná o backtracking.

4.2 Rozšíření komponenty souvislosti

S využitím postupu popsaného v předchozím odstavci jsme schopni vytvořit komponentu souvislosti C obsahující počáteční a koncový uzel. Nicméně může se stát, že popsaný algoritmus vytvoří komponentu, jejíž uzly jsou právě uzly z nějaké nejkratší cesty z počátečního do koncového uzlu (viz obr 9 (a)). Potom by řešení bludiště vygenerovaného z těchto komponent bylo triviální. Proto komponentu C před samotným vygenerováním bludiště rozšíříme. Empiricky bylo zjištěno, že počet uzlů komponenty C by měl být alespoň tři čtvrtiny celkového počtu uzlů grafu (tj. $|C| \geq 3/4n^2$, kde n je rozměr bludiště).

Pro rozšiřování komponenty C použijeme myšlenku založenou na technice simulovaného žíhání (viz [4]). Rozšiřování budeme provádět po jednotlivých řádcích. To znamená, že komponentu C budeme rozšiřovat směrem vlevo a vpravo. Rozšiřování komponenty C pracuje tak, že pro každý uzel u z komponenty C se zjišťuje, jestli je u v této komponentě krajním uzlem, tj. uzel u má levého resp. pravého souseda, který není v komponentě C. Podobně rozlišujeme levý a pravý krajní uzel. V případě, že u je levým resp. pravým krajním uzlem, rozšíříme komponentu C o pevný počet uzlů reprezentující buňky směrem vlevo resp. vpravo. Tento pevný počet je vypočítán z rozměru bludiště. Po tomto rozšíření komponenty C o pevný počet uzlů provedeme další, již náhodné rozšíření o určitý počet uzlů. Způsob rozšiřování je znázorněn na obr. 9 (b). Další uzel se přidá s pravděpodobností

$$p = e^{-\frac{sum}{T}},$$

kde sum je dosavadní přidaný počet uzlů, a T je konstanta spočítaná z rozměru bludiště. Rozšiřování komponenty C v pseudokódu je v algoritmu 4.

Algoritmus 4: Rozšíření komponenty C

Vstup: Komponenta C získaná modifikovaným DFS algoritmem
Výstup: Rozšířená komponenta C'

```
1:  Nastavení hodnoty max, C' ← ∅
2:  foreach u ∈ C do
3:      sum ← 0
4:      if u je levou (pravou) krajní buňkou komponenty C then
5:          while u není krajní buňka bludiště do
6:              if sum < max then
7:                  C' ← C' ∪ {u}, sum ← sum + 1
8:              else
9:                  Vygeneruj náhodné číslo r z intervalu [0, 1] (rovnoměrně)
10:                 if r < e^(-sum/T) then
11:                     C' ← C' ∪ {u}, sum ← sum + 1
12:                 else
13:                     Break
14:             u ← levý (pravý) soused buňky u
15:         end
16: end
17: return C'
```

Poznámky k algoritmu. Z důvodu rozsahu je v pseudokódu uvedeno pouze rozšiřování na levou stranu. Po rozšíření na levou stranu se provede rozšíření i na pravou stranu (na řádku 15. by se znovu provedly řádky 4 – 15, ale s tím, že by se komponenta C rozšiřovala na pravou stranu – kód v závorkách).

Toto náhodné rozšíření je prováděno kvůli vizuální různorodosti výsledného bludiště a tomu, že díky provedené modifikaci DFS je pravděpodobnost výběru krátké cesty vyšší, než pravděpodobnost výběru klikaté cesty. A rozšíření komponenty C, která vznikla z přímé cesty o pevný počet buněk vlevo a vpravo, vede ke struktuře bludiště, ve kterém je možné jednoduše vizuálně rozlišit komponenty od sebe.

4.3 Generování výsledného bludiště

Vytvořením rozšířené komponenty C' a odstraněním hran, které vedou z této komponenty, nám graf reprezentující bludiště rozdělilo na několik komponent. Na každou tuto komponentu použijeme randomizovaný Primův algoritmus pro generování bludiště. Vznikne nám tedy bludiště, které obsahuje může obsahovat několik oddělených částí. Nicméně může se i stát, že komponenta C' obsahuje všechny uzly grafu (všechny buňky bludiště). V tom případě nám vznikne perfektní bludiště. Nakonec v každé dostatečně velké (vzhledem k počtu uzlů) komponentě, kromě komponenty C', určíme jeden uzel reprezentující buňku na kraji bludiště a určíme jej jako falešný cíl. Výsledný algoritmus zapsaný v pseudokódu je v algoritmu 5.

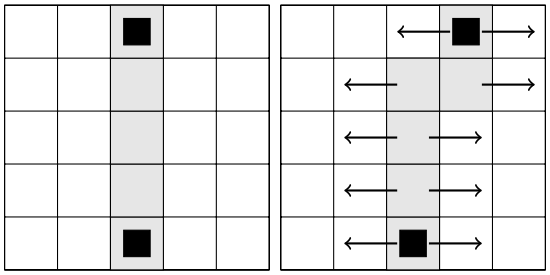
Algoritmus 5: Výsledné generování potenciálně řídkého bludiště

Vstup: Graf $G = (V, E)$ reprezentující bludiště bez vnitřních zdí, počáteční uzel $s \in V$, cílový (koncový) uzel $g \in V$

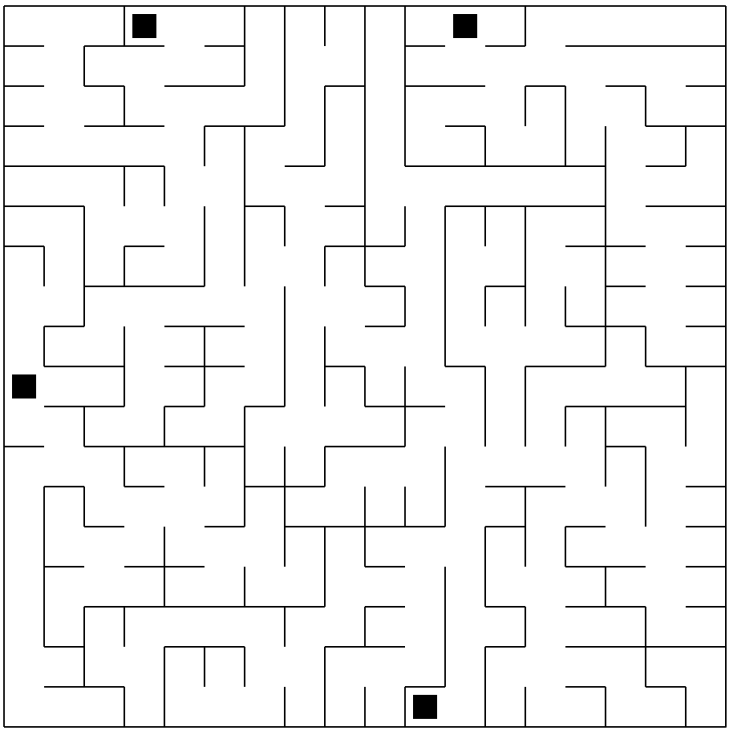
Výstup: Graf potenciálně řídkého bludiště G', množina falešných cílů F.

1: Volba konstanty min, $F \leftarrow \emptyset$, $E' \leftarrow \emptyset$.
2: Pomocí modifikovaného DFS vytvoř komponentu C obsahující uzly s a g.
3: Pomocí algoritmu 4 rozšiř komponentu C na komponentu C'.
4: Uvažuj komponenty souvislosti podgrafu indukovaného uzly $V \setminus C'$ jako množinu $\{C_1, C_2, \ldots, C_n\}$.
5: **foreach** $c \in \{C_0 = C', C_1, C_2, \ldots, C_n\}$ **do**
6: Nechť G_c je podgraf G indukovaný množinou uzlů c.
7: Vygeneruj bludiště randomizovaným Primovým algoritmem v grafu G_c a výsledek ulož do $G_r = (V_r, E_r)$.
8: $E' \leftarrow E' \cup E_r$.
9: **if** $|c| > min$ **then**
10: Náhodně vyber falešný cíl a umísti ho do množiny F.
11: **end**
12: **return** $G' = (V, E')$, F

Příklad většího vygenerovaného bludiště je na obrázku 10. Demonstrační aplikaci napsanou v jazyce JavaScript je možné vyzkoušet na adrese:
`http://www.stud.fit.vutbr.cz/~xhavle03/GAL/projekt.html`.



Obrázek 9: (a) Příklad komponenty C obsahující právě buňky reprezentující uzly nejkratší cesty z poč. do cílového uzlu. (b) Způsob rozšiřování komponenty C.



Obrázek 10: Příklad bludiště vygenerovaného popsaným algoritmem s falešnými cíli.

5 Závěr

Tato práce se zabývala algoritmy pro generování bludišť a jejich případnými aplikacemi. Byla objasněna souvislost mezi generováním perfektních bludišť a kostrami grafu v teorii grafů. V práci byly popsány základní algoritmy pro generování bludišť, vycházející z grafových algoritmů pro hledání koster grafu. Byl popsán způsob využití generování bludišť v počítačových hrách, ve steganografii a možnosti generování třírozměrných bludišť. Podrobněji byla také popsána metoda grafického generování, kdy se využívá černobílého rastrového obrázku jako jisté šablony pro výsledné bludiště.

V práci byla také popsána vlastní modifikace, která se zabývala generováním potenciálně řídkých bludišť s falešnými cíli. Tato modifikace je založena na rozdělení grafu do komponent souvislosti s využitím myšlenky simulovaného žíhání a následné generování celého bludiště. Celý algoritmus byl implementován v jazyce JavaScript.

Reference

[1] Akashi, R. a Fujiimoto, Y. *Generation of a Large Variety of 3-Dimensional Maze Problems with a Genetic Algorithm.* Sv. 4247. [b.m.]: Springer Berlin Heidelberg, 2006. S. 806–813. ISBN 978-3-540-47331-2.

[2] Aldous, D. J. A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics.* 1990, roč. 3. S. 450–465. ISSN 0895-4801.

[3] Foltin, M. *Automated Maze Generation and Human Interaction.* Brno: Masarykova Univerzita, Fakulta Informatiky, 2011. Diplomová práce.

[4] Hromkovič, J. *Theoretical Computer Science: Introduction to Automata, Computability, Complexity, Algorithmics, Randomization, Communication, and Cryptography.* [b.m.]: Springer Berlin Heidelberg, 2003. Texts in Theoretical Computer Science. An EATCS Series. ISBN 9783540140153.

[5] Járai, A. A. The Uniform Spanning Tree and related models. Prosinec 2009. Dostupné na: <`http://www.maths.bath.ac.uk/~aj276/teaching/USF/USFnotes.pdf`>.

[6] Křivka, Z. a Masopust, T. *Grafové algoritmy – slidy k přednáškám předmětu GAL.* 2015. Dostupné na: <`http://www.fit.vutbr.cz/study/courses/GAL/public/gal-slides.pdf`>.

[7] Lee, H.-L., Lee, C.-F. a Chen, L.-H. A perfect maze based steganographic method. *Journal of Systems and Software.* 2010, roč. 83, č. 12. S. 2528 – 2535. ISSN 0164-1212.

[8] Matějka, P. *Algoritmy pro generování a řešení bludišť.* Brno: Masarykova Univerzita, Fakulta Informatiky, 2012. Diplomová práce.

[9] Okamoto, Y. a Uehara, R. How to make a picturesque maze. In *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG2009).* 2009. S. 137–140. Dostupné na: <`http://cccg.ca/proceedings/2009/cccg09_36.pdf`>.

[10] Pech, A., Hingston, P., Masek, M. et al. *Evolving Cellular Automata for Maze Generation.* Sv. 8955. [b.m.]: Springer International Publishing, 2015. S. 112–124. ISBN 978-3-319-14802-1.

[11] Pedersen, H. a Singh, K. Organic Labyrinths and Mazes. In *Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering.* 2006. S. 79–86. ISBN 1-59593-357-3.

[12] Pullen, W. D. *Think Labyrinth: Maze Algorithms.* listopad 2015. Dostupné na: <`http://www.astrolog.org/labyrnth/algrithm.htm`>.

[13] Wikipedia. *Labyrinth.* 2015. Navštíveno 4. 11. 2015. Dostupné na: <`http://en.wikipedia.org/wiki/Labyrinth`>.