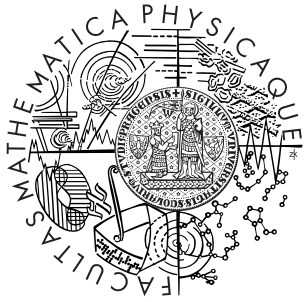
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Michal Švagerka

Adaptívne formácie pre viruálnych agentov

Katedra software a výuky informatiky

Vedoucí diplomové práce: Mgr. Tomáš Plch

Studijní program: Informatika
Studijní obor: Teoretická informatika

Praha 2011

Děkuji svému vedoucímu Tomáši Plchovi za zajímavé téma práce a za rozdílný přístup k problematice, který byl velkým přínosem. Dále děkuji Mateji Vitáskovi, Petru Michalíkovi a Janu Olšinovi za rozsáhlé korektury. V neposlední řadě děkuji Lydce Godušové za psychickou podporu a motivaci při studiu a psaní této práce.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

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 Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 15.4.2011

Název práce: Adaptívne formácie pre viruálnych agentov
Autor: Bc. Michal Švagerka
Katedra (ústav): Katedra software a výuky informatiky
Vedoucí diplomové práce: Mgr. Tomáš Plch

Abstrakt: Dnešní virtuální světy obsahují velké množství virtuálních agentů. To s sebou přináší i nutnost koordinace jejich pohybu. V této práci navrhujeme algoritmy pro řízení pohybu skupiny agentů ve formaci. Důraz je kladen na takové metody, které umožňují udržovat vlastnosti formace a zároveň překonávat rozličné terénní překážky. Tyto metody poté porovnáváme pomocí vhodně zvolené hodnotící funkce na vzorových scénářích.

Klíčová slova: formace, virtuální agent, vyhledávání cesty, terén, adaptace

Title: Adaptive formations for virtual agents
Author: Bc. Michal Švagerka
Department: Katedra software a výuky informatiky
Supervisor: Mgr. Tomáš Plch

Abstract: There is a growing number of virtual agents in today's virtual worlds. This is directly related to the need of coordinating their behavior and movement. In this work we design algorithms to maintain desired formation of agents while moving through difficult terrain. The methods we study should maintain the requirements of the formation as well as traverse along various obstacles. We then use an arbitrary fitness function to compare the performance of these methods on typical scenarios.

Keywords: formations, virtual agent, pathfinding, terrain, adaptation

Obsah

Úvod

Virtuální agenti hrají v počítačových hrách a v umělé inteligenci obecně velkou roli. Ve strategických hrách, kde má hráč možnost ovládat více agentů najednou, je třeba dbát na to, aby spolu tito na zadaných úkolech inteligentně spolupracovali. Jednoduchým příkladem takové činnosti je společný pohyb po virtuálním světě. Vzhled tohoto pohybu může, jakožto často opakující se činnost, výrazně ovlivnit hráčovy dojmy.

Formace měly význam již od raného vojenství. Již v 7. století před naším letopočtem se používaly první formace — falangy [10] — které sdružovaly hoplity do těsné formace, jež skýtala zejména obranné výhody. Postupem času se pro pěchotu vyvinuly další typy formací, například sloupec, řada a šíp. Formace kromě taktických výhod v obraně a útoku přináší i podporu morálky [7]. Možnou nevýhodou formací je jejich snížená pohyblivost [3]. U dnešních střeleckých týmů, které pokrývají větší plochu, je navíc pro člověka příliš náročné neustále vyhodnocovat pohyb ostatních jednotek ve formaci. Význam formací je tak dnes spíše teoretický.

V první kapitole blíže definujeme řešený problém. Provedeme několik architektonických rozhodnutí o tom, jak vypadá náš virtuální svět a jak se v něm agenti pohybují. Pokud by se koncepce světa zásadně odlišila, bylo by nutné některé další úvahy vhodně přizpůsobit.

Ve druhé kapitole prozkoumáme, jaké algoritmy pro řešený problém existují. Přitom rozebereme, proč jsou pro danou situaci nedostatečné či nevyhovující, a navrhneme, jakým způsobem by se mělo ubírat jejich řešení.

S podrobnější znalostí problému a úkolů, které budeme řešit, se ve třetí kapitole podíváme na způsob, kterým budeme kvalitu různých řešení porovnávat. Nejprve navrhneme ohodnocení, které popíše stav formace v daném časovém okamžiku, a poté se pokusíme analyzovat průběh této hodnotící funkce v čase, čímž budeme schopni ohodnotit stav formace v delším časovém úseku.

Čtvrtá kapitola dekomponuje algoritmus na jednotlivé moduly, které řeší konkrétní podproblémy. Navrhneme způsoby, kterým by se tyto daly vyřešit, a demonstrujeme úvahy, které stojí za navrhovanými řešeními.

V poslední, páté kapitole hodnotíme výkon jednotlivých modulů na několika testovacích scénářích a formacích. Tím budeme moci vyjádřit, které problémy se nám podařilo adresovat a do jaké míry bylo jejich řešení úspěšné.

Kapitola 1

Formalizace

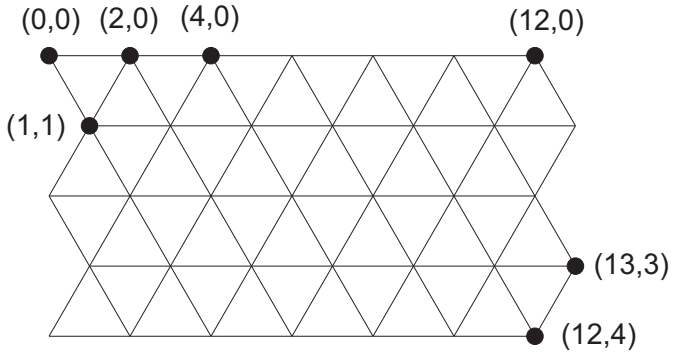
1.1 Mapa

Virtuální svět, ve kterém se budou agenti pohybovat, budeme nazývat **mapa**. Mapa je představována rovinným nakreslením grafu $G = (V, E)$. Grafy vždy vyplňují obdélníkové pole.

Máme k dispozici několik možností, jaký graf použít:

- čtvercovou mřížku, tedy graf, kde mají všechny vrcholy stupeň 4;
- trojúhelníkovou mřížku, tedy stupeň 6;
- čtvercovou mřížku s úhlopříčkami, tedy stupeň 8.

Stupeň vrcholu je v korespondenci s počtem směrů, kterými se může jednotka pohybovat. Čím více mají vrcholy stupňů, tím lépe se bude tento pohyb jevit. Osmisměrová mřížka však skýtá dva problémy: obsahuje hrany různé délky, a obsahuje hrany, které se kříží. Z těchto důvodů zvolíme trojúhelníkovou mřížku. Většinu dalších úvah v textu lze snadno přizpůsobit pro jiný typ mřížky.



Obrázek 1.1: Příklad mapy o rozměrech 7×5 s vyznačenými souřadnicemi některých bodů

1.1.1 Vrcholy

Vrcholy představují místa, kde se mohou vyskytovat jednotky. Na obrázku 1.1 jsou některé z vrcholů vyznačeny černými tečkami. Interní označení vrcholů používá normované souřadnice bodu v rovinném nakreslení. Vrchol v levém horním rohu má souřadnice $(0, 0)$. Jelikož jsou liché a sudé řádky navzájem posunuté, jsou použity pouze vrcholy, pro které je součet $řádek + sloupec$ dělitelný dvěma. Výhoda vynechávání sloupců spočívá v jednodušší implementaci hran a heuristické funkce. V textu označujeme vrcholy písmeny p, q, je-li vrcholů více, tak v_i.

Kromě logických souřadnic vrcholu je vhodné definovat si i souřadnice fyzické.

Fyzické souřadnice vrcholu ve vrcholu s logickými souřadnicemi (x, y) se spočítají předpisem

$$\left(\frac{1}{2} * size * x, \frac{\sqrt{3}}{2} * size * y\right) ,$$

kde $size$ je vzdálenost mezi dvěma sousedními vrcholy.

Je vidět, že zatímco logické souřadnice nabývají ve složkách pouze hodnot přirozených čísel, fyzické dosahují reálných hodnot.

Šířka mapy je určena počtem vrcholů na jednom řádku (x-ová souřadnice vrcholu na pravém okraji mapy je tedy až na konstantu dvakrát větší, než tato šířka). **Výška** mapy je určena počtem jejích řádků.

1.1.2 Hrany

Jednotky se na mapě pohybují pro **hranách** této triangulace. Každá hrana je orientovaná do jednoho ze šesti směrů, jak je uvedeno v tabulce 1.1[1]. Hodnoty Δx a Δy udávají změnu logických souřadnic vrcholu při přechodu přes hranu s danou orientací. Je vidět, že pro všechny hrany platí $|\Delta x| + |\Delta y| = 2$ a $|\Delta y| \leq 1$.

Hrany grafu nesou ohodnocení, které zjednodušeně znamená náročnost hrany. Čím je náročnější, tím déle trvá jednotce, než hranu projde. Hrany jsou orientované, avšak graf je symetrický. Tím je možné dosáhnout například efektu kopce (cesta do kopce je náročnější, než z kopce). Je také možné implementovat neprůchodné hrany (např. pokud taková hrana vede na místo, kde je neprostupná překážka), v takovém případě nemusí být graf symetrický (např. pokud je sklon kopce tak velký, že je z něho možné jít pouze dolů). Hranu z vrcholu p do vrcholu q označujeme $e_{p,q}$, její cenu pak $c_{p,q}$. Každá mapa má stanovenou minimální cenu hrany c_{min} a maximální c_{max}.

[1] Orientace hran je pouze slovní označení, které odpovídá azimutu pouze přibližně

Orientace	**Azimut**	Δx	Δy
Severovýchod	30°	+1	−1
Východ	90°	+2	0
Jihovýchod	150°	+1	+1
Jihozápad	210°	−1	+1
Západ	270°	−2	0
Severozápad	330°	−1	−1

Tabulka 1.1: Seznam orientací hran

Fyzické umístění pro bod na hraně vedoucí z vrcholu (x_1, y_1) do vrcholu (x_2, y_2) se spočítá předpisem

$$((1-t) * x'_1 + t * x'_2, (1-t) * y'_1 + t * y'_2) \ ,$$

kde (x'_1, y'_1) (resp. (x'_2, y'_2)) jsou fyzické souřadnice vrcholu (x_1, y_1) (resp. (x_2, y_2)) a parametr $t \in [0, 1]$ určuje přesný bod na hraně.

1.2 Jednotka

Jednotka je nejmenší entita pohybující se po mapě. Každá jednotka má své logické a fyzické umístění.

Logickým umístěním může být buď umístění ve vrcholu grafu, nebo na libovolné orientované hraně, kde je navíc doplněno o velikost části hrany ($t \in [0, 1]$), kterou již jednotka prošla.

Fyzické umístění představuje obecné souřadnice, které mají význam i bez znalosti grafu mapy. Tyto většinou odpovídají fyzickému umístění příslušného vrcholu nebo bodu na hraně, někdy se však mohou mírně odlišovat, například při detekci kolizí (viz 3.4).

1.3 Cesta

Cesta je konečná posloupnost vrcholů $P = (v_i)_{i=1}^{n}$ (pro $n > 1$), kde $\forall i \in \{1, 2, \ldots, n-1\}$ platí, že existuje hrana $e_{v_i, v_{i+1}}$. Bod v_1 označujeme jako **počátek** cesty, bod v_n jako **cíl** cesty.

Podposloupnost cesty P tvaru $F = (v_i)_{i=j}^{k}$, kde $1 \leq j < k \leq n$, nazveme **fragment** cesty. Dva fragmenty $F_1 = (v_i)_{i=j}^{k}$ a $F_2 = (v_i)_{i=l}^{m}$ na sebe **navazují**, pokud $k = l$. Fragment cesty je taktéž cesta.

Posloupnost hran $E^P = \left(e_{v_i, v_{i+1}}\right)_{i=1}^{n}$ nazýváme **posloupnost hran cesty**.

Cena cesty je definována jako $c(P) = \sum_{i=1}^{n-1} c_{v_i, v_{i+1}}$.

1.4 Schéma formace

Schéma formace je uspořádaná čtveřice $F = (G_F, a, \mathbf{o}, w)$.

$G_F = (S, B)$ je symetrický orientovaný **graf formace**. S je konečná množina **bodů** formace (*spots*[2]). B je konečná množina **vazeb** formace (*bindings*). Vazba formace je uspořádaná dvojice (p, q), kde $p, q \in S$.

Funkce $a : S \to S$ (*ancestor*) definuje **hierarchii** na formaci. Je-li $a(t) = s$, pak s je **bezprostředním velitelem** t a t je **závislý** na s. Hierarchie musí splňovat následující podmínky.

- $a(s) = t \Rightarrow (s, t) \in B$
- $\exists! \, s \in S : a(s) = s$. Tento bod označujeme jako **velitele** formace.
- Pro každý bod s existuje konečná posloupnost bodů $(s_i)_{i=1}^{n}$, kde $\forall i \in \{1, 2, \ldots, n-1\}$ je $a(s_i) = s_{i+1}$, $s_1 = s$ a s_n je velitel formace. Jinými slovy, konečným zřetězením funkce a lze získat velitele formace.

Funkce $\mathbf{o} : B \to \mathbb{R} \times \mathbb{R}$ je **optimální vzdálenost** (*offset*) na vazbě.

Funkce $w : B \to \mathbb{R}^+$ (*weight*) je **váha vazby**, která určuje, jak je vazba v kontextu formace použitá. Tato váha je použita např. při tvorbě kombinovaného prohledávacího prostoru (viz 3.1.1).

Tato definice nám umožňuje tvorbu různých typů schémat. Schémata, která budeme v práci používat, jsou popsána v příloze A.

Podle a a B dále definujeme množiny:

$$B^{\rightarrow} = \bigcup_{s \in S} B_s^{\rightarrow} \, ,$$

kde

$$B_s^{\rightarrow} = \{(s, t) \mid a(t) = s\} \, .$$

[2] Anglické výrazy pro uvedené pojmy jsou použity jako názvy tříd, metod a členských proměnných v programu.

1.5 Formace

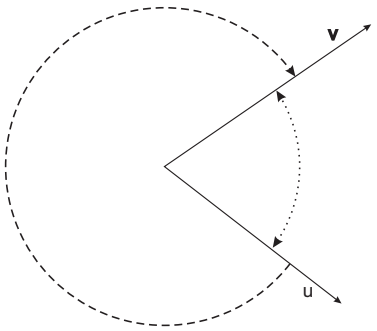
Formace představuje konkrétní instanci schématu formace. K bodům formace jsou vzájemně jednoznačně přiřazeny jednotky.

To nám umožňuje definovat funkci **skutečné vzdálenosti d** : $B \to \mathbb{R} \times \mathbb{R}$ (*distance*) na vazbě. Tato funkce vrací rozdíl umístění těch dvou jednotek, které jsou přiřazeny bodům vazby.

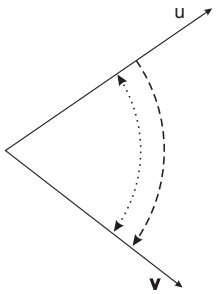
Dále se nám budou hodit dvě funkce, které popisují vztahy mezi směry vektorů:

- Funkce `angleTo`$(\mathbf{u}, \mathbf{v})$ vrací rozdíl směrů $\mathbf{u}$ a $\mathbf{v}$. Tento rozdíl nabývá hodnot z intervalu $[0°, 360°)$.
- Funkce `angleBetween`$(\mathbf{u}, \mathbf{v})$ vrací úhel, který spolu svírají směry $\mathbf{u}$ a $\mathbf{v}$. Tento úhel nabývá hodnot z intervalu $[0°, 180°)$.

U funkce `angleTo` tedy záleží na pořadí směrů, u `angleBetween` tomu tak není. Přehledně tyto hodnoty shrnuje obrázek 1.2 - tečkovaná čára znázorňuje v obou případech hodnotu `angleBetween`$(\mathbf{u}, \mathbf{v})$ a čárkovaná `angleTo`$(\mathbf{u}, \mathbf{v})$.



(a) Příklad rozdílných funkčních hodnot



(b) Příklad shodných funkčních hodnot

Obrázek 1.2: Výpočet hodnot `angleTo` a `angleBetween`

Triviálně platí:

- `angleBetween`$(\mathbf{u}, \mathbf{v}) = \min($`angleTo`$(\mathbf{u}, \mathbf{v}), 360° -$ `angleTo`$(\mathbf{u}, \mathbf{v}))$.
- `angleBetween`$(\mathbf{u}, \mathbf{v}) =$ `angleBetween`$(\mathbf{v}, \mathbf{u})$
- `angleTo`$(\mathbf{u}, \mathbf{v}) +$ `angleTo`$(\mathbf{v}, \mathbf{u}) = 360°$

1.6 Scénář

Scénář se skládá z mapy, schématu formace, jednotek, počátečního a cílového místa[3]. Úkolem formace bude nyní dostat velitele na toto cílové místo a během jeho cesty udržovat co nejlépe vztahy, kterými je popsána. Toho se bude dosahovat volbou cesty a vhodnou organizací ostatních jednotek. Míra splnění úkolu závisí na vlastnostech formace během přesunu a na čase, ve kterém je úkol splněn, přičemž čas splnění úkolu definujeme jako dobu od zadání rozkazu do ukončení pohybu všech jednotek na cílovém místě. Zjednodušeně řečeno, čím méně času bude přesun trvat a čím lépe bude formace udržovat zadané vztahy, tím je úkol lépe splněn.

Definovat způsob řešení zadaného scénáře je stěžejním bodem práce. Hodnocení tohoto řešení bude probíhat na několika scénářích, které jsou uvedeny v příloze B.

[3]Někdy může být výhodné mít na cestě definovány tzv. *waypointy*, které mohou přesněji určit, kudy se má formace pohybovat.

Kapitola 2

Rozbor zadání

2.1 Hodnocení

Základním problémem je formální definice úspěšnosti přesunu. Na něj lze klást dvě kritéria:

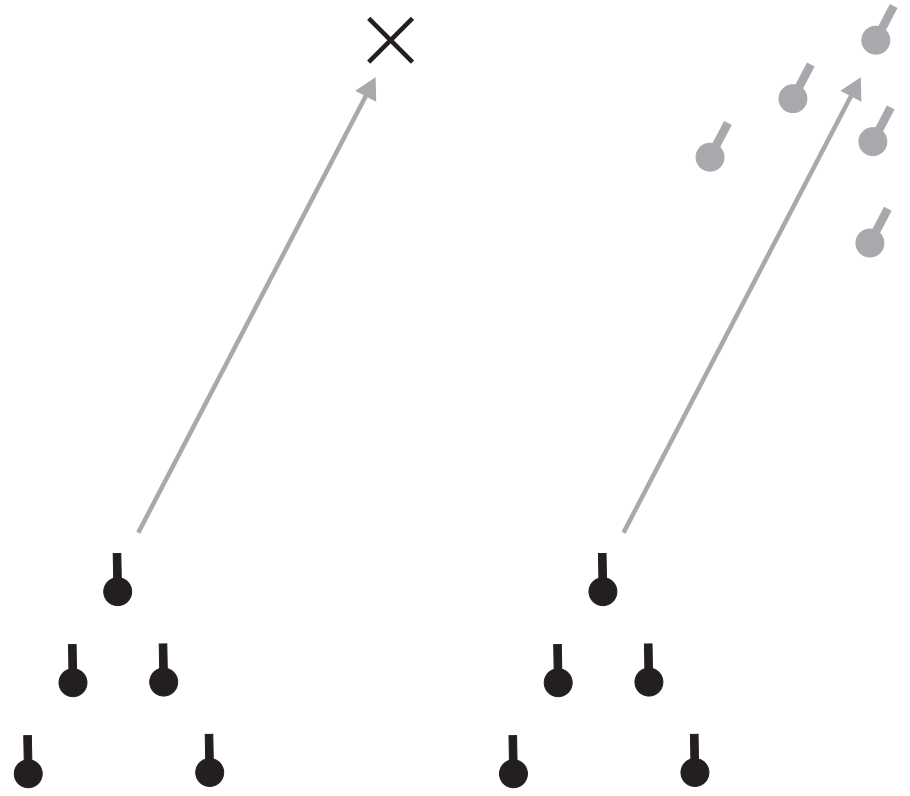
- Prvním kritériem je délka přesunu. Formace se může pohybovat nebezpečným terénem, kde může každý okamžik hrát potenciálně velkou roli pro přežití jednotek. Platí, že čím déle přesun trvá, tím je horší.
- Obtížněji uchopitelným kritériem je míra naplnění vztahů mezi jednotkami, které popisuje schéma formace. Nelze jednoduše říci, které vztahy ve formaci hrají důležitou roli, ani to, jak deformace ovlivňuje vzhled formace při pohledu shora.

Tato kritéria však jsou ve vzájemném rozporu. Chceme-li totiž formaci kopírovat velmi věrně, mohou mít některé jednotky delší nebo obtížnější cestu. Rychlejší postup pak nenechává tolik prostoru k udržování formace. Problematice hodnocení řešení se věnuje kapitola 4.

2.2 Triviální algoritmus

Budeme-li mít metodiku hodnocení, můžeme zkoumat vlastnosti a problémy jednotlivých algoritmů. Nejzákladnější metodou je triviální přesun. Lze jej popsat takto:

1. Vložíme schéma formace na mapu tak, aby budoucí pozice velitele odpovídala cíli přesunu, a formace byla orientovaná směrem od výchozího pole (viz 2.1b). Tím získáme cílové pozice ostatních jednotek ve formaci[1].
2. Pro každou jednotku naplánujeme cestu na její cílové místo.



(a) Formace dostane rozkaz na místo označené křížkem

(b) Jednotlivé jednotky dostanou rozkaz k přesunu na své cílové místo

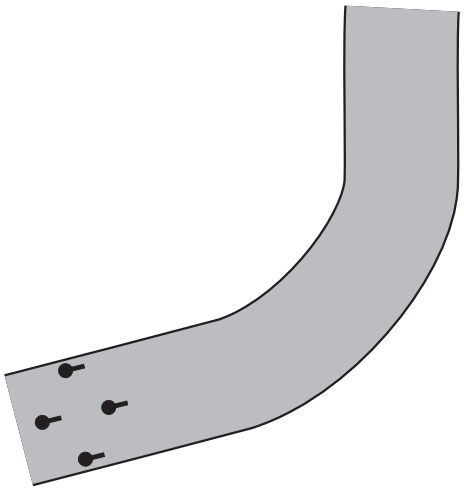
Obrázek 2.1: Schéma triviálního algoritmu pro přesun

Formace takto jistě splní zadaný úkol, přesune se na cílové místo a po ukončení přesunu udržuje vlastnosti formace nejlépe, jak je to možné. V následující podkapitole dojdeme jednoduchými úvahami k několika potenciálním důvodům, proč formace během cesty neudržuje tvar. Pro všechny situace budeme předpokládat, že naše formace má velitele vpředu a uprostřed (co se týče horizontálního směru). Příkladem této formace je například šíp (viz A.3). Pro jiné formace je třeba tyto úvahy přizpůsobit, nicméně většina pozorování zůstává v platnosti.

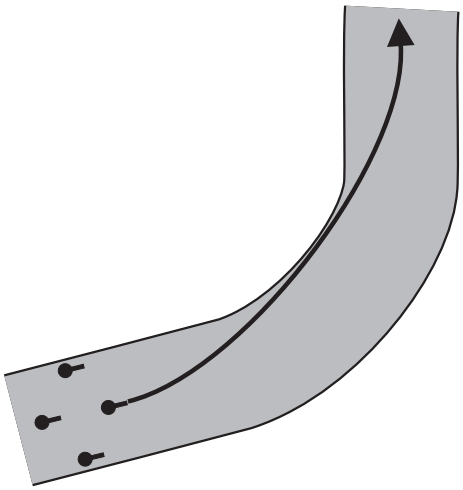
[1] Jelikož prohledávací prostor je poměrně hrubý, nemusí se na daném místě nacházet žádný vrchol. V takovém případě vybereme jednoduše ten nejbližší.

2.3 Problémy

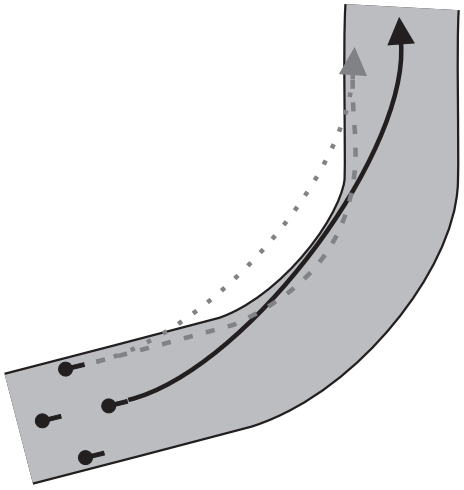
Hned první problém nastává u mapy tvaru jednoduché zatáčky, jako je na obrázku 2.2a. Šedivá barva představuje cestu, která je oproti okolnímu terénu výrazně levnější. Jednoduché prohledávání najde cestu podobnou té na obrázku 2.2b. Tato je vhodná pro jednotku samotnou, avšak selhává ve formaci. Jednotky, které mají své místo vlevo od velitele, totiž řeší rozpor, který vidíme na obrázku 2.2c. Buď budou dodržovat svoji předepsanou cestu a budou se pohybovat dražším terénem, čímž jim bude přesun trvat déle (tečkovaná cesta), nebo půjdou podobnou cestou, jako velitel, čímž sice ušetří čas, ale poruší vlastnosti formace (čárkovaná cesta). Tomu se lze vyhnout tím, že velitel prodlouží svoji cestu tak, aby ve vnitřní straně zatáčky byl dostatek místa pro podřízené jednotky. Ideální cesta by pak měla vést středem cesty, jak je zobrazeno na obrázku 2.2d.



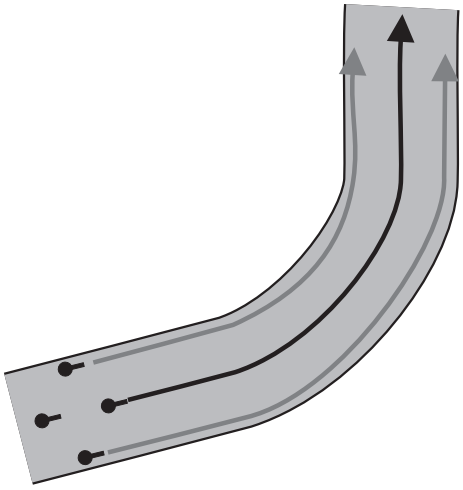
(a) Schéma zatáčky



(b) Cesta velitele u triviálního algoritmu



(c) Cesta jednotky vlevo od velitele



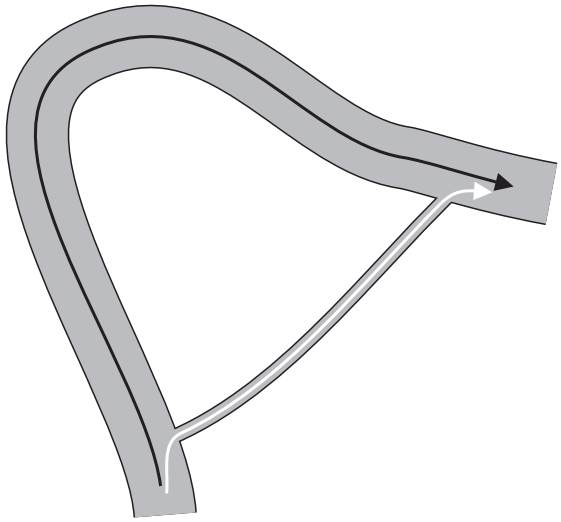
(d) Vhodnější cesta velitele i s cestou pro obě křídelní jednotky

Obrázek 2.2: Cesta formace zatáčkou a problémy s tím spojené

Cesta formace zatáčkou v sobě však ukrývá ještě jeden problém, který je dobře vidět na obrázku 2.2d. Jednotky ve vnitřní straně zatáčky mají kratší cestu než velitel, a jednotky na vnější straně naopak delší. Pokud by všechny jednotky šly stejnou rychlostí, ty na levém křídle by byly oproti veliteli napřed, ty na pravém křídle pro změnu pozadu. Pro jednotky vlevo i pro velitele by tedy bylo rozumné vhodně zpomalit, aby jednotky vpravo formaci stačily.

Další problém je spíše vzhledového původu. Zdá se, že i krátký časový okamžik, ve kterém je část formace již v cíli a nepohyblivá, kdežto některé jednotky se ještě přesunují, působí rušivě. Pokud by všechny jednotky dorazily do cílové pozice najednou, mohlo by to zlepšit dojem z pohybu.

V případě, že je cesta v některém bodě příliš úzká, a kolem je oblast velmi drahá nebo neprůchodná, formace má tendenci smrštit se pod únosnou mez. Například na obrázku 2.3 je šedou barvou vyznačena cesta a bílou neprostupný terén. Bílá šipka označuje cestu, která je pro formaci příliš úzká — formace by se musela příliš zmenšit, aby danou cestou prošla. Naproti tomu šipka označená černou šipkou označuje cestu, která je sice delší, avšak z hlediska dodržení vlastností formace vhodnější.



Obrázek 2.3: Problém příliš úzkého průchodu. Bílá cesta je pro formaci příliš úzká, je třeba použít delší černou.

Mají-li jednotky výběr z více podobně dlouhých cest, mohou nastat ještě jiné problémy. Jelikož jsou od sebe jednotky ve formaci mírně posunuté, může se jejich vyhodnocení toho, která cesta je výhodnější, lišit. Formace se pak rozdělí na dvě nebo více nezávislých částí, z nichž každá cestuje jiným směrem. Vyjma případů, kdy se jedná o dostatečně malou překážku, by bylo vhodné, aby všechny jednotky

dané formace použily cestu stejnou. Rozlišení velikosti překážek není jednoznačné, může záviset na velikosti a typu formace i ceně cesty.

Problémem, který je třeba v podobných algoritmech vždy adresovat, je detekce kolizí a jejich řešení. Triviální algoritmus umožňuje, aby dvě nebo více jednotek měly v cestě stejný vrchol. Je-li navíc tento v podobném úseku cesty, může nastat kolize. Dvě jednotky budou příliš blízko u sebe, čehož si uživatel jistě všimne.

Poslední věc je fakt, že kvůli poměrně hrubému rastru mapy vypadá pohyb jednotek velmi omezený. Zjemnění rastru s sebou může nést výkonové problémy, takže bude třeba problém vyřešit sofistikovanějším způsobem.

Jak je vidět, problémů s triviálním algoritmem je mnoho. Jeden z možných způsobů, jak se pokusit je vyřešit, je navrhnout algoritmus zcela jiný. Druhá možnost, kterou použijeme, je zaměřit se na různé problémy a pokusit se je vyřešit každý zvlášť, případně několik podobných problémů najednou. Pokud by se podařilo pro každý aspekt algoritmu dedikovat vlastní modul, který by problém řešil. V takovém případě by bylo možné snadno přepínat jednotlivá chování. Kromě usnadnění vlastní tvorby a získání určité rozšiřitelnosti návrhu se tím otevírá možnost pro adaptaci na vnější podmínky nebo vnitřní stav agentů.

Kapitola 3

Algoritmus

3.1 Hledání cesty

Jádrem algoritmu provádějícím pohyb jednotek v prostoru bude hledání cesty. Potřebujeme nástroj, který nám pro zadaný výchozí a cílový bod v prostoru najde nejkratší cestu mezi nimi.

Prohledávacím prostorem je v této situaci naše mapa oblasti. Pro samotné hledání nejlevnější cesty pak použijeme algoritmus A*, který je popsán například v [1]. Algoritmus A* je úplný prohledávací algoritmus, který používá heuristický odhad ceny zbývající cesty pro ohodnocení prohledávaných vrcholů. Díky tomu dosahuje v očekávaném případě lepšího výkonu než prohledávání do šířky. Je-li heuristika přípustná, je algoritmus korektní (tedy nalezne nejkratší cestu). Jako heuristická funkce nám v tuto chvíli poslouží následující funkce.

Věta 3.1.1 $h(p,q) = c_{min} * \max\{\frac{|p.x-q.x|+|p.y-q.y|}{2}, |p.y-q.y|\}$ *(kde* c_{min} *je minimální ohodnocení hrany) je přípustná heuristika.*

Důkaz: Označme $dx = |p.x - q.x|$ a $dy = |p.y - q.y|$.

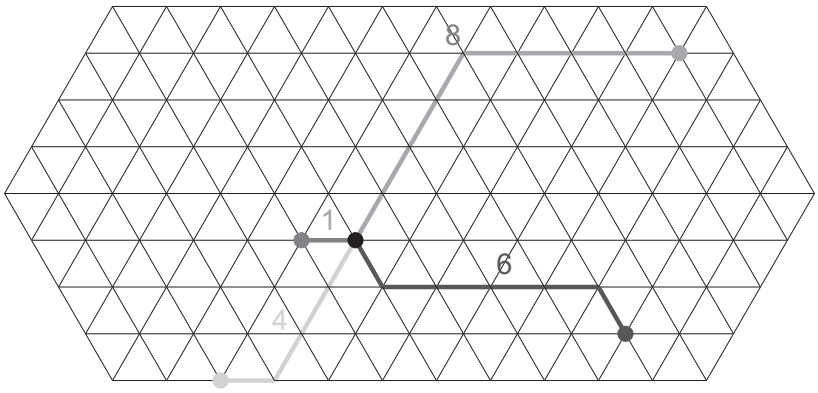
Pro cestu délky jedné hrany triviálně platí $\frac{dx+dy}{2} = 1$ a $dy \leq 1$ (viz kapitola 1.1.2).

Na délku cesty k hran můžeme pohlížet také jako na k cest délky jedné hrany, takže platí $\frac{dx+dy}{2} \leq k$ a $dy \leq k$[1], dohromady $\max\{\frac{dx+dy}{2}, dy\} \leq k$.

Z toho plyne, že $f(p,q) \geq h(p,q) = c_{min} * \max\{\frac{dx+dy}{2}, dy\}$, tedy h je přípustná heuristika.

Poznámka 3.1.2 *Tato heuristika odpovídá nejkratší cestě co do počtu hran vynásobené minimální možnou cenou hrany (viz obrázek 3.1).*

[1] Plyne z nerovnosti $|\sum x_i| \leq \sum |x_i|$.



Obrázek 3.1: Heuristická funkce pro $c_{min} = 1$

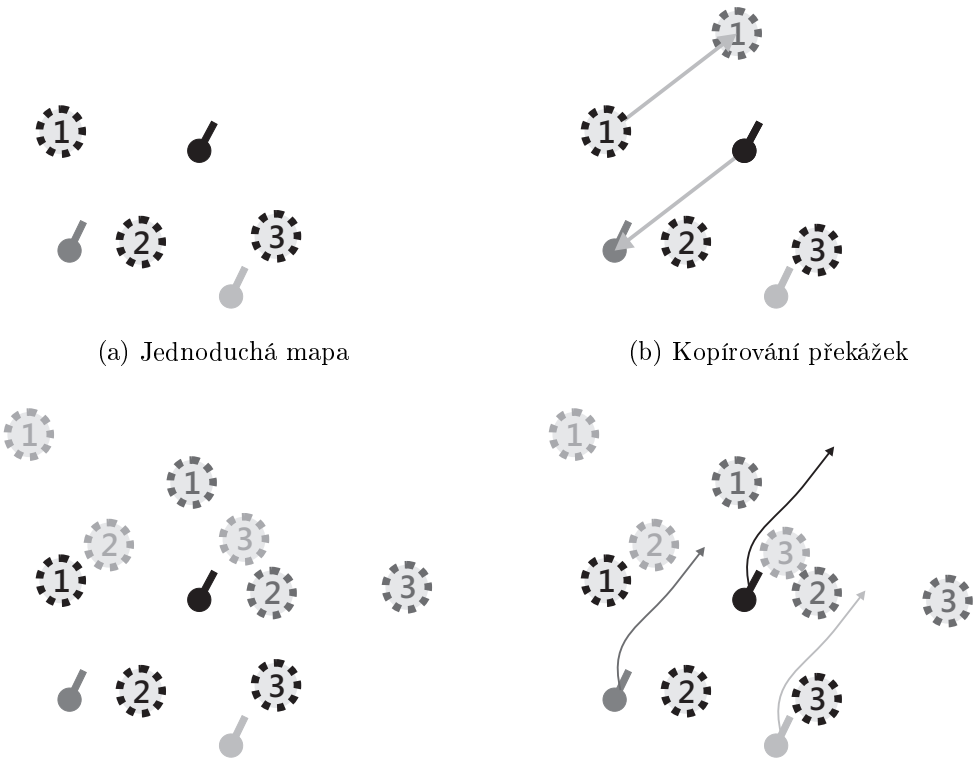
3.1.1 Kombinovaný prohledávací prostor

Triviální postup hledání cesty pro každou jednotku zvlášť do určité míry funguje, jakmile se ale formace dostane do užších míst, nastávají potíže. Je-li např. velitel horizontálně uprostřed formace, vyplatí se, aby si plánoval cestu spíše středem oblastí s nízkou cenou, aby jednotky na křídlech byly v této oblasti také. Pokud je tato oblast velmi úzká a kolem ní je oblast velmi drahá, nebo dokonce neprůchodná, je dokonce rozumné se této oblasti vyhnout, i když by to pro samotnou jednotku byla nejlevnější cesta. Jinak řečeno je vhodné, aby velitel při hledání cesty bral v úvahu i jednotky, které jsou v hierarchii pod ním.

Toho můžeme jednoduchým způsobem dosáhnout tak, že prohledávací mapy jednotlivých jednotek překryjeme posunuté podle toho, jak pozici ve formaci mají. Tuto metodu nejlépe demonstrujeme na příkladě. Na obrázku 3.2a vidíme schéma formace a terénu: kruhové očíslované objekty představují neprůchodné překážky (např. stromy) a různobarevné kruhy s vyznačením orientace jsou jednotky. Chceme zkonstruovat prohledávací mapu velitele (zobrazen černě). Nejprve určíme vektor posunutí závislé jednotky oproti veliteli a všechny překážky přesuneme o opačný vektor. Na obrázku 3.2b vidíme tento krok pro tmavě šedou jednotku a strom číslo 1, na obrázku 3.2c vidíme výslednou mapu. Tuto mapu pak velitel použije pro vyhledání cesty, kterou mohou kopírovat všechny závislé jednotky bez obav z neprůchodného terénu (viz 3.2d).

Tato myšlenka se dá snadno aplikovat na naši trojúhelníkovou reprezentaci s ohodnocením hran. Místo přesunu překážek budeme kombinovat cenu hran na odpovídajících místech.

$$c^c_{p,q} = c_{p,q} + \frac{\sum_{b \in B^{\rightarrow}_s} w(b) * c_{p+\mathbf{O}(b),q+\mathbf{O}(b)}}{\sum_{b \in B^{\rightarrow}_s} w(b)}$$



(a) Jednoduchá mapa

(b) Kopírování překážek

(c) Výsledná mapa

(d) Nalezená cesta

Obrázek 3.2: Konstrukce kombinovaného prohledávacího prostoru

Je-li hrana u závislé jednotky neprůchodná, není vhodné, aby byla odpovídající hrana neprůchodná i u velitele. Pokud bychom ve výše uvedeném příkladě přidali ještě několik závislých jednotek nebo překážek, mohlo by se stát, že by se vytvořily neprůchodné, nebo dokonce úplně nedostupné oblasti. Přitom by bylo určitě možné najít takovou cestu, kterou by většina závislých jednotek mohla projít, a jen některé by se musely z kurzu mírně odchýlit a deformovat tak uskupení. Toho lze snadno dosáhnout tím, že pro účely konstrukce kombinované mapy nahradíme neprůchodné hrany v mapách závislých jednotek hranami s cenou $2c_{max}$.

Poznámka 3.1.3 *Jelikož jsou mapy potenciálně velké a formace se může otáčet, deformovat, či dokonce úplně měnit, není vhodné konstruovat kombinovanou mapu celou, ale počítat pouze ty hrany, které jsou aktuálně třeba. Tento přístup je sice výkonově znát na samotném vyhledávání cesty, ale pro malé počty jednotek ve formaci není toto zpomalení výrazné.*

3.2 Plánování

Velkou roli v udržování formace může hrát způsob plánování cest. Triviální přístup je naplánovat všem jednotkám postup přímo do cílové pozice. Ukazuje se však, že při takovém postupu formace nedrží tvar příliš dlouho, a příchod do cílové pozice je velmi chaotický. Nejen že se mohou jednotky v závislosti na profilu terénu hýbat výrazně jinou rychlostí, některé z nich ale mohou mít i kratší cestu (např. na vnitřní straně zatáčky). Dokonce se může stát, že by se v případě velké překážky uprostřed cesty (např. budova, jezero) mohla formace rozdělit na dvě části, z nichž každá obchází překážku z jiné strany, čímž by se formace rozdělila na dvě nezávislé části.

3.2.1 Periodické přeplánování

Aby se vliv těchto problémů výrazně zmenšil, je možné s úspěchem používat tzv. periodické přeplánování. Základem této metody je rozdělit cestu na několik fragmentů a uvnitř každého z nich plánovat cestu zvlášť.

Velitel formace si nejprve naplánuje celou cestu do cílového bodu. Poté určí první fragment cesty, a všechny ostatní jednotky naplánuje do jeho cíle. Po projití určité části fragmentu určí velitel fragment nový, a postup zopakuje.

Účinnost této metody závisí na zvolených parametrech, a to:

- F — délka fragmentu v počtu hran
- P — počet prošlých hran, po kterých následuje výběr nového fragmentu

Parametr P je závislý na F. Zřejmě by mělo být $P \leq F$, jinak by se po projití fragmentu formace zastavila. Bez synchronizace příchodu (viz 3.3) by se některé jednotky mohly zastavovat i při $P = F$ nebo při P blízko F. Je-li naopak P příliš malé, nastává výběr nového fragmentu velmi často[2] a navíc se většina cesty ve fragmentu současném počítá zbytečně. Jako kompromis se zdá být vhodná hodnota $P \sim 0.5 \cdot F$.

Stačí tedy optimalizovat parametr F. Na rozdíl od druhého parametru je příliš malé F na škodu. Ubírá totiž formaci na volnosti při obcházení malých překážek (např. strom) a překonávání menších rozdílů v náročnosti cesty. Navíc, pokud formace po cestě zatáčí (což je téměř vždy), pak se jednotky, které jsou daleko za velitelem, budou muset pohybovat po oblouku na své nové místo, a to téměř kolmo na požadovaný směr pohybu. Vysoké F naopak znamená, že pro krátké cesty (a krátké úseky cest) nehraje přeplánování a fragmentaci žádnou pozitivní roli.

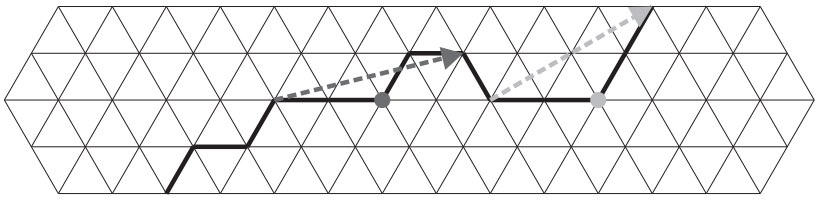
[2] To je problém výkonnostního charakteru. Není-li výkon tak důležitý, je možné P zmenšit třeba až na jednu hranu.

Vhodné hodnoty parametru F bude třeba určit analýzou chování tohoto algoritmu jeho pro různé hodnoty. Tato hodnota se pak může vhodně měnit podle daného schématu formace nebo podle požadavků na rychlost či kvalitu přesunu.

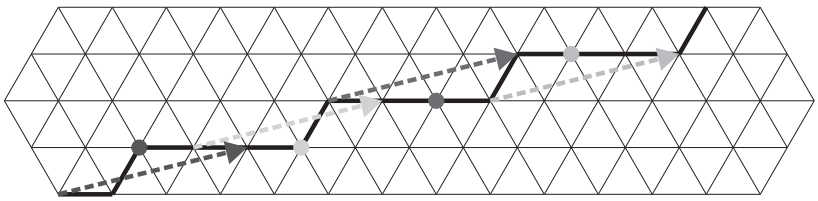
3.2.2 Fragmentace po zatáčkách

Problém přístupu periodického přeplánování je v tom, že při fragmentaci vůbec není brán v úvahu tvar cesty. Mezi přeplánováním před zatáčkou a po ní může být velký rozdíl — po projití zatáčky by měla být formace jinak orientovaná než před ním. Na dlouhých úsecích bez zatáčky je naopak zbytečné plánovat cestu po více úsecích. Pokud bychom tedy cestu rozdělili na přímé úseky bez zatáček, mohla by fragmentace tuto roli splňovat.

Aby bylo možné určit, co je přímý úsek cesty, musíme být nejprve schopni říci, jaký směr má cesta jednotky v daném vrcholu. Určení směru cesty má lokální charakter, je tedy závislé na několika okolních hranách. V praxi se ukázalo být rozumné vzít azimut mezi vrcholem dvě hrany po cestě a vrcholem dvě hrany proti cestě (viz obrázek 3.3). Velikost tohoto „okénka“ je dostatečně malá na to, aby se skutečná zatáčka projevila rychle, ale aby malá fluktuace v cestě zatáčku neznamenala. Na obrázku 3.4 vidíme cestu, která je z globálního hlediska rovná, a jen diskrétnost mapového prostoru způsobuje, že se v cestě vyskytují hrany s různou orientací. Ve všech bodech je ale podle zvolené definice směr cesty stejný.



Obrázek 3.3: Směr cesty v několika bodech zkonstruovaný metodou „okénka“



Obrázek 3.4: Robustnost definice směru cesty

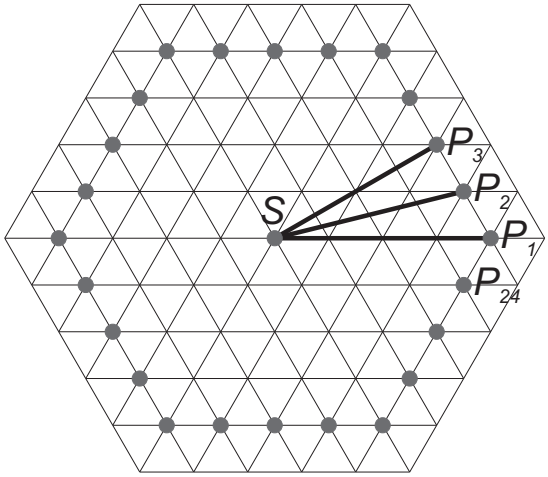
Mějme nyní dva různé úseky cesty F_1 a F_2. Největší úhel, který spolu svírají směry cesty ve vrcholech z úseku F_1, označme φ_1, analogicky φ_2 pro směry cesty

ve vrcholech F_2. Liší-li se úhly φ_1 a φ_2 pouze o nějakou malou hodnotu ϵ, je vhodné, aby platilo, že jsou oba úseky F_1 a F_2 přímé, nebo není přímý ani jeden z nich.

$$|\varphi_1 - \varphi_2| < \epsilon \Rightarrow (F_1 \text{ je přímý} \Leftrightarrow F_2 \text{ je přímý})$$

Aby byla definice dostatečně robustní, je třeba, aby se rozdíly směrů podobné velikosti chovaly z hlediska detekce přímých úseků stejně. Proto nyní zjistíme, jaké hodnoty můžou tyto rozdíly nabývat.

Označme S libovolný bod na mapě. Označme po řadě $P_1, P_2 \ldots P_{24}$ všechny body ve vzdálenosti 4 hrany (protože 4 je velikost okénka) od S a ještě definujme $P_{25} = P_1$ a $P_{26} = P_2$. Schéma je souměrné podle každé přímky $\leftrightarrow SP_i$, což nám umožňuje jednoduše určit úhly, které svírají jednotlivé směry.



Obrázek 3.5: Množina bodů ve vzdálenosti 4 hran od středového bodu S

Pro sousední směry zjistíme měřením

$$|\angle P_{2k} S P_{2k+1}| = 13.885^\circ \,,$$

$$|\angle P_{2k-1} S P_{2k}| = 16.115^\circ$$

pro $k \in 1, 2, \ldots, 12$.

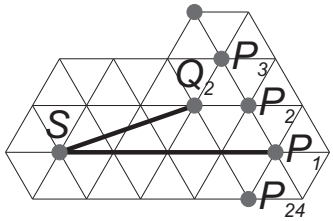
Pro dva směry ob jeden je situace jednodušší, platí

$$|\angle P_i S P_{i+2}| = 30^\circ$$

pro $i \in 1, 2, \ldots, 24$.

Zatím jsme ale počítali pouze s cestami, které se vždy vzdalují od počátečního bodu, což však obecně neplatí. Možných směrů však ani tak není výrazně více,

až na symetrie dokonce pouze jeden. Na obrázku 3.6 je tento směr označen jako $\leftrightarrow SQ_2$. Ostatní body bližší k S leží vždy na nějaké úsečce SP_i a definují tedy stejný směr.



Obrázek 3.6: Bod Q_2 definuje možný směr pohybu při velikosti okénka 4.

Měření nám dává

$$|\angle P_{2k-1}SQ_{2k}| = 19.106^\circ \ .$$

Nejmenší úhly, které směry svírají, jsou tedy (pro $k \in \{1, 2, \ldots, 12\}$):

$$|\angle P_{2k}SQ_{2k}| = 2.991^\circ \ ,$$

$$|\angle P_{2k+1}SQ_{2k}| = 11.894^\circ \ ,$$

$$|\angle P_{2k}SP_{2k+1}| = 13.885^\circ \ ,$$

$$|\angle P_{2k-1}SP_{2k}| = 16.115^\circ \ ,$$

$$|\angle P_{2k-1}SQ_{2k}| = 19.106^\circ \ ,$$

$$|\angle P_{2k+2}SQ_{2k}| = 27.009^\circ \ ,$$

$$|\angle P_{2k-1}SP_{2k+1}| = 30^\circ \ ,$$

$$|\angle P_{2k}SP_{2k+2}| = 30^\circ \ .$$

Ze symetrie směrové růžice plyne, že každý možný úhel mezi směry má velikost ve tvaru $k \cdot 30^\circ + \psi$, kde $k \in \{0, 1, \ldots, 11\}$ a ψ je jeden z výše uvedených naměřených úhlů.

Největší rozdíly jsou mezi dvojicemi úhlů $(k \cdot 30^\circ + 2.991^\circ, k \ \cdot 30^\circ + 11.894^\circ)$ a $(k \cdot 30^\circ + 19.106^\circ, k \ \cdot 30^\circ + 27.009^\circ)$, shodně 8.903°. Je tedy vhodné, aby práh rozlišující přímé úseky od těch, které přímé nejsou, ležel v jednom z těchto intervalů. Jelikož by tento práh neměl nabývat vysokých hodnot, nabízí se pouze několik možností: $\{10^\circ, 20^\circ, 40^\circ, 50^\circ\}$.

Definice 3.2.1 *Úsek cesty označíme jako φ_0-**přímý** právě tehdy, když pro každou dvojici jeho vrcholů platí, že směry cesty v těchto vrcholech svírají úhel menší, než práh φ_0, kde $\varphi_0 \in \{10^\circ, 20^\circ, 40^\circ, 50^\circ\}$.*

Poznámka 3.2.2 *Různá hodnota parametru φ_0 bude zřejmě mít vliv na průběh algoritmu. Tento vliv bude třeba analyzovat a vybrat hodnotu, která je pro daný algoritmus nejvhodnější, nebo umožnit jejich výběr podle daného schématu formace, či podle požadavků na rychlost či kvalitu přesunu.*

Tato definice nám umožňuje jednoduchý algoritmus pro detekci φ_0-přímých úseků (viz kód 3.1). v_i jsou po řadě vrcholy cesty, funkce `angleBetween` vrací úhel, který spolu svírají směry dvou vrcholů. Algoritmus vrací množinu vrcholů, která představuje první přímý úsek na cestě.

$min \leftarrow max \leftarrow v_1$
$i \leftarrow 1$
while `angleBetween`$(min, max) < \varphi_0$ **do**
  $i \leftarrow i + 1$
  if lastNode(v_i) **then**
    return $\{v_1, v_2, \ldots, v_i\}$
  end if
  $\alpha \leftarrow$ `angleBetween`(min, max)
  $\beta \leftarrow$ `angleBetween`(min, v_i)
  $\gamma \leftarrow$ `angleBetween`(max, v_i)
  if $\beta + \gamma > \alpha$ **then**
    if $\beta < \gamma$ **then**
      $min \leftarrow v_i$
    else
      $max \leftarrow v_i$
    end if
  end if
end while
return $\{v_1, v_2, \ldots, v_{i-1}\}$

Kód 3.1: Algoritmus pro detekci přímých úseků

Lemma 3.2.3 *Pro každé $n \in \mathbf{N}$ platí: Dvojice vrcholů z $v_1, v_2, \ldots, v_n$, jejichž směry spolu svírají největší úhel, je min a max.*

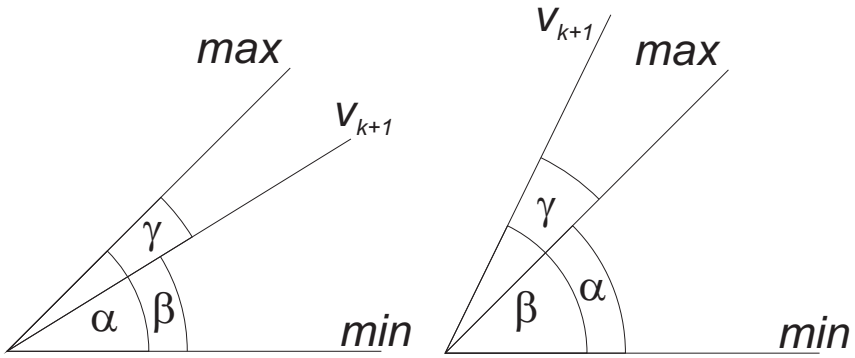
Důkaz: indukcí podle n.

$n = 1$: Máme pouze jediný vrchol v_1, a $min = max = v_1$

Předpokládejme nyní, že pro $n = k$ platí, že směry min a max spolu svírají maximální úhel. Označme po řadě α, β, γ úhly mezi dvojicemi směrů (min,max), (min,v_{k+1}), (max,v_{k+1}).

Je-li směr v_{k+1} mezi *min* a *max* (viz obrázek 3.7a), pak platí $\beta+\gamma = \alpha$, *min* a *max* se nezmění, a stále se jedná o vrcholy, jejichž směry spolu svírají maximální úhel.

Je-li naopak směr v_{k+1} mimo *min* a *max* (viz obrázek 3.7b), pak platí $\beta+\gamma > \alpha$. Je-li nyní $\beta > \gamma$, znamená to, že směr *max* je mezi v_{k+1} a *min*. Největší úhel tedy spolu nyní svírají směry *min* a v_{k+1}, což je v souladu s přiřazením $max = v_{k+1}$. Analogicky pro $\beta < \gamma$. Konečně pokud platí $\beta = \gamma$, musí nutně být $min = max$, a tedy přiřazení $max = v_i$ je v pořádku.



(a) v_{k+1} mezi *min* a *max* (b) v_{k+1} mimo *min* a *max*

Obrázek 3.7: Pozice v_{k+1}

Dokázali jsme tedy, že věta platí pro $n = 1$ a že pokud věta platí pro $n = k$, platí i pro $n = k + 1$, čímž je věta dokázána, pro všechna přirozená n.

Lemma 3.2.4 *Pokud cyklus skončí, vrátí nejdelší možný přímý úsek.*

Důkaz: Stačí dokázat:

Cyklus skončí $\Leftrightarrow$ Byla nalezena dvojice vrcholů, jejichž směry spolu svírají úhel větší než φ_0.

$\Rightarrow$: Plyne přímo z ukončovací podmínky cyklu a z faktu, že $\{min, max\} \subseteq \{v_1, v_2, \ldots, v_n\}$

$\Leftarrow$: z 3.2.3 víme, že `angleBetween`(min, max) největší úhel, který spolu svírají dva vrcholy z $\{v_1, v_2, \ldots, v_n\}$.

Věta 3.2.5 *Algoritmus je konečný a korektní.*

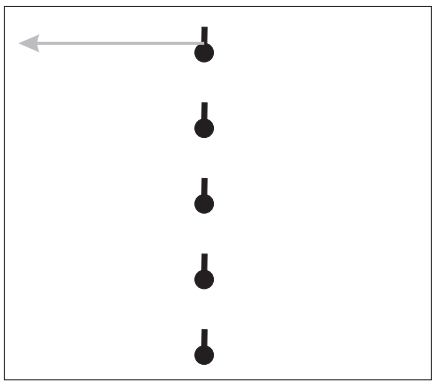
Důkaz: Algoritmus lineárně prochází konečnou posloupnost vrcholů, je tedy konečný.

Korektnost plyne z konečnosti a 3.2.4.

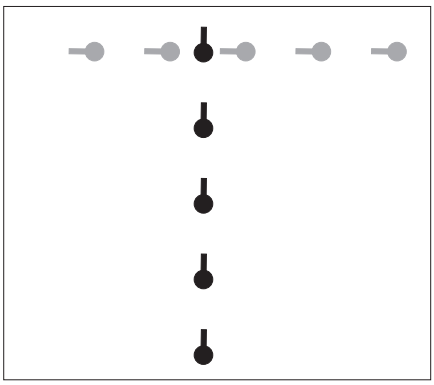
Poznámka 3.2.6 *: Pro fragmentaci po zatáčkách není vhodné, aby vznikaly příliš krátké fragmenty, ze stejných důvodů, jako je popsáno u 3.2.1. Pokud by měl vzniknout příliš krátký fragment, použije se fragment stanovené minimální délky* $(n = 3)$.

3.2.3 Adaptace formace

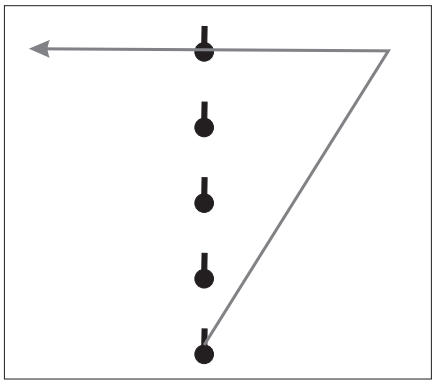
Oba způsoby řešení, tedy periodické přeplánování a fragmentace po zatáčkách, mají své výhody, avšak jejich implementace odkrývá závažný problém, který se nejvíce projeví na ostrých zatáčkách dlouhé formace. Na obrázku 3.8a je znázorněna formace čára, která má rozkaz k pohybu vlevo[3]. Standardní plán, který všechny metody (včetně triviální) vytvoří, bude podobný tomu na obrázku 3.8b. Je nabíledni, že pro jednotku, která je nejvíce vzadu, vznikne cesta, která je vyznačena na obrázku 3.8c. Je možné, že plánování v dalších fragmentech tuto cestu mírně zlepší, stejně tak je-li v okolí náročnější terén (tedy formace se pohybuje po cestě), nastane vychýlení z cesty později. Cesta pak bude vypadat tak, jak je znázorněno na obrázku 3.8d. To má ovšem do ideální cesty daleko — jednotka prochází část cesty zcela zbytečně. Navíc, je-li terén mimo optimální trasu náročnější, je tím formace i neúměrně zpomalena.



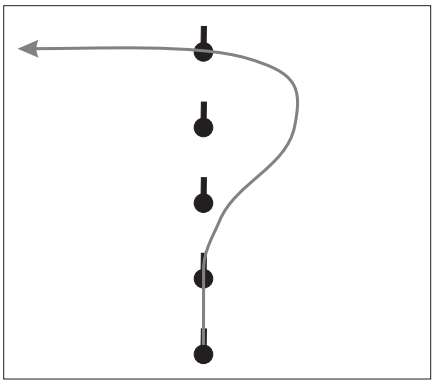
(a) Stav formace s vyznačením jejího záměru



(b) Plán vytvořený standardním způsobem



(c) Naplánovaná cesta poslední jednotky



(d) Skutečná cesta poslední jednotky

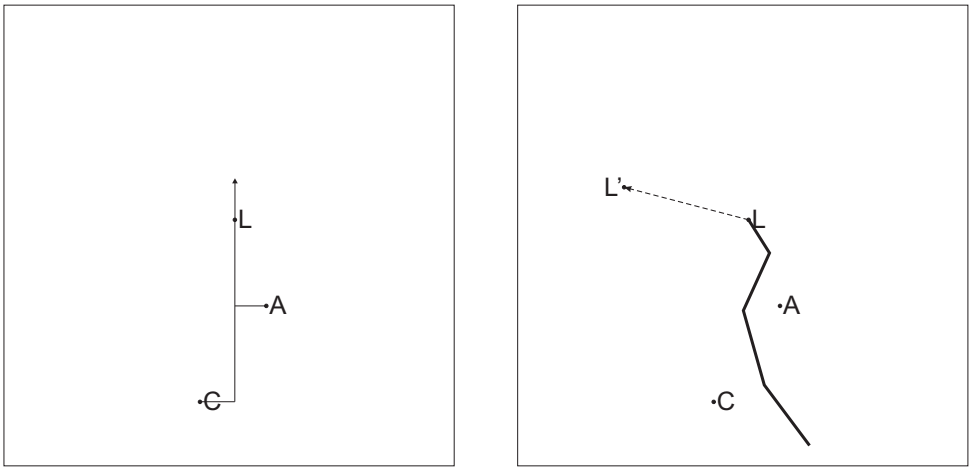
Obrázek 3.8: Ilustrace problému dlouhých formací v zatáčkách

Příčinou tohoto problému je způsob, jakým se tvoří plán pro jednotky ve formaci (viz obrázek 2.1). Jednotky se plánují podle toho, jakým směrem má být

[3]Je lhostejné, zda se jedná o přímý rozkaz formaci, nebo daná situace nastala v určitém fragmentu cesty.

formace na cílové pozici natočena. Dokud není tento směr výrazně odlišný od současné orientace, je všechno v pořádku. Ve výše uvedeném příkladě tomu tak ale není, a tudíž vzniká popisovaný problém.

Základním kamenem řešení je rozdělit formaci na dvě části, z nichž každá bude mít jiné natočení, tak, aby trajektorie jednotek byla co nejvíce přirozená. Pro přední polovinu formace to neznamená žádnou změnu — bude orientovaná tím směrem, kterým se má celá formace pohybovat (tedy v našem případě vlevo). Pro zadní polovinu to bude směr co nejbližší směru, jaký měla formace předtím, než zatočila. Při plánování každého fragmentu pak upravíme způsob, jakým se bude vybírat cílové místo pro ostatní jednotky. Ty jednotky, jejichž cílová pozice je ve standardním algoritmu před pozicí, ve které je momentálně velitel, budou považovány za předek formace a jejich plán zůstane beze změny. Všechny ostatní jednotky budou považovány za zadní část formace, a naplánují se tak, jako kdyby velitel formace ze současné pozice pokračoval stále stejným směrem.



(a) Stav formace s vyznačením orientace a souřadnic závislých jednotek

(b) Předchozí cesta a plán velitele formace

Obrázek 3.9: Příklad dlouhé formace

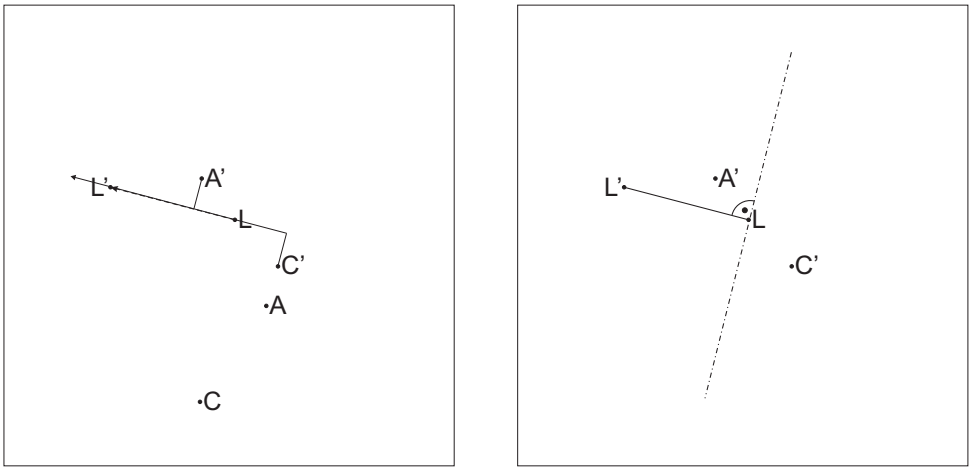
Implementace této myšlenky není obtížná. Mějme například formaci, jako je na obrázku 3.9a — L je velitel formace, A a C jsou další jednotky[4]. Dále je zde vyznačena současná orientace formace a souřadnice jednotek ve formaci[5]. Budeme si pamatovat cestu, kterou již velitel formace prošel[6] — tučnou lomenou čáru. Tu nám spolu s plánem velitele (úsečka LL') ukazuje obrázek 3.9b. Při plánování budeme nejprve postupovat standardně. Vytvoříme plán pro všechny jednotky — na obrázku 3.10a představuje bod A' (resp. C') plán pro jednotku A (resp. C).

[4]Pro přehlednost uvádíme pouze dvě vybrané jednotky.

[5]Předpokládáme, že tyto jsou v optimálním umístění.

[6]Nepotřebujeme cestu celou, ale stačí nám úsek, který je tak dlouhý, aby se na něj vešla celá formace.

Osa úsečky LL' procházející bodem L pak rozděluje formaci na přední a zadní část. Z 3.10b tedy vidíme, že jednotka A je v přední části a jednotka C v zadní části.



(a) Standardní plán pro formaci (b) Hranice mezi částmi

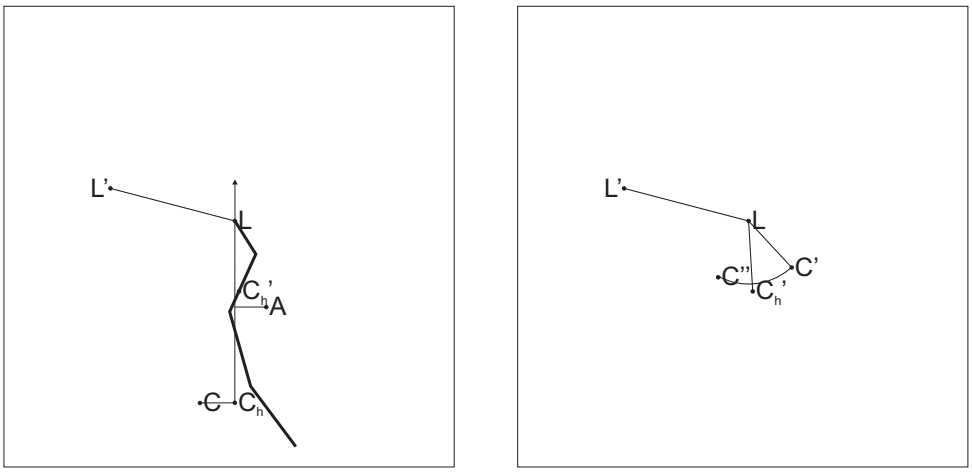
Obrázek 3.10: Rozdělení dlouhé formace na přední a zadní část

Plán pro jednotku A tedy ponecháme, a dále uvažujeme pouze jednotku C. V historii cesty velitele najdeme takový bod C'_h, aby platilo

$$|LL'| + |LC'_h| = |LC_h| \,,$$

kde bod C_h je projekce bodu C na osu (směr) formace. Jelikož z 3.10b je zřejmé, že $|L'L| < |L'C|$, víme, že pokud je historie velitele dostatečně dlouhá, můžeme bod C'_h najít[7]. Bod C'_h nám definuje otočení formace v zadní části — víme, že formace je otočena o $180° - |\angle L'LC'_h|$. Stačí tedy okolo bodu L otočit bod C'_h o stejný úhel, čímž získáme bod C''_h (viz 3.11b), což je hledaná cílová poloha jednotky C.

[7]Kvůli tomu, že naše mapa je diskrétní, vezmeme místo C'_h vrchol, který je tomuto bodu nejblíže.



(a) Nalezení historické polohy velitele

(b) Konstrukce polohy velitele, pokud by neměnil směr

Obrázek 3.11: Plánování jednotek v zadní části formace

Zdá se, že jednotka ze zadní části formace v tomto případě není vychýlená doprava, jak tomu bylo při jednoduchém způsobu plánování, což byl náš cíl.

3.3 Synchronizace příchodu

Jeden z důležitých aspektů, které má adaptivní formace splňovat, je co nejmenší odchylka vzdálenosti dvou jednotek při pohybu. Má-li jedna jednotka výrazně kratší cestu než jiná (například má méně náročný terén, nebo je ve vnitřní straně zatáčky), začne se od ní v průběhu cesty vzdalovat, a navíc na cílové místo dorazí dříve. Myšlenkou synchronizace příchodu je to, že pokud takovou jednotku zpomalíme, pak i v průběhu cesty bude formaci držet lépe.

Pokud používáme nějakou formu fragmentace cesty (viz 3.2.1 nebo 3.2.2), budeme synchronizaci příchodu provádět v každém fragmentu cesty zvlášť. Až na poslední fragment se tedy vlastně o žádnou synchronizaci příjezdu nejedná, protože než velitel dorazí do cíle fragmentu, naplánuje se další fragment.

Základem je určit maximální cenu cesty v daném fragmentu (tedy cenu cesty nejpomalejší jednotky). Následně u všech rychlejších jednotek upravíme rychlost průchodu nebo dokonce cestu samotnou. Je několik možných přístupů, jak toto udělat.

3.3.1 Multiplikativní změna rychlosti

Nechť M je cena cesty nejpomalejší jednotky (tedy maximální cena cesty) a m cena cesty synchronizované jednotky. Pro danou jednotku pak cenu každé hrany na její cestě vynásobíme $\frac{M}{m}$. Jelikož $M \geq m$, tak $\frac{M}{m} \geq 1$ a tedy cena žádné hrany se nezmenší. Protože $\frac{M}{m} \cdot m = M$, dorazí všechny jednotky do cíle ve stejný okamžik jako nejpomalejší z nich.

Multiplikativní změna rychlosti tedy funguje, ale nikterak dobře. U konkrétní jednotky může docházet k větším výkyvům rychlosti, pokud se na cestě střídají levné a drahé hrany. Na drahé hraně se pak jednotka může hýbat výrazně pomaleji, než je nezbytně nutné, a na levnějších hranách bude stále výrazně rychlejší, než nejpomalejší jednotka.

3.3.2 Aditivní změna rychlosti

Nechť M je maximální cena cesty, m cena cesty synchronizované jednotky a n její délka co do počtu hran. Pro danou jednotku pak k ceně každé hrany na její cestě přičteme $\frac{M-m}{n}$. Jelikož $M \geq m$, tak $\frac{M-m}{n} \geq 0$ a tedy cena žádné hrany se nezmenší. Protože $m + \frac{M-m}{n} \cdot n = M$, dorazí všechny jednotky do cíle ve stejný okamžik, jako nejpomalejší z nich.

3.3.3 Adaptivní změna rychlosti

Nechť M je maximální cena cesty, $c_1, \dots c_n$ buď ceny hran na cestě synchronizované jednotky. Vhodně zvolíme práh D a přepočítáme ceny hran jako $d_i = \max\{c_i, D\}$, čímž se cena žádné hrany nezmenší. Pokud konstantu D zvolíme tak, aby $\sum d_i = M$, pak dorazí všechny jednotky do cíle ve stejný okamžik, jako nejpomalejší z nich.

Algoritmus pro nalezení prahu D je následující:

$d_i \leftarrow c_i$
while $\sum d_i < M$ **do**
  $m \leftarrow \sum d_i$
  $S \leftarrow \min d_i$
  $T \leftarrow \min\{d_i \neq S\}$
  $c \leftarrow |\{d_i = S\}|$
  if $m + c \cdot (T - S) < M$ **then**
    $d_i \leftarrow \max(d_i, T)$
  else
    $d_i \leftarrow \max(d_i, \frac{M-m}{c} + S)$
  end if
end while
return $\min d_i$

Kód 3.2: Algoritmus pro nalezení prahu D pro adaptivní synchronizaci příchodu

Lemma 3.3.1 *Po celou dobu běhu platí, že* $d_i \geq c_i$.

Důkaz: Na začátku běhu algoritmu je $d_i = c_i$. Každé další přiřazení do d_i má tvar $d_i \leftarrow max(d_i, K)$, tedy d_i se nikdy nezmenší.

Lemma 3.3.2 *Pokud algoritmus skončí, pak platí* $\sum d_i = M$.

Důkaz: V jedné iteraci cyklu nastane vždy jedna ze dvou možností:

1. Platí-li vnitřní podmínka $(m + c \cdot (T - S) < M)$, pak c hran změní svoji cenu z S na T. Následně tedy bude platit

$$\sum d_i = m - cS + cT = m + c \cdot (T - S) < M$$

   a běh algoritmu bude pokračovat.

2. V opačném případě c hran změní svoji cenu z S na $\frac{M-m}{c} + S$. Bude tedy platit
$$\sum d_i = m - cS + c \cdot \left(\frac{M-m}{c} + S \right) = M$$
a algoritmus zároveň skončí.

Lemma 3.3.3 *Je-li v nějaké iteraci $d_j = S$, pak v příští iteraci bude taktéž platit $d_j = S$.*

Důkaz: Pokud algoritmus neskončil, pak bylo do d_j přiřazeno T. d_j je tak nové minimum.

Věta 3.3.4 *Pokud algoritmus skončí, pak $D = \min d_i$ je hledaný práh.*

Důkaz: Chceme dokázat, že $d_i = \max\{c_i, D\}$. Z 3.3.1 víme, že $d_i \geq c_i$. Z definice D pak platí, že $d_i \geq D$. Pak buď $d_i = c_i$ a jsme hotovi, nebo $d_i > c_i$. V takovém případě muselo být d_i v některém cyklu zvýšeno. To znamená, že se ale jednalo o minimum, které už bylo podle 3.3.3 minimem vždy. Tedy $d_i = D$. Z toho a z 3.3.2 plyne, že algoritmus je korektní.

Lemma 3.3.5 *Algoritmus skončí po nejvýše n iteracích vnějšího cyklu.*

Důkaz: V každé iteraci, po které algoritmus neskončí, musí platit $m + c \times (T - S) < M$. Počet různých hodnot mezi d_i se tím zmenšil o 1 (protože všechny S se změnily na T, které již mezi d_i bylo). Na začátku běhu algoritmu bylo různých hodnot nejvýše n, zároveň jich však nikdy nemůže být nikdy méně než 1. Může tedy proběhnout maximálně $n - 1$ iterací, po kterých algoritmus neskončí, tedy nejvýše n celkově.

Věta 3.3.6 *Časová složitost algoritmu je $O(n^2)$, kde n je délka cesty v počtu hran.*

Důkaz: Cyklus může proběhnout maximálně n-krát, časová složitost jednoho běhu cyklu je triviálně $O(n)$. Algoritmus je tedy konečný.

Poznámka 3.3.7 *Časovou složitost je možné ještě zredukovat na $O(n \log n)$. Na hledání minima můžeme totiž použít haldu ($O(\log n)$). Přiřazování do d_i není potřeba, jelikož vždy přiřazujeme jenom hodnotu rovnou novému minimu, takže si stačí pamatovat aktuální počet minim ($O(1)$). Součet $\sum d_i$ stačí spočítat na začátku běhu a poté jen upravovat stejnými odhady, jako jsme použili v důkazu 3.3.2 ($O(1)$). Tato implementace však nebyla použita, jelikož hodnota n je řádově 10, a tak by algoritmus prakticky rychlejší nebyl.*

V tabulce 3.1 vidíme porovnání jednotlivých metod, které jsme pro synchronizaci příchodu navrhli. Levý sloupec obsahuje ceny hran na cestě, kde neproběhla žádná synchronizace příchodu. Ostatní tři sloupce představují hodnoty po synchronizaci různými metodami pro $M = 24$.

	Žádná	Multiplikativní	Aditivní	Adaptivní
	1,30	1,73	1,70	1,40
	1,40	1,87	1,80	1,40
	1,20	1,60	1,60	1,40
	0,60	0,80	1,00	1,40
	0,60	0,80	1,00	1,40
	0,80	1,07	1,20	1,40
	0,60	0,80	1,00	1,40
	0,60	0,80	1,00	1,40
	0,90	1,20	1,30	1,40
	1,10	1,47	1,50	1,40
	2,60	3,47	3,00	2,60
	3,10	4,13	3,50	3,10
	1,50	2,00	1,90	1,50
	1,00	1,33	1,40	1,40
	0,70	0,93	1,10	1,40
$\sum$	18,00	**24,00**	**24,00**	**24,00**

Tabulka 3.1: Porovnání metod synchronizace příchodu

3.4 Detekce kolizí

Jak bylo již uvedeno v kapitole 2, triviální algoritmus nevyvíjí vůbec žádné úsilí pro detekci a řešení kolizí. Zdá se, že různé způsoby plánování spolu se synchronizací příchodu způsobují, že kolize nastávají méně často, avšak nemizí úplně. Proto je třeba navrhnout explicitní algoritmus pro detekci kolizí a jejich následné řešení.

Ke kolizi dvou jednotek může dojít pouze v případě, že cesta obou obsahuje stejný vrchol v. Kvůli způsobu, jakým jsou cesty jednotek plánovány, to nemůže být pro obě jednotky cíl cesty. Jakákoliv metoda synchronizace příchodu zaručí, že ke kolizi nedojde, pokud je v cílovým vrcholem pro právě jednu z jednotek. Podobnou úvahou lze vyloučit i start cesty.

Stačí tedy řešit případy, ve kterých je bod v součástí cesty, a tedy víme, že mu nějaký bod předchází a nějaký následuje. Můžeme tedy předpokládat, že cesty jsou tedy tvaru:

$$\begin{aligned} P_1 &= (\ldots, p_1, v, q_1, \ldots) \\ P_2 &= (\ldots, p_2, v, q_2, \ldots) \ . \end{aligned}$$

Detekce kolidujících jednotek má časovou složitost $O(n^2)$, kde n je počet jednotek na mapě. Vhodným hešováním vrcholů cesty lze dosáhnout toho, že tato složitost bude nižší při velkém počtu jednotek na mapě.

3.4.1 Změna fyzického umístění

Jedno z možných řešení je ponechání plánu jednotek a změna jeho vykonání. K tomu využijeme definici logického a fyzického umístění (viz 1.2). Ponechání plánu jednotky znamená, že logické umístění nebudeme měnit. Fyzické umístění však nebude vždy odpovídat logickému, ale v případě hrozby kolize bude mírně vychýlené.

Každá jednotka bude mít svoji vlastní detekci kolizí. Pokud se v její blízkosti a zorném poli vyskytuje jiná jednotka, bude to znamenat, že hrozí kolize. Nebudeme tedy brát v úvahu jednotky, které stojí za námi, protože ty se k nám nepřibližují.

Způsob řešení kolizí bude prostý. Hrozí-li kolize, zvětšíme vychýlení jednotky ve směru kolmém na směr pohybu jednotky; pokud kolize nehrozí, zmenšíme jej.

To nám dává následující algoritmus:

if someoneIsInSight() **then**
  $o^+ \leftarrow \mu * |\mathbf{v}| * t$
  $\mathbf{o} \leftarrow \mathbf{o} + o^+ * \text{unit}(\text{normal}(\mathbf{v}))$
else
  $o^- \leftarrow \min(|\mathbf{o}|, \nu * |\mathbf{v}| * t)$
  $\mathbf{o} \leftarrow \mathbf{o} - o^- * \text{unit}(\mathbf{o})$
end if
return **o**

Kód 3.3: Algoritmus pro detekci kolizí a jejich řešení změnou fyzického umístění

Vektor $\mathbf{o}$ značí vychýlení (*offset*) fyzického umístění jednotky oproti logickému a vektor $\mathbf{v}$ je okamžitá rychlost (*velocity*). Vektorová funkce $||$ vrací velikost vektoru, normal vrací normálový vektor a unit jednotkový vektor. t je časová délka (*time*) současné iterace. Parametry μ a ν určují rychlost, s jakou vychýlení (resp. návrat do logického umístění) nastává.

Při snižování vychýlení je třeba dát pozor, aby toto neoscilovalo kolem nuly — proto je velikost snížení vychýlení shora omezená současnou velikostí vychýlení. Je-li pak někdy

$$\nu * |\mathbf{v}| * t \geq |\mathbf{o}| \ ,$$

pak platí

$$o^- = |\mathbf{o}| \ .$$

Jelikož

$$\mathbf{o} = |\mathbf{o}| * \text{unit}(\mathbf{o}) \ ,$$

pak poslední přiřazení má tvar

$$\mathbf{o} \leftarrow \mathbf{0} \ ,$$

tedy nové vychýlení je nulové.

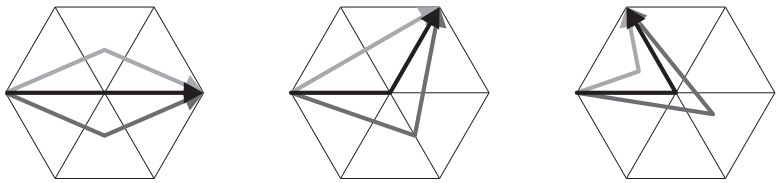
Způsob, jakým vychýlení upravujeme, způsobuje, že jednotka se někdy pohybuje rychleji, než by po hraně měla. Jednoduše však můžeme ukázat, že toto urychlení je nejhůř konstantní. Délka úseku, který by jednotka urazila, pokud by k žádné kolizi nedošlo, je $|\mathbf{v}| * t$. Jednotka je vychýlena o $\mu * |\mathbf{v}| * t$ (resp. $\nu * |\mathbf{v}| * t$) ve směru kolmém na směr pohybu. Celková délka úseku, který projde, bude tedy rovna $\sqrt{1 + \mu^2} * |\mathbf{v}| * t$ (resp. $\sqrt{1 + \nu^2} * |\mathbf{v}| * t$), čili vzdálenost lineární vzhledem k $|\mathbf{v}| * t$.

Toto řešení sice kolizím nezabrání vždy, avšak jeho úspěšnost je poměrně vysoká. Výhodou tohoto řešení je jeho snadná implementace a také časová složitost, která je $O(n)$, kde n je počet jednotek. Aby nebylo třeba testovat potenciálně

mnoho jednotek na mapě na vzdálenost a viditelnost ve funkci someoneIsInSight(), je možné pro vyhledávání blízkých jednotek použít vhodný kd-strom (viz [5]).

3.4.2 Vychýlení středového bodu

Druhou možností, jak řešit problém kolizí, je změnit cestu jednotky z p_i do q_i tak, aby ke kolizi nedošlo, což je možné provést vychýlením bodu v na jednu či druhou stranu mimo vrcholy a hrany. Podle úhlu, který spolu svírají hrany e_{p_iv} a e_{vq_i}, můžeme rozlišit tři různé průchody (viz obrázek 3.12): přímý, pod ostrým úhlem (60°) a pod tupým úhlem (120°). Pro každý z nich můžeme bod v vychýlit do dvou stran po ose přesunu a tím získat dvě nové možné cesty.



(a) Alternativní cesty pro přímý průchod

(b) Alternativní cesty pro průchod pod tupým úhlem

(c) Alternativní cesty pro průchod pod ostrým úhlem

Obrázek 3.12: Alternativní cesty pro různé typy průchodu vrcholem

Tyto cesty potom můžeme použít jako náhradu původní cesty, abychom tím zabránili kolizi. Délky těchto cest vyjádřené jako násobek délky původní cesty shrnuje tabulka 3.2. Pokud bychom chtěli použít tyto vychýlené cesty jako náhradu cesty původní, a zachovat při tom délku průchodu hranou (abychom nemuseli znovu dělat synchronizaci příchodu), není dobré, aby byla nová cesta o mnoho delší. Proto budeme vždy používat pouze kratší z obou alternativních cest. Při přímém průchodu můžeme použít obě, jelikož jsou stejně dlouhé.

	Délka cesty	
Úhel	Levá	Pravá
0°	0.5	1.5
60°	$\sqrt{0.4375}$	$\sqrt{2.4375}$
120°	$\sqrt{0.75}$	$\sqrt{1.75}$
180°	$\sqrt{1.1875}$	$\sqrt{1.1875}$
240°	$\sqrt{1.75}$	$\sqrt{0.75}$
300°	$\sqrt{2.4375}$	$\sqrt{0.4375}$

Tabulka 3.2: Délky průchodů cest s vychýleným středovým bodem

Nyní je třeba pro každou jednotku vybrat vhodnou cestu tak, aby výsledné průchody pro jednotky neobsahovaly kolize. K tomu potřebujeme nejprve několik definicí:

Definice 3.4.1 *Mějme dvě funkce* $f, g : [0,1] \to \mathbb{R}^n$. ***Nejbližší přiblížení dvojice funkcí*** *je minimum funkce* $D(t) = d(f(t), g(t))$ *na intervalu* $[0,1]$, *kde* $d : \mathbb{R}^n \to \mathbb{R}$ *je libovolná metrika. Tuto hodnotu budeme značit* $\mathrm{ca}(f, g)$.

Definice 3.4.2 *Mějme* F *množinu (posloupnost) funkcí* $f : [0,1] \to \mathbb{R}^n$. ***Nejbližší přiblížení množiny*** F *je* $min\{\mathrm{ca}(f, g) \mid f, g \in F, f \neq g\}$. *Tuto hodnotu budeme značit* $\mathrm{ca}(F)$.

Definice 3.4.3 *Systémem funkcí rozumíme posloupnost* $\mathcal{F} = (F_i)_{i=1}^m$, *kde* F_i *je množina funkcí* $f : [0,1] \to \mathbb{R}^n$. *Podsystémem funkcí rozumíme takovou posloupnost funkcí* $\tilde{\mathcal{F}} = (f_i)_{i=1}^m$, *kde* $\forall i \in 1, \ldots, m : f_i \in F_i$. ***Nejvzdálenější posloupnost funkcí*** *systému* $\mathcal{F}$ *je takový podsystém* $\mathcal{F}^+$, *pro který je* $\mathrm{ca}(\mathcal{F}^+)$ *maximální.*

V našem případě bude množina F_i obsahovat různé průchody jedné jednotky, a posloupnost $\mathcal{F}$ bude sdružovat množiny průchodů všech jednotek, kterých se průchod bodem v týká. $\mathcal{F}^+$ potom obsahuje po řadě nejlepší průchody pro každou jednotku.

Zbývá vyřešit, jak zjistit jednotlivé hodnoty $\mathrm{ca}(f, g)$. V našem případě používáme prostor $\mathbb{R}^2$ a euklidovskou metriku. Každý průchod má tvar dvou lineárních funkcí za sebou (viz obrázek 3.13a). Jelikož musíme vzít v úvahu i náročnost terénu, nemusí jedna lineární funkce přecházet v druhou v obou funkcích ve stejném bodě. Interval $[0,1]$ však můžeme rozdělit na 3 různé podintervaly, ve kterých jsou obě funkce lineární (pro prostor $\mathbb{R}$ ukazuje rozdělení na podintervaly obrázek 3.13b). Budeme-li mít minimum na každém z těchto podintervalů, snadno získáme minimum na celém intervalu $[0,1]$.

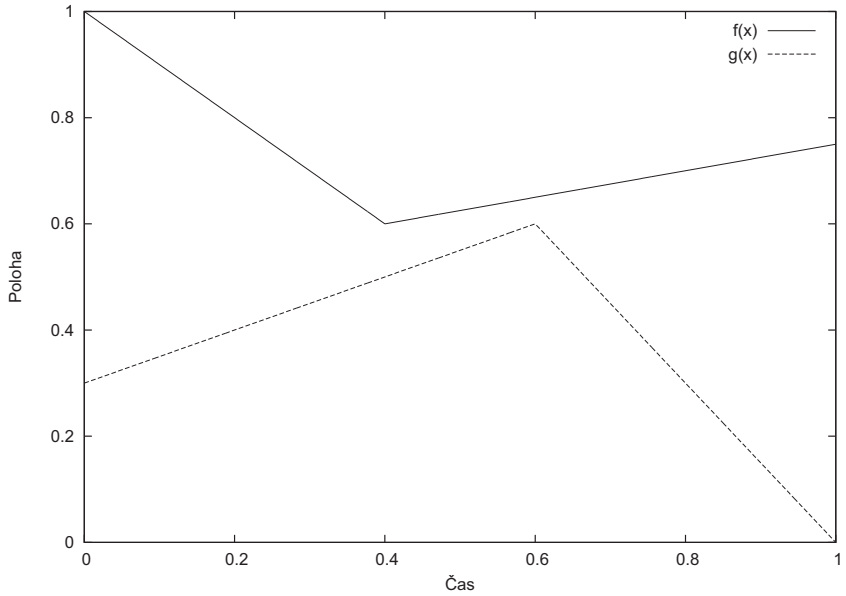
Minimum euklidovské vzdálenosti dvou lineárních funkcí najdeme jednoduchým postupem. Nejprve tyto vyjádříme jako

$$\begin{aligned} x &= x_0 + t x_s \\ y &= y_0 + t y_s \\ \tilde{x} &= \tilde{x}_0 + t \tilde{x}_s \\ \tilde{y} &= \tilde{y}_0 + t \tilde{y}_s \end{aligned}$$

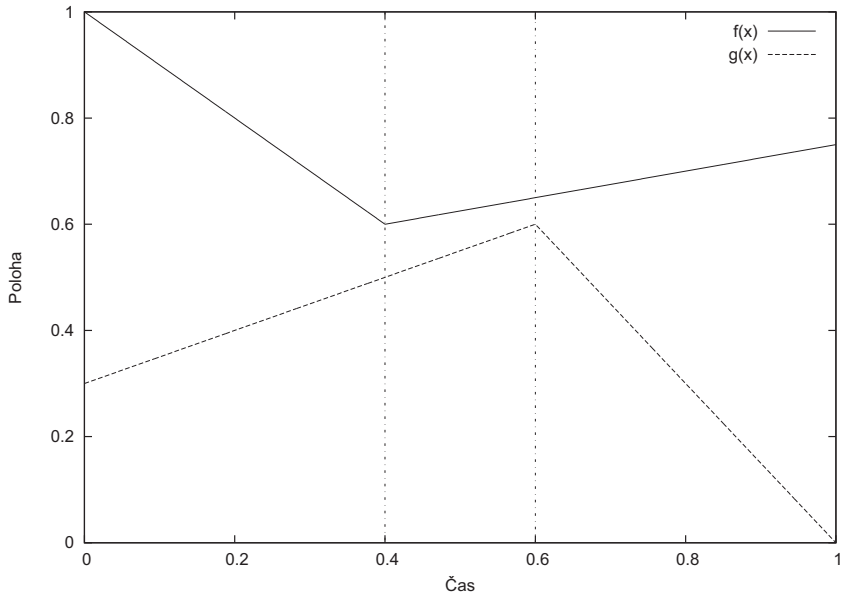
pro $t \in [0,1]$.

Hledáme extrémy funkce $d(f(t), g(t))$ na kompaktní množině $t \in [0,1]$.

$$D = \sqrt{(x - \tilde{x})^2 + (y - \tilde{y})^2}$$



(a) Lomené lineární funkce



(b) Rozdělení intervalu na podintervaly, kde jsou funkce lineární

Obrázek 3.13: Rozdělení intervalu

Jelikož $f(x) = \sqrt{x}$ je funkce rostoucí[8], stačí nám hledat extrémy funkce

$$\begin{aligned} \tilde{D} &= (x - \tilde{x})^2 + (y - \tilde{y})^2 \\ &= \left[(x_0 + tx_s) - \left(\tilde{x}_0 + t\tilde{x_s}\right)\right]^2 + \left[(y_0 + ty_s) - \left(\tilde{y}_0 + t\tilde{y_s}\right)\right]^2 \end{aligned}$$

Určíme tedy první derivaci této funkce

$$\frac{\partial \tilde{D}}{\partial t} = 2\left(x_s - \tilde{x}_s\right)\left[(x_0 + tx_s) - \left(\tilde{x}_0 + t\tilde{x_s}\right)\right] + 2\left(y_s - \tilde{y}_s\right)\left[(y_0 + ty_s) - \left(\tilde{y}_0 + t\tilde{y_s}\right)\right]$$

[8]Je-li f ryze monotónní funkce, platí: g má v x_0 extrém $\Rightarrow$ složená funkce $f \circ g$ má v x_0 extrém.

$$= \quad 2\left[(x_0 - \tilde{x}_0)(x_s - \tilde{x}_s) + (y_0 - \tilde{y}_0)(y_s - \tilde{y}_s)\right] + 2t\left[(x_s - \tilde{x}_s)^2 + (y_s - \tilde{y}_s)^2\right]$$

a položíme ji rovnu nule.

Nyní máme dvě možnosti:

1. Buď platí, že
$$(x_s - \tilde{x}_s)^2 + (y_s - \tilde{y}_s)^2 = 0 \ ,$$
čili $x_s = \tilde{x}_s$ a $y_s = \tilde{y}_s$. Potom je ale
$$\forall t \in \mathbb{R} : \frac{\partial \tilde{D}}{\partial t}(t) = 0 \ ,$$
a tedy D má (neostré) globální minimum např. v $t_0 = 0$.

2. V opačném případě musí platit
$$(x_s - \tilde{x}_s)^2 + (y_s - \tilde{y}_s)^2 > 0 \ .$$
Potom D může nabývat lokálního extrému na $\mathbb{R}$ pouze v bodě
$$t_0 = -\frac{(x_0 - \tilde{x}_0)(x_s - \tilde{x}_s) + (y_0 - \tilde{y}_0)(y_s - \tilde{y}_s)}{(x_s - \tilde{x}_s)^2 + (y_s - \tilde{y}_s)^2} \ .$$
Druhá derivace
$$\frac{\partial^2 \tilde{D}}{\partial t^2} = 2\left[(x_s - \tilde{x}_s)^2 + (y_s - \tilde{y}_s)^2\right] > 0$$
je vždy kladná, D má tedy v t_0 lokální (a dokonce globální) minimum.

Z toho plyne:

$$\min D = \begin{cases} D(0) & \text{pro } t_0 \in (-\infty, 0) \\ D(t_0) & \text{pro } t_0 \in [0, 1] \\ D(1) & \text{pro } t_0 \in (1, \infty) \end{cases}$$

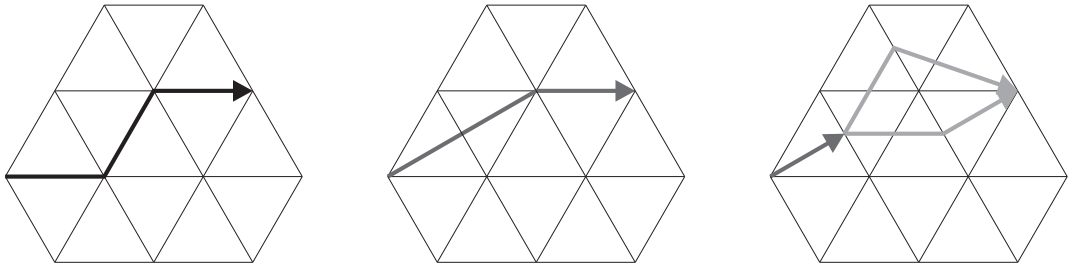
Máme tedy algoritmus, který určí hodnotu nejbližšího přiblížení dvou funkcí f a g v čase $O(1)$. Hodnotu nejbližšího přiblížení množiny funkcí F pak vyčíslíme dle definice v čase $O\left(2^{|F|}\right)$. Nejvzdálenější posloupnost funkcí pak určíme taktéž podle definice v čase
$$O\left(2^{|F|} \cdot \prod_{i=1}^{|F|} |F_i|\right) \ .$$

Tento algoritmus nyní využijme pro detekci a řešení kolizí. Vždy, když máme nějakou množinu jednotek T, která má stejný následující vrchol cesty v, zkonstruujeme pro každou jednotku $t_i \in T$ množinu cest P_i, která bude obsahovat původní cestu, a také alternativní cesty (viz obrázek 3.12). Sestrojíme systém funkcí $\mathcal{P} = (P_i)_{i=1}^{|T|}$, a najdeme nejvzdálenější posloupnost funkcí $\mathcal{P}^+$. Jednotce t_i pak přiřadíme cestu $\mathcal{P}_i^+$, která je nejlepším řešením kolizí ve smyslu vychýlení středového bodu.

Jelikož $|P_i| \leq 3$, pak se složitost algoritmu redukuje na $O\left(6^{|T|}\right)$, což pro malé $|T|$ dává rozumné hodnoty. Bude-li na jednom místě větší množství jednotek, není časově zvládnutelné spočítat všechny kombinace. Ukazuje se, že to však ani není příliš třeba. Velké množství jednotek na jednom místě totiž znamená, že určitému počtu kolizí nelze tímto algoritmem zabránit (protože se na takové malé místo zkrátka tolik jednotek nevejde).

V takovém případě si však můžeme pomoci pravděpodobnostní aproximací. Při dané kolizní situaci nebudeme testovat všechny kombinace alternativních cest, ale pouze polynomiálně velkou podmnožinu. Tímto dostaneme aproximaci nejvzdálenější posloupnosti funkcí, která se pravděpodobně od optimálního řešení nebude příliš lišit.

Může se nám samozřejmě stát, že kolize nastane pro určitou jednotku na dvou či více po sobě jdoucích vrcholech. V takovém případě je třeba v dalších vrcholech již počítat s předchozími vrcholy jako s vychýlenými, a tedy alternativní cesty jsou jiné. Na obrázku 3.14a máme vyznačen průchod dvěma vrcholy. V prvním vrcholu nastane kolize. Ta je vyřešena nalezením alternativní cesty na obrázku 3.14b. V druhém vrcholu cesty však má též dojít ke kolizi, proto je třeba zvážit alternativní průchody. Ty jsou (kromě původně nalezené cesty, která bude též brána v úvahu) zobrazeny na obrázku 3.14c.



(a) Původní cesta

(b) Vybraná alternativní cesta pro průchod prvním vrcholem

(c) Alternativní cesty pro druhý vrchol

Obrázek 3.14: Dva alternativní průchody za sebou

Kapitola 4

Metodika hodnocení

4.1 Statická fitness

Pro sledování úspěšnosti formace je vhodné definovat systém automatického hodnocení. Pro tento účel definujeme fitness[1] formace v daném časovém okamžiku.

4.1.1 Fitness vzdálenosti

Prvním faktorem, který definuje, jak je formace dobrá, jsou vzdálenosti mezi jednotkami. Vzdálenosti mezi některými dvojicemi jednotek jsou totiž ve schématu formace přesně definovány. Fitness vzdálenosti je definována jako

$$f^D = \frac{1}{|B|} * \sum_{b \in B} \left(|\mathbf{d}(b)| - |\mathbf{o}(b)|\right)^2 \ .$$

4.1.2 Úhlová fitness

Druhým důležitým faktorem jsou úhly, které jednotky ve formaci svírají. Ve schématu formace jsou definovány nejen vzdálenosti, ale i azimuty k ostatním bodům. Jelikož otočení celé formace není na škodu, budeme místo azimutů měřit, pod jakým úhlem vidí jednotka všechny různé dvojice bodů, na které má vazbu. Úhlová fitness je definována jako

$$f^A = \frac{1}{|B|} * \sum_{s \in S} |B_s| * f_s^A \ ,$$

[1] Nejedná se o fitness v pravém slova smyslu, jelikož nižší hodnoty zde znamenají lepší vlastnosti formace. Korektnější označení by bylo chyba nebo odchylka.

kde

$$f_s^A = \frac{1}{\binom{|B_s|}{2}} * \sum_{(b_1,b_2)\in\binom{B_s}{2}} \left(\texttt{angleTo}\left(\mathbf{d}(b_1),\mathbf{d}(b_2)\right) - \texttt{angleTo}\left(\mathbf{o}(b_1),\mathbf{o}(b_2)\right)\right)^2 .$$

4.1.3 Kombinovaná fitness

Z každé z těchto vlastností jsme schopni získat informaci o tom, jak je formace úspěšná. Pokud ale potřebujeme, aby fitness byla jednorozměrná (což může být třeba pokud by fitness měla být vstupem některého z modulů, ale i pro snadnější porovnání úspěšnosti několika algoritmů), je třeba tyto dvě hodnoty nějakým způsobem zkombinovat.

Nabízí se několik variant:

- **fitness úhlu**[2]: f^A
- **fitness vzdálenosti**[3]: f^D
- **fitness maxima**: $f^M = \max\left(f^A, f^D\right)$
- **fitness minima**: $f^N = \min\left(f^A, f^D\right)$
- **fitness geometrického průměru**: $f^S = \sqrt{f^A f^D}$
- **fitness aritmetického průměru**: $f^P = \frac{f^A + f^D}{2}$
- **fitness harmonického průměru**: $f^H = 2 \times \left(\frac{1}{f^A} + \frac{1}{f^D}\right)^{-1}$

Za účelem ověření vlastností těchto navrhovaných fitness byl implementován nástroj pro testování a optimalizaci fitness.

První jeho funkcí je možnost přesunovat jednotky ve formaci a sledovat, jak se mění fitness.

Druhá funkce je nazvána Optimalizace fitness. Jejím vstupem jsou jednotky, které mohou být libovolně chaoticky rozmístěné. Funkce hladovým algoritmem optimalizuje fitness formace. Provádí to tak, že vybere náhodně jednotku a přesune jí na takové místo, kde bude kombinovaná fitness celé formace co nejnižší[4]. Jedna iterace tohoto algoritmu představuje jednu lokální optimalizaci každé jednotky ve formaci.

[2]Ignoruje fitness vzdálenosti.

[3]Ignoruje fitness úhlu.

[4]Hodnota je nalezena prohledáním celé mapy, a poté jejích nejlepších částí, ve postupně se zjemňujícím rastru, až na velikost čtverce 8×8. Kvůli velikosti rastru už nejsou rozdíly mezi malými hodnotami fitness významné.

Myšlenkou této metody je zjistit, jak věrně popisuje daná fitness vlastnosti formace. Existence lokálních minim by totiž mohla způsobit, že „lepší“ vzhled formace by mohl být ohodnocen horší fitness. Cílem je tedy vybrat takovou fitness, která způsobuje nejrychlejší a nejpravděpodobnější konvergenci formace k ideálnímu stavu.

Vlastnosti všech fitness funkcí budeme porovnávat v prvních deseti iteracích optimalizačního procesu na vzorku 100 formací s náhodně rozmístěnými jednotkami. Jelikož není možné porovnávat kombinovanou fitness, protože ta má při různém předpisu různou sémantiku, budeme porovnávat úhlovou fitness a fitness vzdálenosti zvlášť. Tabulku 4.1a interpretujeme takto — ve sloupci označeném f^C a řádku i máme výběrový průměr f^A všech 100 testovaných formací po i iteracích, ve kterých byla jako kombinovaná fitness použita funkce f^C. Obdobně interpretujeme ostatní tři tabulky (4.1b, 4.2a, 4.1a).

Iterace	f^A	f^D	f^N	f^M	f^P	f^S	f^H
1	9173	39581	9173	28893	17467	10757	13410
2	3510	36385	3621	6422	3670	2928	2942
3	2282	35656	2656	2058	1785	1923	1672
4	1606	35689	2014	1129	867	1693	1415
5	1392	34433	1801	820	537	1656	1189
6	1192	34473	1498	768	500	1652	829
7	1076	34546	1385	694	493	1650	658
8	1004	34574	1311	676	491	1648	529
9	965	34632	1272	664	491	1646	519
10	941	34559	1249	686	491	1645	516

(a) Výběrový průměr fitness úhlu

Iterace	f^A	f^D	f^N	f^M	f^P	f^S	f^H
1	8504	40926	8504	28962	18037	9163	14187
2	1950	37894	1950	4835	1587	997	947
3	961	36345	961	631	168	110	88
4	434	36641	434	103	24	25	19
5	185	35740	185	28	13	13	13
6	86	35709	86	17	12	11	11
7	53	34441	53	13	10	9	11
8	35	35065	35	12	9	9	8
9	23	35065	23	11	9	9	8
10	15	34402	15	11	9	9	8

(b) Medián fitness úhlu

Tabulka 4.1: Naměřené hodnoty f^A pro různé kombinované fitness

Jak ukazují naměřená data (tabulky 4.1 a 4.2), samotná úhlová fitness, nebo fitness vzdálenosti nestačí. Při použití pouze jedné složky fitness pro optimalizaci druhá taktéž mírně klesá, avšak výrazně pomaleji. Lokální optimum pro úhlovou

Iterace	f^A	f^D	f^N	f^M	f^P	f^S	f^H
1	290774	14489	290774	18972	16047	46792	20600
2	160629	506	160620	6366	1276	7399	1492
3	122693	181	122212	1521	412	3988	466
4	102648	103	102061	764	246	3453	273
5	90710	68	90156	564	135	3286	211
6	83425	49	82926	485	94	3221	177
7	78494	42	78023	461	81	3202	131
8	74986	37	74541	453	77	3189	103
9	71766	33	71348	432	77	3190	88
10	68821	31	68404	429	77	3188	78

(a) Výběrový průměr fitness vzdálenosti

Iterace	f^A	f^D	f^N	f^M	f^P	f^S	f^H
1	263995	14381	263995	18506	15060	42265	18395
2	138079	474	138079	5603	1065	3092	1010
3	103154	163	103154	618	194	644	224
4	84976	64	84976	114	45	353	63
5	73855	24	73855	28	19	283	26
6	67586	15	67586	16	15	249	20
7	61558	12	61558	11	13	247	16
8	58634	10	58634	10	11	242	17
9	53194	9	53194	10	10	239	15
10	50223	9	50223	10	10	237	15

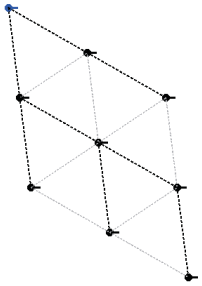
(b) Medián fitness vzdálenosti

Tabulka 4.2: Naměřené hodnoty f^D pro různé kombinované fitness

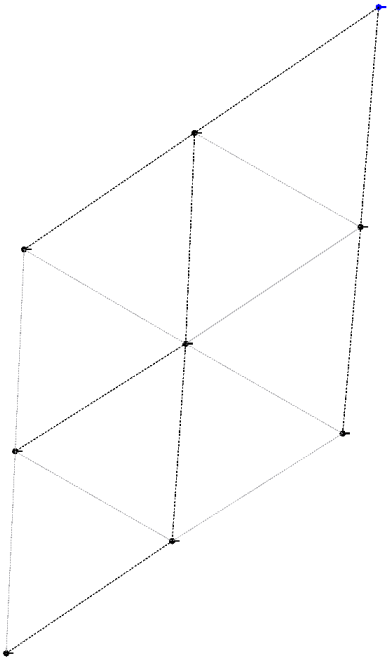
fitness je formace, která má správně všechny úhly, avšak je výrazně větší čí menší (viz obrázek 4.1b). Lokální optimum pro fitness vzdálenosti je formace, která má správně všechny vzdálenosti, avšak v některých svých bodech je několikrát přetočená (viz obrázek 4.1c).

Fitness minima (f^N) se chová velmi podobně, jako samotná úhlová fitness. Vidíme, že výběrový průměr f^D je u fitness minima o méně než 1% nižší, výběrový průměr f^A je o 32% vyšší, mediány f^A a f^D jsou naprosto stejné. Výrazný vzrůst fitness pro f^A však naznačuje, že tato metoda se chová hůře, než samotná úhlová fitness. Zřejmě je to kvůli tomu, že pro většinu formací se špatnou hodnotou fitness je hodnota f^D výrazně vyšší, protože chybové členy vzdálenosti nejsou omezeny, zatímco chybové členy úhlu ano, takže je většinou optimalizována jen f^A.

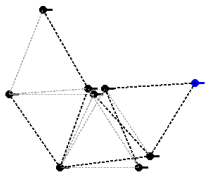
Fitness geometrického průměru (f^S) se chová taktéž podobně, pouze s tím rozdílem, že do stejného lokálního optima konverguje rychleji a s větší pravděpodobností, a přitom optimalizuje i f^D (tedy neprodukuje tak velké formace). f^D dosahuje průměrně o 95% nižší hodnoty.



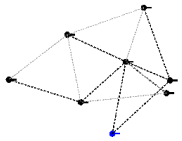
(a) Optimální tvar formace a globální optimum všech fitness funkcí



(b) Lokální optimum pro f^A, f^N a f^S



(c) Lokální optimum pro f^D



(d) Lokální optimum pro f^P, f^M, f^H

Obrázek 4.1: Lokální extrémy různých fitness

Fitness aritmetického průměru (f^P), fitness harmonického průměru (f^H) a fitness maxima (f^M) se všechny chovají podobně a též vracejí podobné výsledky. Rozdíly u mediánů jsou zanedbatelné a zřejmě vznikají kvůli hrubosti mřížky, na které algoritmus probíhá. U středních hodnot jsou rozdíly výraznější, zejména f^M za oběma metodami zaostává v případě fitness vzdálenosti. f^P má mírně lepší hodnoty než f^H, nicméně obě fitness se zdají být vhodné. Všechny tři fitness se shodně ve většině případů vyhýbají lokálním optimům. Pokud se takové již vyskytne, má většinou tvar podobný obrázku 4.1d[5].

[5] Z obrázku to není příliš dobře vidět, ale jednotka uprostřed formace, ze které vede 6 vazeb, je ve skutečnosti složena ze dvou překrytých jednotek, které spolu nejsou spojeny vazbou, a vůči okolním bodům mají poměrně korektní vzdálenosti.

4.2 Dynamická fitness

Statická fitness je užitečná pro zhodnocení stavu formace v konkrétním čase. Nedokáže však vystihnout změny této fitness v čase, ani délku trvání přesunu. Zdánlivě je možné postihnout oba tyto faktory předpisem

$$F = \int_0^T f^C \, \mathrm{d}t \ ,$$

kde f^C je libovolná kombinovaná fitness, t je čas a T délka trvání přesunu. Jelikož naše pozorování fitness jsou diskrétní, můžeme vyjádřit integrál jako sumu

$$F = \sum_{i=0}^{n} t_i f_i^C \ ,$$

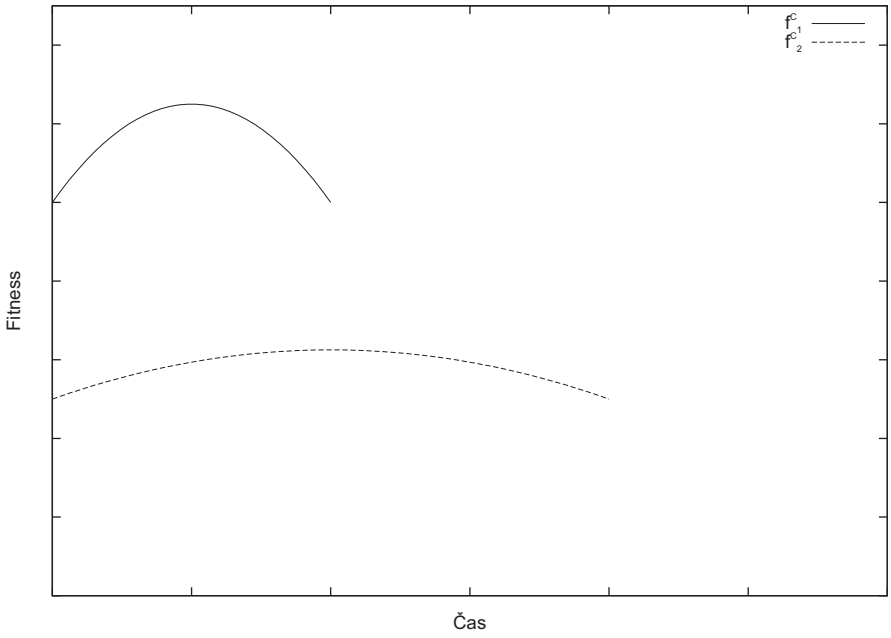
kde f_i^C je hodnota fitness v i-tém pozorování, a t_i je čas, který uplynul od předchozího pozorování.

Tento předpis splňuje několik základních požadavků na fitness:

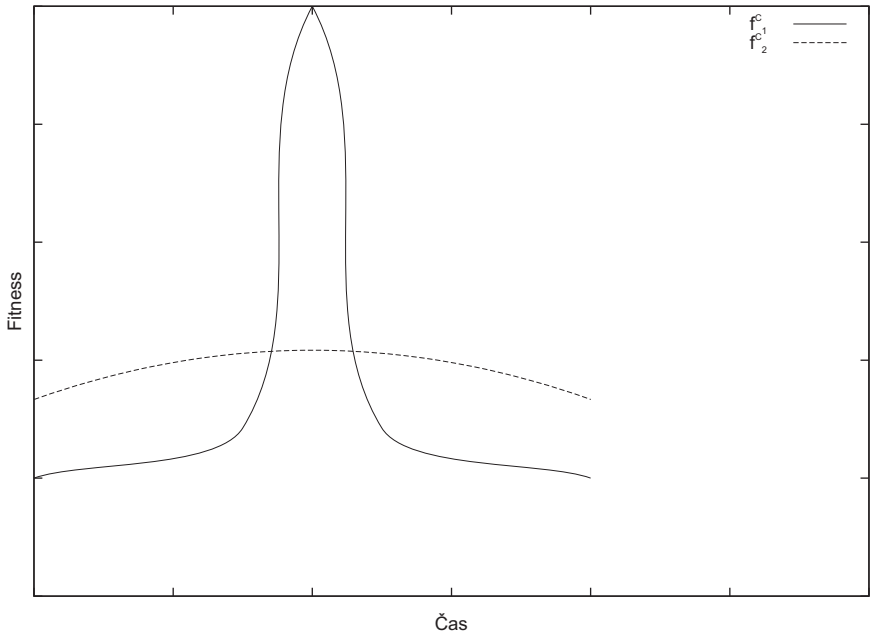
- F **je monotónní vzhledem k** T, tedy čím déle přesun trvá, tím vyšší je hodnota F
- F **je monotónní vzhledem k** f^C, tedy čím horší je fitness v jednotlivých okamžicích, tím vyšší je hodnota F

F však nedokáže dobře postihnout složitější případy:

- T **a** f^C **jdou proti sobě.** Je-li například $T_2 = 2{*}T_1$ a $f_2^C(2t) = f_1^C(t)/2$ pro $t \in [0;T]$ (viz 4.2a), pak se integrály rovnají, ale v závislosti na sémantice formace může být lepší kterákoliv z variant. Potom navíc může platit, že rychlejší cesta s vysokou f^C může být lepší varianta, než pomalejší varianta, která má dokonce nižší hodnotu F.
- **Tvar** f^C **se liší.** Podobně jako v předchozím případě není jasné, zda je lepší varianta s víceméně stálým průběhem f_1^C, nebo varianta, kde je f_2^C většinu času nižší, avšak obsahuje i výrazná lokální maxima (viz 4.2b).



(a) Poloviční délka průchodu s dvojnásobnou fitness



(b) Fitness s intervalem výrazně horší fitness

Obrázek 4.2: Různé tvary fitness se stejnou hodnotou F

Faktor času a průběhu funkce tedy nelze takto jednoduše kvantifikovat. Rozhodující je totiž určit, co je v dané situaci důležitější, zda délka přesunu, nebo jeho kvalita. To můžeme vystihnout vhodným škálováním fitness

$$F(k) = \int_0^T \left(f^C\right)^k \,\mathrm{d}t$$

pro parametr $k \in \mathbb{R}^+$. Hodnota $k < 1$ preferuje rychlejší přesuny, hodnota $k > 1$ naopak lépe hodnotí kvalitu přesunu.

Tento vzorec ve formě sumy použijeme pro odhad funkce F:

$$n \cdot \min t_i \cdot \left(\min f_i^C\right)^k \leq F(k) \leq n \cdot \max t_i \cdot \left(\max f_i^C\right)^k \ .$$

Funkci F lze z obou stran odhadnout exponenciální, pokud tedy F nahradíme vhodnou exponenciálou, budeme moci různé průběhy této funkce vhodně porovnat.

Parametry exponenciální funkce $y = Ae^{Bx}$ odhadneme metodou nejmenších čtverců [4]:

$$\begin{aligned} A &= \exp \frac{\sum\left(x_i^2 y_i\right)\sum\left(y_i \log y_i\right) - \sum\left(x_i y_i\right)\sum\left(x_i y_i \log y_i\right)}{\sum y_i \sum\left(x_i^2 y_i\right) - \left(\sum x_i y_i\right)^2}\,, \\ B &= \frac{\sum y_i \sum\left(x_i y_i \log y_i\right) - \sum\left(x_i y_i\right)\sum\left(y_i \log y_i\right)}{\sum y_i \sum\left(x_i^2 y_i\right) - \left(\sum x_i y_i\right)^2}\,. \end{aligned}$$

Pro odhad použijeme hodnoty funkce F v několika bodech z intervalu $\left[\frac{1}{10}, 10\right]$, konkrétně v bodech $x_i = 10^{\frac{i-10}{10}}$ pro $i \in \{0, 1, \ldots, 20\}$.

Průběh fitness tedy můžeme vhodně charakterizovat pomocí dvou parametrů:

1. $E_1 = Ae^B$ — hodnota v bodě 1, která určuje, jak je algoritmus dobrý bez dalších podmínek — čím nižší, tím lepší.
2. $\ddot{E} = e^B$ — základ exponenciály, který má opačnou sémantiku než parametr k — čím vyšší je hodnota, tím rychlejší přesuny algoritmus generuje.

Kapitola 5

Výsledky

Navrhli jsme několik modulů komplexního algoritmu, je třeba zhodnotit jejich funkčnost. Pro tento účel použijeme čtyři testovací formace (viz příloha A) dvou různých velikostí a šest testovacích scénářů (viz příloha B). Budeme zkoumat, jak se mění fitness v závislosti na zvoleném algoritmu, scénáři a formaci. Jednoduchým skriptem otestujeme všechny kombinace algoritmů a několik kombinací jejich parametrů. Celkově se jedná o vzorek 6016 konfigurací. Následně pomocí statistického programu `R`[9] analyzujeme hodnoty E_1 a $\ddot{E}$ fitness běhu těchto konfigurací.

5.1 Detekce kolizí

Začněme s algoritmem detekce kolizí a porovnejme výsledky základního algoritmu (tedy algoritmu, který žádné kolize neřeší) a algoritmu změny fyzického umístění (viz 3.4.1). Z tabulky 5.1a vidíme, že kolizní algoritmus má mírně negativní vliv na kvalitu přesunu. Zřejmě je to způsobeno tím, že algoritmus detekce kolizí řeší pouze lokální chyby, a nijak nepomáhá držet tvar formace. Fitness pak nezohledňuje, kolik kolizí při běhu nastalo, takže pokud jsou kolize řešeny lokálním vychýlením jednotky z optimální trasy, je pochopitelné, že fitness musí vzrůst.

V tabulce 5.1b (resp. 5.1c) máme výběrový rozptyl mediánů E_1 (a $\ddot{E}$) fitness běhů s různým typem detekce kolizí na daném scénáři (resp. s daným schématem formace). Vysoký rozptyl značí, že použitá metoda má velký vliv na výkon formace na daném scénáři (resp. s daným schématem formace). Nízký rozptyl značí, že pro danou situaci nehraje modul příliš velkou roli.

Z 5.1b je vidět, že ve scénáři *Cvičiště* nemá detekce kolizí téměř žádný vliv (rozptyl hodnoty E_1 je velmi nízký). Naopak velkou roli hraje detekce kolizí ve scénářích *Zatáčka* a *Soutěska*, kde zřejmě nastává větší množství kolizí.

Tabulka 5.1c nám prozrazuje, že vliv detekce kolizí na většinu schémat je podobný. Vliv formace klesá s její velikostí - čím totiž formace obsahuje více jednotek, tím větší prostor zaplňuje, čímž snižuje množství kolizí, a tedy jejich vliv. Výrazně vybočuje pouze schéma čára. Nízký počet kolizí v prvním případě je způsoben zejména nízkým počtem jednotek. Naopak při vyšším počtu jednotek formace neudržuje dostatečnou soudržnost kvůli nedostatečnému počtu vazeb ve schématu, což v náročnějším terénu může zvyšovat počet kolizí.

Kolizní algoritmus	median(E_1)($\cdot 10^3$)	median($\ddot{E}$)($\cdot 10^6$)
žádný	11.796	12.84
změna fyzického umístění	14.136	12.10

(a) Vliv detekce kolizí

Scénář	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Cvičiště	0.00002	0.0001
Otočka	0.0003	6.067
Zatáčka	5.734	2.679
Soutěska	2.151	1.106
Bažina	1.151	1.355
Cesta	0.916	0.340

(b) Vliv scénáře na detekci kolizí

Formace	Počet jednotek	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Čára	3	0.00015	0.0002
Čtverec	4	0.1207	0.182
Diamant	4	3.816	1.303
Klín	5	3.075	0.259
Čára	7	9.003	0.231
Čtverec	8	0.0796	0.0004
Diamant	9	0.3123	0.196
Klín	9	1.010	0.00001

(c) Vliv formace na detekci kolizí

Tabulka 5.1: Vliv detekce kolizí na fitness

5.2 Plánování

Plánovací algoritmus, periodické přeplánování (viz 3.2.1), má parametry F a P, které ovlivňují jeho vlastnosti. Chování algoritmu analyzujeme pro tři ro-

zumně nízké hodnoty těchto parametrů. Z tabulky 5.2a vyplývá, že menší hodnoty generují kvalitnější přesuny. U scénáře *Otočka* není tento rozdíl příliš patrný.

Druhá metoda, fragmentace po zatáčkách (viz 3.2.2) se svým parametrem φ_0, která vznikla jako pokus o vylepšení fragmentace v prvním algoritmu, dosahuje nejlepších výsledků s hodnotou $\varphi_0 = 10°$. Ostatní hodnoty se v různých scénářích chovají odlišně. Ve scénářích *Cvičiště* a *Cesta* hodnota $\varphi_0 = 40°$ zaostává za ostatními, ve scénáři *Bažina* je naopak nejlepší. Obecně není možné tyto hodnoty kvalitativně odlišit. Co se týče formací je situace jasnější — menší hodnota φ_0 znamená o něco lepší vlastnosti. U schématu čtverec je tento vliv největší.

Plánovací algoritmus	median(E_1)($\cdot 10^3$)	median($\ddot{E}$)($\cdot 10^6$)
Triviální	21.300	39.710
Periodické přeplánování ($F = 3$, $P = 2$)	5.396	5.236
Periodické přeplánování ($F = 5$, $P = 3$)	8.665	7.666
Periodické přeplánování ($F = 10$, $P = 5$)	16.271	19.68
Fragmentace po zatáčkách ($\varphi_0 = 10°$)	10.952	9.403
Fragmentace po zatáčkách ($\varphi_0 = 20°$)	13.234	11.48
Fragmentace po zatáčkách ($\varphi_0 = 40°$)	11.893	11.48
Fragmentace po zatáčkách ($\varphi_0 = 50°$)	13.126	11.78

(a) Vliv plánování na fitness

Scénář	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Cvičiště	4.72	107.7
Otočka	27.05	65.94
Zatáčka	25.95	45.85
Soutěska	14.18	502.8
Bažina	173.83	8863
Cesta	12.56	147.7

(b) Vliv scénáře na plánování

Formace	Počet jednotek	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Čára	3	0.638	0.9679
Čtverec	4	56.91	161.8
Diamant	4	55.82	114.1
Klín	5	23.83	56.20
Čára	7	22.08	31.44
Čtverec	8	43.47	174.4
Diamant	9	18.58	200.9
Klín	9	22.70	117.3

(c) Vliv formace na plánování

Tabulka 5.2: Vliv plánování na fitness

Překvapivě se jako lepší z obou metod jeví metoda jednodušší, a to periodické přeplánování. Obě metody jsou však výrazně lepší než triviální algoritmus.

5.3 Synchronizace příchodu

Různé metody synchronizace příchodu se chovají velmi podobně (viz 5.3a). Pohyb formací s adaptivní synchronizací se jeví opticky nejpřirozenější, avšak data tuto hypotézu nepodporují. Ukazuje se, že ve složitějších scénářích (*Zatáčka*, *Soutěska*, *Bažina*, *Cesta*) hraje synchronizace větší roli, v jednodušších není tak důležitá. Velké rozdíly jsou mezi jednotlivými formacemi — čtverec a diamant dávají bez synchronizace mnohem horší výsledky než s ní. U schématu čára není naopak synchronizace příliš potřeba. To se dalo očekávat, vzhledem k tomu, že tato synchronizace byla motivována problémem, který je znázorněný na obrázku 2.2. Tento problém totiž u formace čára, která má v podstatě nulovou šířku, nenastává.

Synchronizační algoritmus	median(E_1)($\cdot 10^3$)	median($\ddot{E}$)($\cdot 10^6$)
žádný	16.354	24.82
adaptivní	11.678	10.26
aditivní	11.336	10.74
multiplikativní	11.278	10.53

(a) Vliv synchronizace příchodu na fitness

Scénář	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Cvičiště	0.016	0.0496
Otočka	0.00005	46.40
Zatáčka	5.722	37.00
Soutěska	19.757	1167
Bažina	10.348	408.8
Cesta	1.464	77.76

(b) Vliv scénáře na synchronizaci

Formace	Počet jednotek	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Čára	3	0.002	0.0121
Čtverec	4	5.407	284.0
Diamant	4	30.688	24.87
Klín	5	1.038	22.34
Čára	7	1.234	0.3008
Čtverec	8	18.776	355.6
Diamant	9	10.246	190.5
Klín	9	3.126	47.64

(c) Vliv formace na synchronizaci

Tabulka 5.3: Vliv synchronizace příchodu na fitness

5.4 Vyhledávání cesty

Posledním modulem, jehož vlastnosti zbývá prozkoumat, je modul odpovědný za vyhledávání cesty. Ukazuje se, že metoda kombinovaného prohledávacího prostoru (viz 3.1.1) zlepšuje výkon formace ve většině scénářů. U scénáře *Cesta* není zlepšení průkazné, někdy si dokonce kombinovaný prohledávací prostor vede i hůře. To je zřejmě způsobeno větším množstvím zatáček v cestě, se kterými má algoritmus potíže. Scénáře *Cvičiště* a *Otočka* by neměly být ovlivněny vůbec, jelikož obsahují pouze hrany stejné ceny. Stává se však, že kvůli zaokrouhlovacím chybám při výpočtu ceny hran v kombinovaném prohledávacím prostoru jsou některé hrany vyhodnoceny jako mírně kratší, což způsobuje výběr jiné cesty v případě, že do cílové pozice vede více stejně dobrých cest.

Prohledávací algoritmus	median(E_1)($\cdot 10^3$)	median($\ddot{E}$)($\cdot 10^6$)
jednoduché	13.930	14.05
kombinované	12.007	11.11

(a) Vliv vyhledávání cesty na fitness

Scénář	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Cvičiště	0.266	0.2980
Otočka	0.00006	0.0090
Zatáčka	4.852	10.13
Soutěska	1.006	0.1067
Bažina	0.571	10.27
Cesta	0.293	13.22

(b) Vliv scénáře na vyhledávání cesty

Formace	Počet jednotek	var(E_1)($\cdot 10^6$)	var($\ddot{E}$)($\cdot 10^{12}$)
Čára	3	0.157	0.0587
Čtverec	4	1.241	42.15
Diamant	4	5.080	0.0305
Klín	5	0.212	7.627
Čára	7	0.774	0.3688
Čtverec	8	4.906	32.54
Diamant	9	0.916	2.202
Klín	9	0.667	17.06

(c) Vliv formace na vyhledávání cesty

Tabulka 5.4: Vliv vyhledávání cesty na fitness

Podobně jako u synchronizace příchodu, složitější vyhledávací algoritmus je přínosný zejména pro široké formace, jako je čtverec a diamant.

Závěr

Zadaný problém se nám podařilo rozdělit na čtyři nezávislé podproblémy — vyhledávání cesty, plánování pohybu, synchronizace příchodu a detekce kolizí. U každého z nich jsme ukázali, v jakých situacích hraje klíčovou roli. Pro každou z těchto úloh jsme navrhli několik metod jejího řešení. Tyto metody jsme posléze analyzovali.

Ukázalo se, že synchronizace příchodu, bez ohledu na konkrétní způsob, kterým je prováděna, hraje důležitou a pozitivní roli. Co se týče plánování, zdá se, že jednodušší přístup k fragmentaci cesty na úseky funguje lépe než sofistikovanější přístup detekce přímých úseků. Oba způsoby jsou ale výrazným vylepšením oproti triviálnímu algoritmu. Kombinovaný prohledávací prostor hraje spíše pomocnou roli. Je možné, že by pomohla lepší integrace s ostatními moduly, v opačném případě by se nejspíše ukázalo, že daná metoda není příliš vhodná a k celé věci by bylo třeba najít jiný, lepší přístup. Metody detekce kolizí řeší pouze lokální problémy a k celkové situaci ve formaci výrazně nepřispívají.

Všechny přístupy mají společné to, v jakých situacích jsou prospěšné. Ukazuje se, že formace, které mají větší šířku, jsou schopny více využít výhod, které tyto přístupy poskytují. Metody také nejlépe fungují ve středně obtížných scénářích, ve složitějších již jejich přínos klesá.

Některé navržené postupy využívají některých vlastností dvourozměrného prostoru, nicméně až na metody detekce přímých úseků by mělo být možné je přizpůsobit pro 3D-světy. Je však pravděpodobné, že by tyto moduly narazily na určitá omezení projekce dvourozměrné formace a nebyly by schopny se plně přizpůsobit nerovnostem trojrozměrného terénu.

Modulární architektura poskytuje prostor pro další inovaci, a to jak z hlediska kvality řešení známých situací, tak možností přidání nových funkcí a schopností jednotek. Pokročilé metody synchronizace příchodu by například mohly měnit průběh cesty rychlých jednotek, a to tak, aby tyto během pohybu cíleně vylepšovaly svoji bezpečnost a přínos formaci navštěvováním bodů, které dočasně zvýší jejich krytí.

Použité značení

`angleTo`$(\mathbf{d}, \mathbf{e})$	úhel, který svírají směry d a e ($[0°, 360°)$)
`angleBetween`$(\mathbf{d}, \mathbf{e})$	úhel mezi směry d a e ($[0°, 180°]$)
B (*binginds*)	množina vazeb ve formaci
B_s	množina vazeb na místo s
$B_s^{\rightarrow}$	množina vazeb z místa s
c_{min}	minimální možná cena hrany v mapě
$c_{p,q}$	cena hrany z vrcholu p do vrcholu q
$c^c_{p,q}$	cena hrany z vrcholu p do vrcholu q v kombinovaném prohledávacím prostoru
$\mathbf{d}(b)$ (*distance*)	okamžitá vzdálenost mezi místy ve vazbě b
E_1	nulová hodnota proložené exponenciály $F(k)$
$\ddot{E}$	základ proložené exponenciály $F(k)$
F	úsek cesty (fragment)
$F(k)$	škálovaná fitness
$f(p, q)$	cena nejkratší cesty z vrcholu p do vrcholu q
f^A	úhlová fitness
f^D	fitness vzdálenosti
f^P	fitness aritmetického průměru
$h(p, q)$	heuristický odhad ceny nejkratší cesty z vrcholu p do vrcholu q
$\mathbf{o}(b)$ (*offset*)	optimální vzdálenost mezi místy ve vazbě b
P	cesta (posloupnost vrcholů)
S (*spots*)	množina míst ve formaci
$v_1, v_2, \ldots, v_n$	vrcholy na mapě

Seznam tabulek

Seznam obrázků

Seznam ukázek kódu

Literatura

[1] Rabin S.: *AI Game Programming Wisdom*, Cengage Learning, 2002

[2] *gnuplot*, http://www.gnuplot.info/

[3] Delbrück H.: *History of the Art of War*, University of Nebraska Press, 1990

[4] Weisstein, Eric W.: *Least Squares Fitting–Exponential*, MathWorld, Wolfram, http://mathworld.wolfram.com/LeastSquaresFittingExponential.html

[5] Cormen T. H., Leiserson C. E., Rivest R. L.: *Introduction to Algorithms*, MIT Press, McGraw-Hill, 2009, kap. 10

[6] *MinGW — Minimalistic GNU for Windows*, http://www.mingw.org/

[7] Clausewitz, Carl Von: On War, Oxford University Press, 2007

[8] *Qt — Cross-platform application and UI framework*, http://qt.nokia.com

[9] *R Project for Statistical Computing*, http://www.r-project.org/

[10] Lendon, J.E.: *Soldiers & Ghosts: A History of Battle in Classical Antiquity*, Yale University Press, 2005

Příloha A

Schémata formací

Pro testování algoritmů budeme používat několik různých schémat formací. Tato schémata definujeme nejprve obecně, a poté ukážeme jejich instance pro konkrétní počet jednotek.

A.1 Řada

Nejjednodušší schéma, které budeme používat, je řada, která má podobu prostého zástupu jednotek, viz obrázek A.1.

(a) Velikost 3

(b) Velikost 7

Obrázek A.1: Schéma formace řada

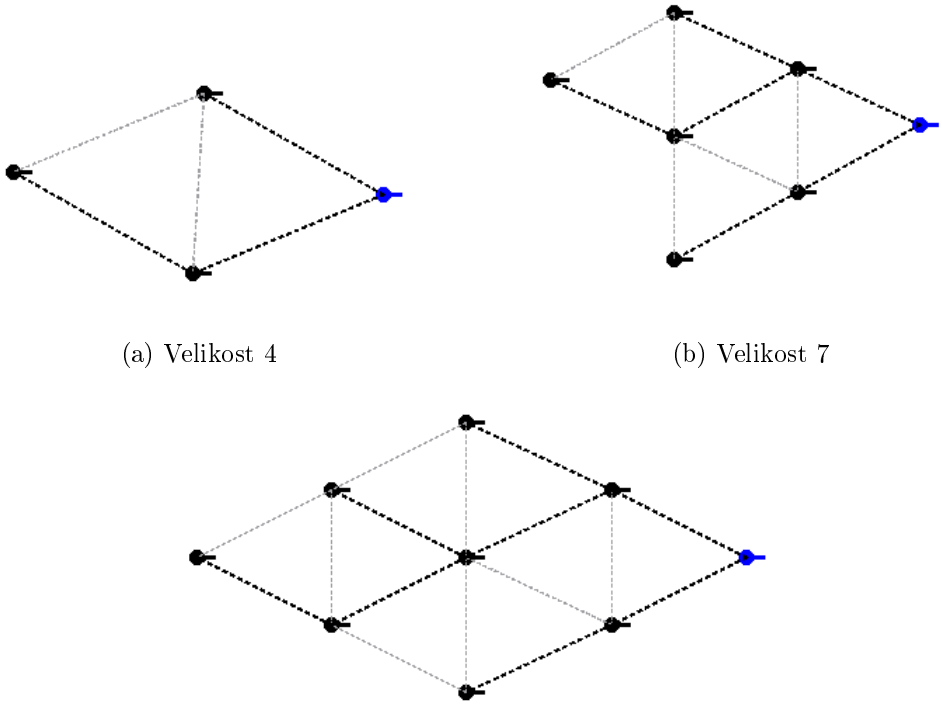
A.2 Diamant

Formace diamant je vrstevnaté schéma, tedy schéma, které se sestává z několika vrstev bodů, přičemž vazby se vyskytují pouze v rámci jedné vrstvy, nebo mezi body v sousedních vrstvách. Diamant typu n se skládá z $2n - 1$ vrstev, přičemž počet jednotek ve vrstvě je definován předpisem:

$$\text{units}(i) = \begin{cases} i & \text{pro } i \leq n \\ 2n - i & \text{pro } i > n \end{cases}$$

Vidíme, že v první vrstvě je jediná jednotka, která je zároveň velitelem formace. Velikost dvou sousedních vrstev se liší vždy právě o jedna.

Takto definovanou formaci můžeme zkonstruovat pouze pro n, které je druhou mocninou přirozeného čísla, jak vidíme na obrázcích A.2a a A.2c. Pro jiné počty konstrukci provedeme tak, že vezme nejbližší vyšší mocninu, a ze schématu odebereme tolik bodů, aby měla formace požadovanou velikost. Body přitom odebíráme odzadu. Příklad takového schématu je na obrázku A.2b. Tato hybridní schémata však nebudeme při testování brát v úvahu.



(a) Velikost 4

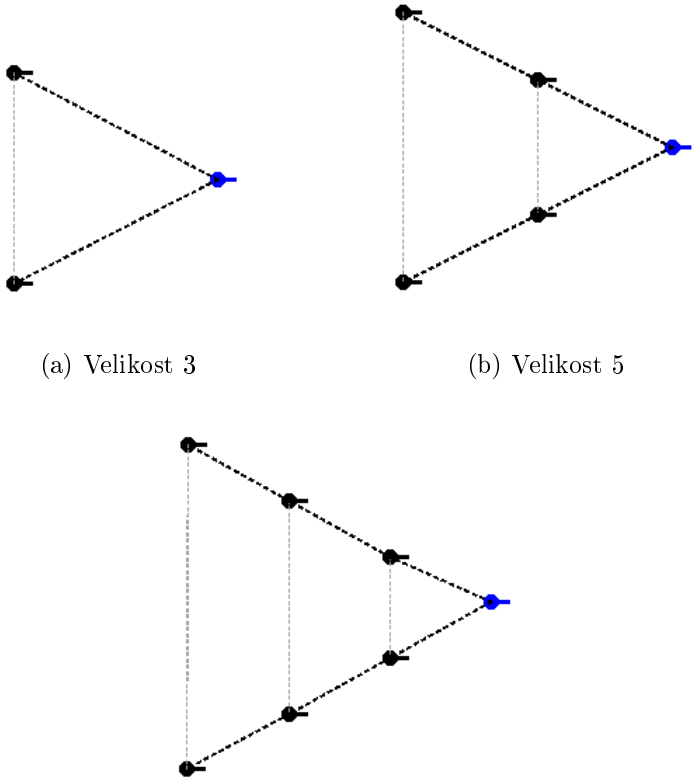
(b) Velikost 7

(c) Velikost 9

Obrázek A.2: Schéma formace diamant

A.3 Klín

Druhou vrstevnatou formací, kterou použijeme, je klín. Klín má libovolný počet vrstev, kde první vrstva obsahuje toliko velitele formace, a každá další obsahuje právě dva body, a je mírně větší, než předchozí. Příklad takové formace je na obrázku A.3. Stejně jako v případě formace diamant, i zde můžeme definovat hybridní formace, u kterých není poslední vrstva zaplněna, ale ani zde je nebudeme brát v úvahu.



(a) Velikost 3

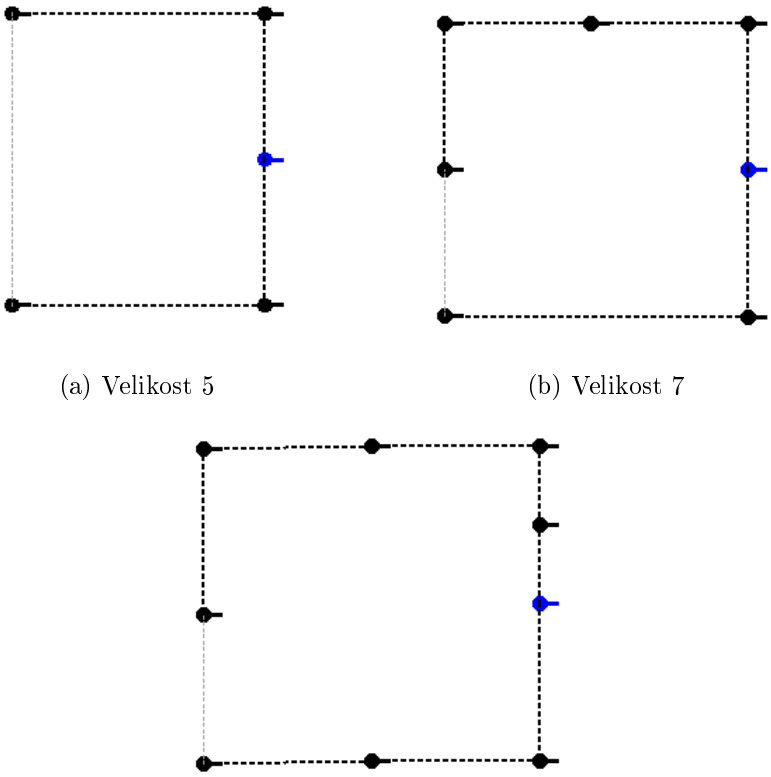
(b) Velikost 5

(c) Velikost 7

Obrázek A.3: Schéma formace klín

A.4 Čtverec

Schéma čtverec jako jediné není vrstevnaté. Čtverec, jak již název napovídá, má tvar čtverce, v jehož každém rohu se nachází bod. Velitel je umístěn uprostřed přední hrany čtverce. Další jednotky se umísťují rovnoměrně na všechny 4 strany.



(a) Velikost 5

(b) Velikost 7

(c) Velikost 9

Obrázek A.4: Schéma formace čtverec

Příloha B

Scénáře

Pro testování algoritmů máme k dispozici několik modelových scénářů. Scénáře jsou založeny na určité typické situaci.

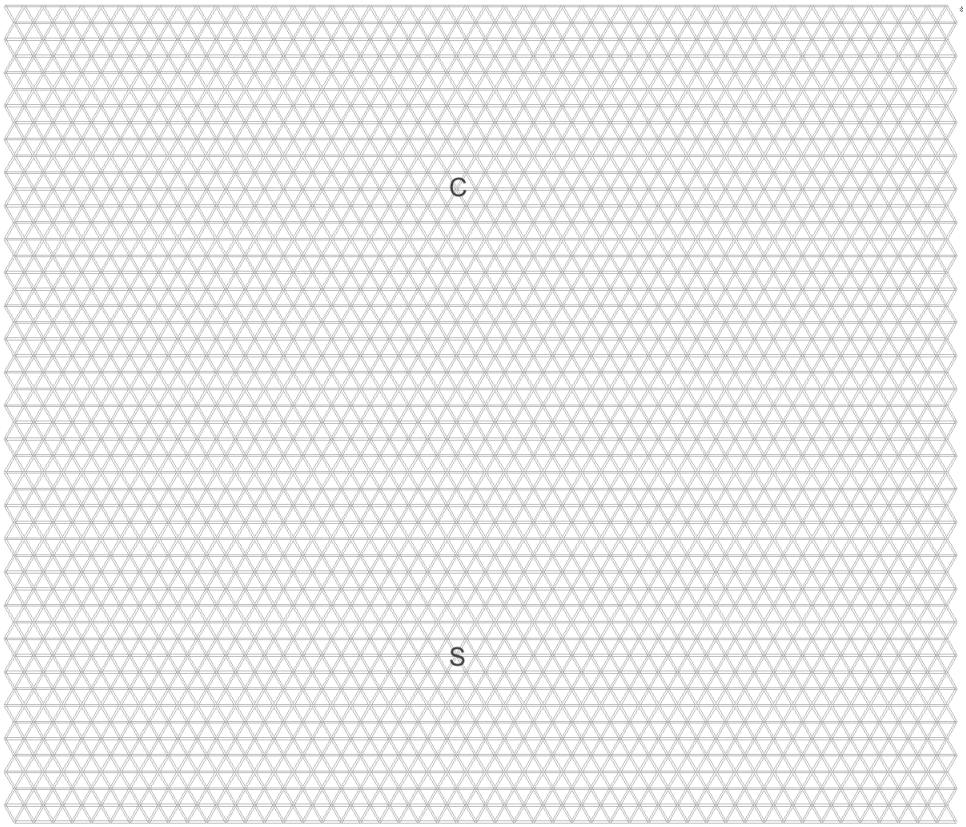
U každého scénáře je uveden obrázek mapy, který obsahuje hrany různých stupňů šedi. Hrany s nejmenší cenou jsou vyznačeny černou barvou, hrany s maximální cenou bílou. Na mapě jsou dále vyznačeny pozice počátečního a cílového bodu. Jako poslední údaj je uveden název souboru, ve kterém je scénář uložen. Soubory scénářů jsou na přiloženém DVD ve složce `scenarios`.

Typ formace, její velikost, ani použité moduly nejsou v tomto případě uvedeny, testují se totiž různé kombinace uvedených prvků nezávisle na scénáři.

B.1 Cvičiště

První jednoduchý scénář testuje, jak se formace pohybuje, pokud na ní nejsou kladeny žádné další požadavky. Mezi výchozím a cílovým bodem existuje přímá cesta bez jakýchkoliv zatáček a nerovností. Terén na mapě je tedy všude stejně náročný.

Velikost:	50×50
Start:	$(47, 39)$
Počáteční azimut:	$0°$
Cíl:	$(47, 11)$
Soubor scénáře:	`1-Simple.sce`

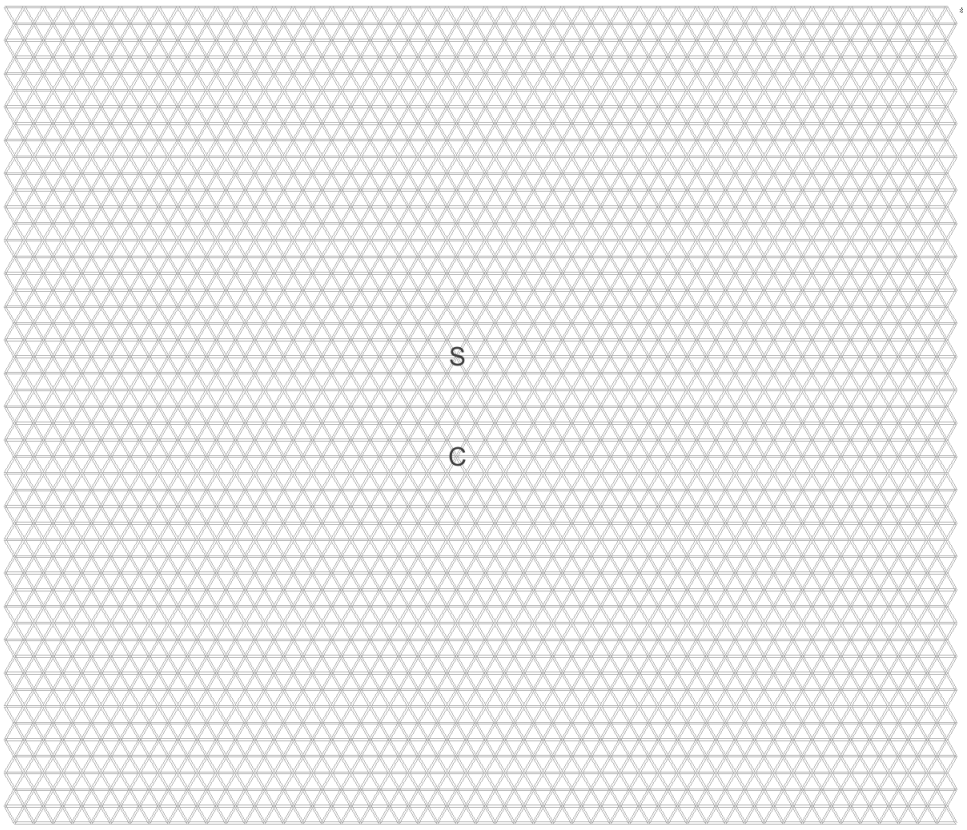


Obrázek B.1: Schéma scénáře *Cvičiště*

B.2 Otočka

Scénář *Otočka* testuje schopnost splnění obvyklého typu rozkazu - otočení formace na místě o 180°. Jednotky by tento úkol měly splnit jednoduchým způsobem, a to průchodem na cílové místo nejkratší cestou. Ve většině případů průchod prochází středem formace. V okolí tohoto bodu vzniká velké množství kolizí, které je třeba řešit. Pro jednoduchost je terén na mapě všude stejně náročný.

Velikost:	50×50
Start:	$(47, 21)$
Počáteční azimut:	0°
Cíl:	$(47, 27)$
Soubor scénáře:	`2-Turnaround.sce`

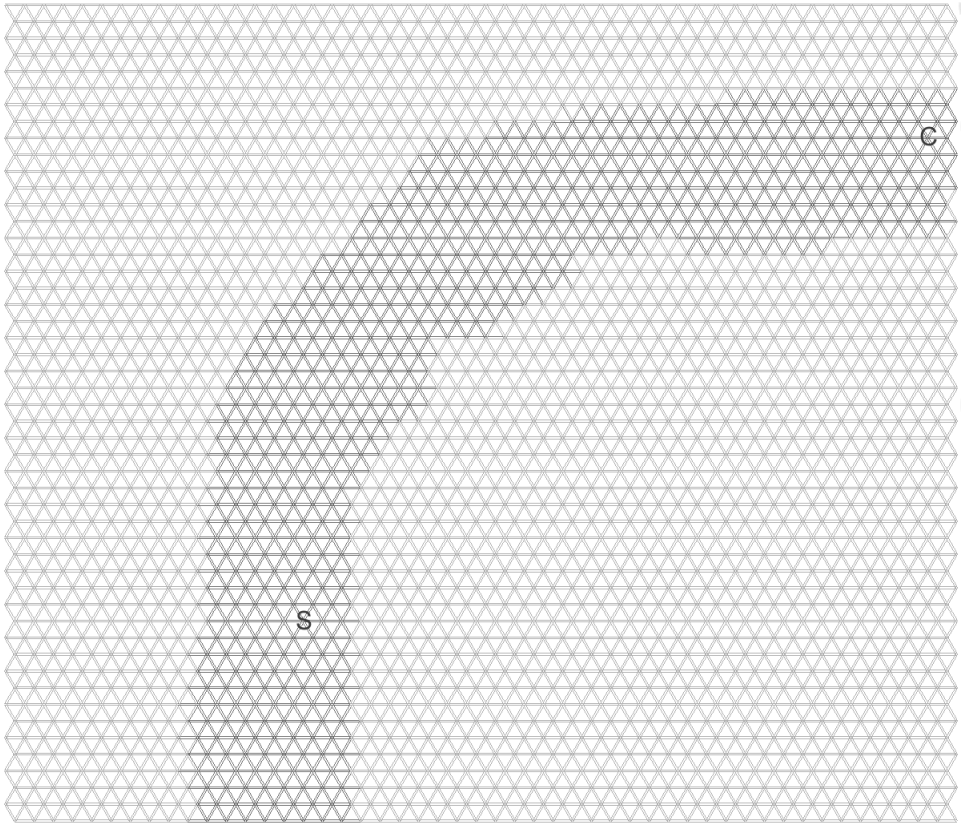


Obrázek B.2: Schéma scénáře *Otočka*

B.3 Zatáčka

Scénář *Zatáčka* demonstruje schopnosti formace přesunovat se po cestě, která zatáčí. V kapitole 2.3, kde je tato situace rozebrána, je odhaleno několik potenciálních problémů, kterým je třeba čelit. Tento scénář testuje úspěšnost zvoleného řešení.

Velikost:	50×50
Start:	$(29, 37)$
Počáteční azimut:	$0°$
Cíl:	$(96, 8)$
Soubor scénáře:	`3-Curve.sce`

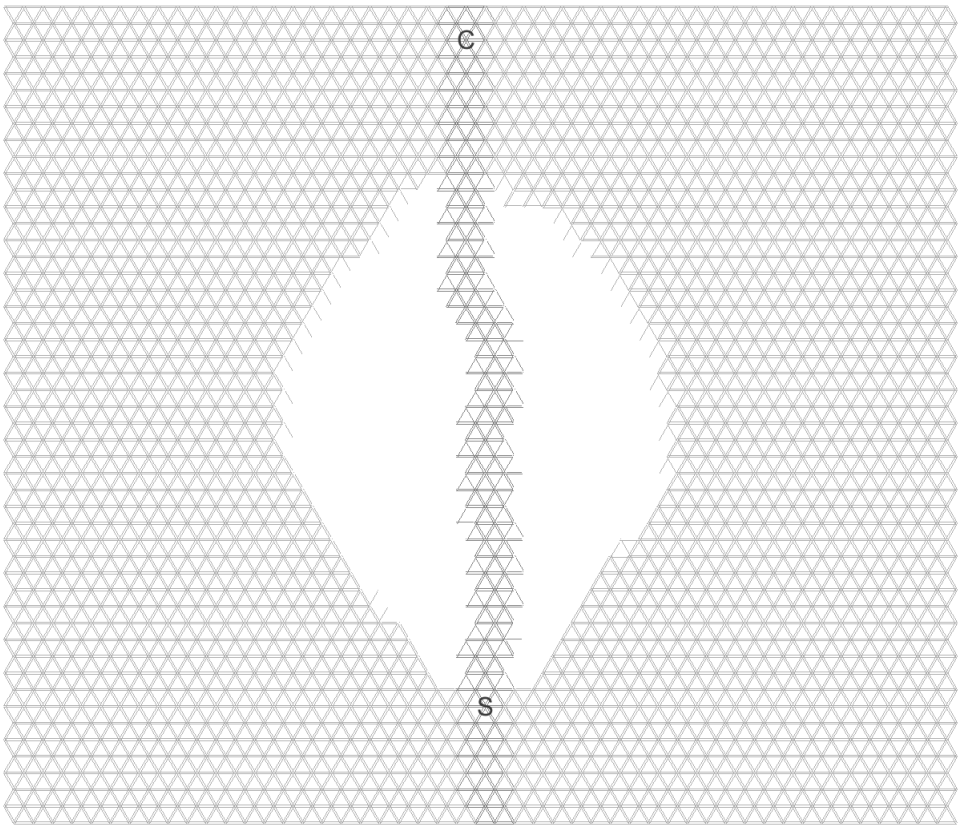


Obrázek B.3: Schéma scénáře *Zatáčka*

B.4 Soutěska

Scénář *Soutěska* představuje situaci, která byla popsána na obrázku 2.3, tedy problém příliš úzké cesty na to, aby jí formace byla schopná projít. Aby nebylo řešení korelované s řešením jiných problémů (například průchod formace zatáčkou), jsou průchod soutěskou i alternativní cesta co nejpřímější, a tedy nejjednodušší.

Velikost:	50×60
Start:	$(50, 42)$
Počáteční azimut:	$0°$
Cíl:	$(48, 2)$
Soubor scénáře:	`4-Canyon.sce`

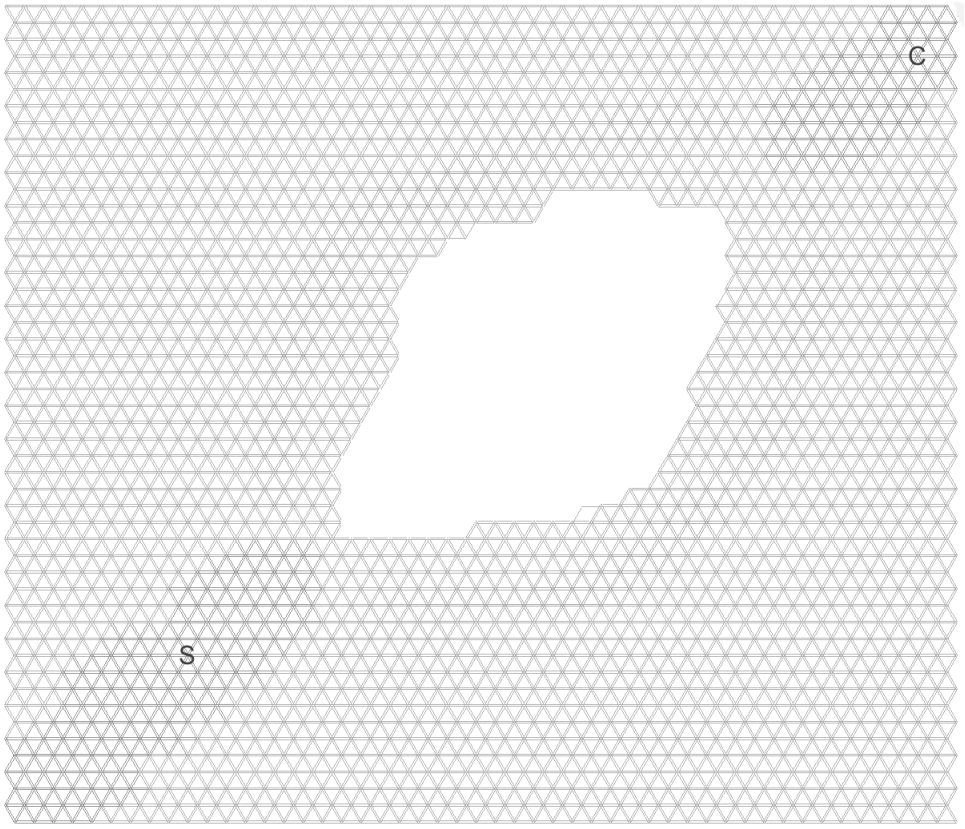


Obrázek B.4: Schéma scénáře *Soutěska*

B.5 Bažina

Scénář *Bažina* testuje řešení příbuzného problému, a to volbu společné cesty v případě velké překážky. Triviální algoritmus v tomto scénáři generuje cesty, při kterých různé jednotky obcházejí překážku z různých stran.

Velikost:	50×50
Start:	$(19, 39)$
Počáteční azimut:	$45°$
Cíl:	$(95, 3)$
Soubor scénáře:	`5-Swamp.sce`

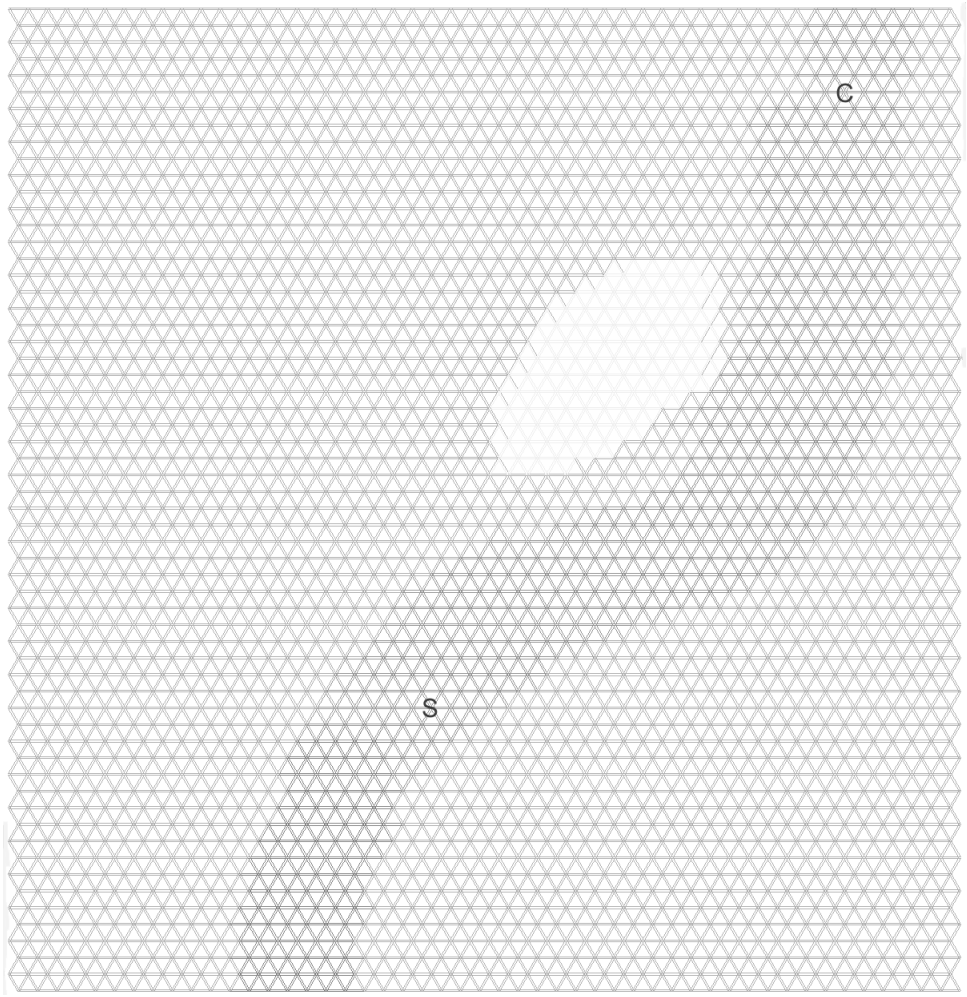


Obrázek B.5: Schéma scénáře *Bažina*

B.6 Cesta

Scénář Cesta představuje složitější typ scénáře, ve kterém je několik různých prvků. Může se na něm tedy ukázat, jak jednotlivé podsystémy spolupracují, a zda je schopna formace udržet tvar i poté, co několikrát změní směr.

Velikost:	50×60
Start:	$(44, 42)$
Počáteční azimut:	$40°$
Cíl:	$(87, 5)$
Soubor scénáře:	`6-Comples.sce`



Obrázek B.6: Schéma scénáře *Cesta*

Příloha C

Uživatelská dokumentace

C.1 Úvod

Program `adaptivni-formace` je aplikace, která implementuje techniky navrhované v textu a poskytuje nástroje pro jejich testování a analýzu. Aplikace je napsána v programovacím jazyku `C++` za pomoci knihovny Qt[8], pro tvorbu grafů je použit program gnuplot[2]. Aplikace je k dispozici na DVD přiloženém k práci.

C.2 Instalace

C.2.1 Instalace ze zdrojových kódů

Pro instalaci je potřebná knihovna Qt verze alespoň 4.6.0. Její zdrojové kódy je možné sehnat na `http://qt.nokia.com`. Druhým způsobem je instalace předkompilovaných balíčků, jedná se o `libqt4-core`, `libqt4-gui`, `libqt4-xml`, `libqt4-dev` a jejich závislosti. Pro systém Windows lze získat knihovnu zkompilovanou pomocí MinGW[6] přímo z `http://qt.nokia.com`. Zdrojové kódy aplikace jsou na přiloženém DVD v adresáři `src`, soubory zdrojů v adresáři `res`.

V kořenovém adresáři pak spustíme `qmake adaptivni-formace.pro` pro vygenerování souborů `Makefile`[1]. Kompilaci a sestavení pak provedeme spuštěním příkazu `make`[2]. Aplikace bude zkompilována ve složce `release`[3].

C.2.2 Instalace předkompilovaného balíčku

Předkompilovaný balíček pro Windows XP a vyšší se nachází ve složce `bin`. Složka obsahuje všechny knihovny potřebné ke spuštění aplikace.

[1] Přirozeně je třeba data nejprve nakopírovat na zapisovatelné médium

[2] Příkaz může mít jiný název podle typu kompilátoru, např. `mingw32-make` pro *MinGW* nebo `nmake` pro kompilátor *Microsoft Visual Studia*.

[3] Případně `debug` podle typu sestavení.

C.3 Nastavení

Nastavení různých vlastností programu ovlivňuje soubor `ini/settings.ini`. Soubor je ve formátu INI file.

C.3.1 Gnuplot

Pro tvorbu grafů potřebuje aplikace program gnuplot. Cestu k němu udává položka nastavení `tools/gnuplot`. Nebude-li cesta správná, nebudou se žádné grafy vytvářet.

C.3.2 Soubory scénářů

Scénáře popsané v příloze B jsou uloženy ve složce `scenarios`. Hodnota nastavení `directories/scenarios` udává, ve které složce program scénáře hledá a kam je ukládá. Toto nastavení je užitečné zejména pro ovládání programu z příkazové řádky.

C.3.3 Složka logů

Logovací soubory se ve výchozím nastavení ukládají do složky `log`, což je možné změnit pomocí položky nastavení `directories/logs`. Formát souborů logu je popsán v kapitole D.5. Je důležité, aby měl program práva k zápisu do této složky.

C.3.4 Fitness

Parametry ve složce fitness ovlivňují jednak chování funkce Optimalizace fitness (viz strana 45), dále nastavují použitou kombinovanou fitness.

C.3.5 Moduly

Dalším možným nastavením v INI souboru je náhrada nastavení modulů z otevřeného scénáře pomocí hodnot v tomto souboru. To je možné provést pomocí hodnot `pathFinding`, `speedSynchronizing`, `planning`, `executing` a `positioning`. Hodnota 0 znamená, že se použije nastavení ze scénáře, ostatní hodnoty se interpretují jako konstanty, definované v souboru `src/modules/moduleprovider.h`.

Zbylé hodnoty představují další parametry jednotlivých modulů.

C.3.6 Formace

Stejně jako v případě modulů, i nastavení formace je možné ovlivnit v INI souboru. Odpovídající hodnoty jsou v sekci `formation`.

C.4 Spuštění

Při spuštění programu je možné zadat několik parametrů na příkazové řádce:

- `-h` nebo `--help` — vytiskne nápovědu k příkazové řádce a ukončí program.
- `-s` nebo `--nosplash` — zakáže uvítací obrazovku (vhodné při automatickém spouštění či analýze scénářů (viz dále)).
- `-r` nebo `--run` — spustí scénáře, jejichž názvy jsou uvedeny jako parametry na příkazové řádce, a uloží informace o jejich průběhu do logu. Scénáře jsou spuštěny bez grafického rozhraní, celý proces je tedy výrazně rychlejší.
- `-a` nebo `--analyze` — provede analýzu fitness funkce na zadaném scénáři.
- `--i`*`t key=value`* — překrytí hodnoty z INI souboru hodnotou na příkazové řádce:
  - *t* představuje označení typu proměnné (`i` — celočíselná hodnota, `d` — číslo s plovoucí desetinnou čárkou, `s` — řetězec).
  - *key* představuje celou cestu ke klíči (včetně sekce).
  - *value* je hodnota.

Pomocí parametrů `--i` a `-r` byl proveden sběr vzorků pro kapitolu výsledky[4].

C.5 Ovládání

Program je SDI[5] pro soubory scénářů. Scénář může být otevřen v jednom ze tří módů. Všechny módy mají společné základní ovládání mapy. Držením levého tlačítka myši a tažením se lze po mapě posunovat. Na přibližování a oddalování mapy použijeme kolečko myši.

[4] Skript pro sběr dat naleznete v souboru `bin/run.bat`.

[5] *Single Document Interface* — otevřen může být v jednom okamžiku nejvíce jeden soubor.

C.5.1 Editace

V módu editace (*Edit*) máme k dispozici několik nástrojů na tvorbu scénářů. Ceny hran můžeme měnit pomocí několika štětců. V dokovacím okně *Drawing Tools* lze zvolit tvar a velikost štětce. Dále máme na výběr ze tří operací, které bude štětec na hranách provádět — přičtení konstanty, násobení konstantou a nastavení ceny na konstantu. Aplikace štětce na mapu probíhá pravým tlačítkem myši.

V dokovacím okně *Formation* nastavujeme vlastnosti formace - tedy její typ, počet jednotek a startovní orientaci. Po stisknutí tlačítka *Start* (resp. *Target*) pravým tlačítkem vybereme na mapě počáteční (resp. cílovou) pozici formace. Dále zde můžeme nastavit moduly, které se použijí v algoritmu pro přesun formace.

C.5.2 Spuštění

V módu spuštění (*Run*) je na mapě inicializována formace, která dostane rozkaz k přesunu na cílové místo. Formaci lze též pravým tlačítkem dávat jiné rozkazy. Držením `Shift` při stisknutí pravého tlačítka je možné sestavit cestu s několika průběžnými body cesty (*waypointy*).

V menu *Flow* je možné simulaci zrychlovat, zpomalovat, či pozastavit.

C.5.3 Fitness

V módu fitness (*Fitness*) máme přístup k několika dalším funkcím.

Máme k dispozici inicializovanou formaci, u které můžeme ručně přesouvat libovolné jednotky a sledovat přitom, jak se mění statická fitness formace. Při výběru libovolné jednotky se mapa zabarví podle toho, jak by se změnila fitness při přesunu vybrané jednotky na zadanou pozici — přičemž místa s nejhorší fitness jsou vyznačena červeně, lepší žlutě a poté zeleně, a oblast s nejnižší hodnotou fitness modře.

Příkaz *Randomize fitness* umístí jednotky formace na náhodná místa na mapě.

Příkaz *Optimize fitness* provede jednu iteraci funkce optimalizace fitness, která je popsána na straně 45.

Příkaz *Analyze fitness* spustí experiment analýzy fitness, který je taktéž popsán na straně 45.

Příloha D

Programátorská dokumentace

D.1 Mapa

Interface mapy definuje třída `AbstractAreaGraph`. Mapa poskytuje přístup k vrcholům a hranám. Existují dvě různé implementace mapy. První (`AreaGraph`) reprezentuje fyzickou mapu, tak jak je popsána v kapitole 1.1. Druhá varianta (`CombinedAreaGraph`) konstruuje z několika dalších map kombinovaný prohledávací prostor (viz 3.1.1).

Pro reprezentaci vrcholů se používá třída `Node`. Vrchol je reprezentován odkazem na mapu a souřadnicemi na ní.

Třída `Edge` představuje hrany na mapě. Hrana se vždy skládá z vrcholu a jednoho ze šesti směrů (`Direction`).

Pro složitější operace týkající se směru je k dispozici třída `Azimut`, která má interface pro výpočet hodnot funkcí `angleTo` a `angleBetween`.

D.2 Algoritmus

Navržený algoritmus je dekomponován na pět různých modulů, které se starají o jednotlivé funkce. Všechny moduly vycházejí ze společného předka — třídy `AbstractModule`.

D.2.1 Exekuční modul

Exekuční modul je zodpovědný za provádění pohybu jednotek při simulaci. Modul získá od vyhledávacího modulu aktuální cestu jednotky a zaznamenává průběh cesty pro případné použití v dalších modulech. Interface modulu definuje třída `AbstractPathExecutingModule`. Existuje pouze jeden typ exekučního modulu, a to `SimplePathExecutingModule`.

D.2.2 Plánovací modul

Plánovací modul je zodpovědný za plánování jednotky a všech jejich podřízených podle dané metody plánování. Interface plánovacích modulů definuje třída `AbstractPlanningModule`. Jsou implementovány tři typy modulů:

- `SimplePlanningModule` — jednoduchý plánovací modul, který je znázorněn na obrázku 2.1.
- `PeriodicPlanningModule` — modul, který implementuje techniku periodického přeplánování (viz 3.2.1). Modul obsahuje tři parametry:
  - `fragmentLength` — délka jednotlivých fragmentů.
  - `minFragmentLength` — minimální délka fragmentu. Pokud by měl být poslední fragment kratší, spojí se s předchozím fragmentem.
  - `replanFrequency` — perioda přeplánování.
- `CurvePlanningModule` — modul implementující techniku fragmentace po zatáčkách (viz 3.2.2). Modul má parametr `curvephi0`, který představuje hodnotu φ_0 použitou v algoritmu.

D.2.3 Modul synchronizace příchodu

Modul synchronizace příchodu je volán plánovacím algoritmem pro finalizaci nalezené cesty. Třída `AbstractSpeedSynchronizingModule` definuje interface modulu. Jsou implementovány čtyři typy modulů:

- `IgnoringSpeedSynchronizingModule` — modul bez synchronizace příchodu.
- `AddingSpeedSynchronizingModule` — modul aditivní změny rychlosti (viz 3.3.2).
- `MultiplicativeSpeedSynchronizingModule` — modul multiplikativní změny rychlosti (viz 3.3.1).
- `NormalizingSpeedSynchronizingModule` — modul adaptivní změny rychlosti (viz 3.3.3).

D.2.4 Vyhledávací modul

Vyhledávací modul je zodpovědný za nalezení cesty ze současné pozice jednotky (kterou poskytuje modul pozice) do cíle (který poskytuje plánovací modul). Interface tohoto modulu je definován třídou `AbstractPathFindingModule`. Jsou implementovány dva typy modulů:

- `SimplePathFindingModule` — jednoduchý vyhledávací algoritmus A*.
- `CombinedPathFindingModule` — vyhledávací algoritmus A* nad kombinovaným prohledávacím prostorem (viz 3.1.1).

D.2.5 Modul pozice

Modul pozice generuje z logického umístění jednotky její fyzické umístění na mapě (které je posléze použito pro vizuální reprezentaci jednotky). Toto fyzické umístění může být generováno přímo z hrany, kterou jednotka prochází, nebo může být vychýleno z různých důvodů (detekce a řešení kolizí). Interface modulu je definován třídou `AbstractPositioningModule`. Jsou implementovány dva typy modulů:

- `SimplePositioningModule` — přímý výpočet fyzického umístění z logického.
- `CollisionAvoidancePositioningModule` — modul s detekcí kolizí a jejich řešením metodou změny fyzického umístění (viz 3.4.1).

D.3 Jednotka

Jedna jednotka ve formaci je prezentována objektem třídy `Troop`. Inicializace jednotky probíhá ve čtyřech krocích:

1. Nejprve se nastaví mapa metodou `setAreaGraph(AbstractAreaGraph*)`.
2. Poté se přiřadí jednotka na příslušné místo ve formaci metodou `setFormationSpot(Formation*,FormationSpot*)`.
3. Proběhne inicializace všech modulů zadaným poskytovatelem modulů `initializeModules(ModuleProvider*)`.
4. Na závěr je jednotka umístěna na svoji počáteční pozici metodou `setPosition(QPointF)`.

Chování jednotky je zvenku ovlivňováno následujícími metodami:

- `moveTo(QPointF)` (resp. `waypointTo(QPointF)`) použijí plánovací modul pro naplánování cesty na dané místo (resp. přidání bodu cesty).
- Metoda `logic(double)` dostane na vstupu počet milisekund, které uběhly od posledního volání. Následně tento parametr předá exekučnímu modulu, který je zodpovědný za provedení pohybu jednotek.

D.4 Formace

Schéma formace, jak je popsáno v kapitole 1.4, je implementováno třídou `FormationSchema`. Třída v sobě sdružuje seznam bodů formace (reprezentovaných třídou `FormationSpot`) a vazeb formace (`FormationBinding`). Třída poskytuje metody pro ruční konstrukci schématu.

Podtřída `ConstructibleSchema` poskytuje interface pro schémata, které se automaticky konstruují podle zadaných parametrů (počet jednotek ve formaci a velikost prostoru, který má formace zabírat). Pro každý typ schématu je pak implementována zvláštní třída (jmenovitě `BoxSchema`, `DiamondSchema`, `LineSchema` a `WedgeSchema`).

Objekt třídy `Formation` pak představuje konkrétní instanci formace (viz 1.5), ve které jsou bodům formace vzájemně jednoznačně přiřazeny jednotky. Konstrukci celé formace včetně jednotek obstarává třída `Formations`.

D.5 Fitness

Statickou fitness formace (podle kapitoly 4.1) počítá třída `FitnessEvaluator`. Třída `FitnessManager` tento vyhodnocovací objekt obsluhuje a vytváří si seznam záznamů typu `FitnessRecord`. Jakmile je přesun formace ukončen, uloží data o něm do složky logu v následujícím formátu:

- *nnnnn*`.log` — kde *nnnnn* je pěticiferné číslo udávající pořadí záznamu (pro rozlišení více záznamů o přesunech ve stejné složce). Soubor je prázdný.
- *nnnnn*`.log.data.txt` — soubor s hodnotami statické fitness. První položka řádku udává časové razítko záznamu, druhá položka fitness úhlu, třetí fitness vzdálenosti a čtvrtá kombinovanou fitness.
- *nnnnn*`.log.data.png` — obsahuje graf s daty z předchozího souboru.
- *nnnnn*`.log.int.txt` — soubor s hodnotami dynamické fitness. První položka řádku udává parametr k, druhá hodnota udává hodnotu funkce dynamické fitness $F(k)$ v bodě k.
- *nnnnn*`.log.int.png` — obsahuje graf s daty z předchozího souboru.

Třída `ExpFitter` se používá pro nafitování hodnot dynamické fitness na exponenciální funkci, čímž ji činí vhodnější pro porovnávání (viz kapitola 4.2).

D.6 Scénář

Třída `Scenario` reprezentuje veškeré informace o scénáři - obsahuje v sobě mapu a nastavení formace.

Scénář lze ovládat v jednom ze tří módů (viz C.5). Pro každý mód existuje speciální třída, tzv. manažer módu, který v sobě spojuje logiku jednotlivých komponent a volitelně grafického rozhraní. Interface těchto manažerů definuje třída `ScenarioManager`.