FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Interaktivní prostředí pro vizualizaci a testování udržování formací v multi-agentním hledání cest
Student:	Jakub Votrubec
Vedoucí:	prof. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je vytvoření interaktivního grafického prostředí pro vizualizaci a testování udržování formací v diskrétní variantě multi-agentního hledání cest v dvourozměrných mřížkových mapách. Předpokládáme, že prostředí bude umožňovat zadávání instance problému a editovat mapu, kde se agenti pohybují. U vizualizační fáze předpokládáme schopnost reagovat na dynamické změny prováděné uživatelem, tj. například na nové překážky nebo změny cílových pozic agentů.

Úkoly pro řešitele jsou následující:
1. Prozkoumat existující algoritmy pro multi-agentní hledání cest (MAPF) a dále jejich rozšíření, které zohledňují vzájemné pozice agentů při pohybu v mřížkové mapě.
2. Navrhnout interaktivní grafické prostředí, které umožní vizualizovat (animovat) plány generované těmito algoritmy.
3. Grafické interaktivní prostředí implementovat za použití vhodných grafických knihoven a otestovat jej v relevantních scénářích.
4. Vytvořený software zdokumentovat po uživatelské a programátorské stránce.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 3. prosince 2021 v Praze.

Bakalářská práce

INTERAKTIVNÍ PROSTŘEDÍ PRO VIZUALIZACI A TESTOVÁNÍ UDRŽOVÁNÍ FORMACÍ V MULTI-AGENTNÍM HLEDÁNÍ CEST

Jakub Votrubec

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: prof. RNDr. Pavel Surynek, Ph.D.
11. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především svému vedoucímu práce za pomoc, rady, trpělivost a konstruktivní kritiku, kterou mi poskytoval během tvorby mé práce. Dále bych chtěl poděkovat své rodině a přátelům za podporu a pochopení během mých studií.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 .....

Abstrakt

V teoretické části práce se zabýváme seznámením čtenáře s problematikou udržování formací v multi-agentním prostředí a vysvětlením algoritmu použitého v praktické části. V praktické části poté navrhujeme a implementujeme interaktivní aplikaci pro vizualizaci a testování udržování formací v multi-agentním prostředí.

Zvolený problém jsme vyřešili vybudováním interaktivní aplikace v herním enginu Godot a postavením struktury pro konkrétní implementace algoritmů v jazyce C++. K testování používáme algoritmus pro udržování formací, kterému se věnujeme v literární rešerši.

Hlavním výsledkem je aplikace, která umožňuje uživatelsky přívětivý způsob využití algoritmu pro řešení uživatelem zadaných problémů, dále umožňuje problémy upravovat a zkoušet najít jejich optimální řešení.

Kompletní zdrojový kód a již zkompilovanou aplikaci připravenou na spuštění lze nalézt v přiloženém médiu.

Klíčová slova vizualizační aplikace, multi-agentní hledání cest, udržování formací, navádění agentů, modifikovatelnost problému hledání cest, interaktivní prostředí, C++, Godot

Abstract

In the theoretical background, we introduce the reader to the problem of maintaining formations in a multi-agent environment and explain the algorithm used in the practical part. In the practical part, we then design and implement an interactive application for visualizing and testing formation maintenance in a multi-agent environment.

We solved the chosen problem by building an interactive application in the Godot game engine and creating a structure for implementations of the algorithms in C++. For testing, we use the formation maintenance algorithm discussed in the theoretical background.

The main result is an application that provides a user-friendly way of using the algorithm to solve user-specified problems, as well as allowing us to modify the problems and try to find their optimal solution.

The complete source code and the already compiled application ready to run can be found on the attached media.

Keywords visualization application, multi-agent pathfiding, formation maintenance, navigation of agents, modifiable pathfinding tasks, interactive environment, C++, Godot

Seznam zkratek

PF	Vyhledávání cest pro jednoho agenta (Pathfinding)
AI	Umělá inteligence (Artificial intelligence)
LRA*	A* s lokální opravou (Local Repair A*)
CA*	Kooperativní A* (Cooperative A*)
MAPF	Multi-agentní vyhledávání cest (Multi-agent Pathfinding)
CBS	Hledání cest založené na konfliktech (Conflict Based Search)
CT	Strom omezení (Conflict Tree)
MAiF	Udržování formací v multi-agentním hledání cest (Moving Agents in Formations)
SWARM-MAPF	Kombinace rojového algoritmu a algoritmu pro multi-agentní hledání cest
CBS-M	Varianta hledání cest založené na konfliktech, která minimalizuje rozpětí
UI	Uživatelské rozhraní (User interface)

Kapitola 1
Úvod

Vyhledávání cest je algoritmická úloha pro zjišťování optimální cesty pro agenta v rámci předem známého prostředí a určené startovní a cílové pozice. Udržování formací v multi-agentním prostředí je algoritmická úloha vyhledávání trasy pro více agentů najednou s tím, že musí během cesty co nejvíce udržovat zadanou formaci (rozložení agentů vzhledem k vedoucímu agentovi). Toto vyhledávání je důležité pro logistické úkony nad více agenty nebo roboty. Slouží k lepší organizaci skupiny agentů a optimálnímu přístupu agentů k zadanému úkolu (dorazí ve formaci nebo udrží formaci nejlépe přizpůsobenou jejich zadanému úkolu).

Naše práce najde využití při hledání optimální cesty v libovolně navrženém prostředí, například při potřebě navést roboty v určité formaci k cíli s tím, že pro efektivitu provedení cíle je potřeba přístupu robotů k práci v určité formaci, dále například při převozu zboží v určité formaci za účelem ochrany cíle či organizace přesunu a v neposlední řadě se může jednat o využití ve strategických videohrách, kde je přesun jednotek ve formaci, která je nejlepší pro nejvyšší efektivitu jednotek, jedním ze základních stavebních kamenů strategického postupu k vítězství. Další využití naší aplikace bude při testování algoritmů pro hledání cest. V aplikace bude možné zadávat a upravovat problémy, které dobře otestují zvolený algoritmus.

V práci se zabýváme přiblížením aktuálních algoritmů pro řešení multi-agentního hledání cest v zadané formaci, jedním z nich je algoritmus od [1] týmu vedeného Jiaoyang Li, který řeší problematiku udržování formací v multi-agentním prostředí.

Hlavním cílem naší práce je vybudování aplikace, která bude umožňovat mnoho užitečných funkcí nad daným problémem hledání cest. Aplikace bude umožňovat vizualizaci cest, algoritmem vyhodnocených jako optimální, a tím zlepšovat možnost uživatele si představit zadaný problém a jeho potenciální řešení.

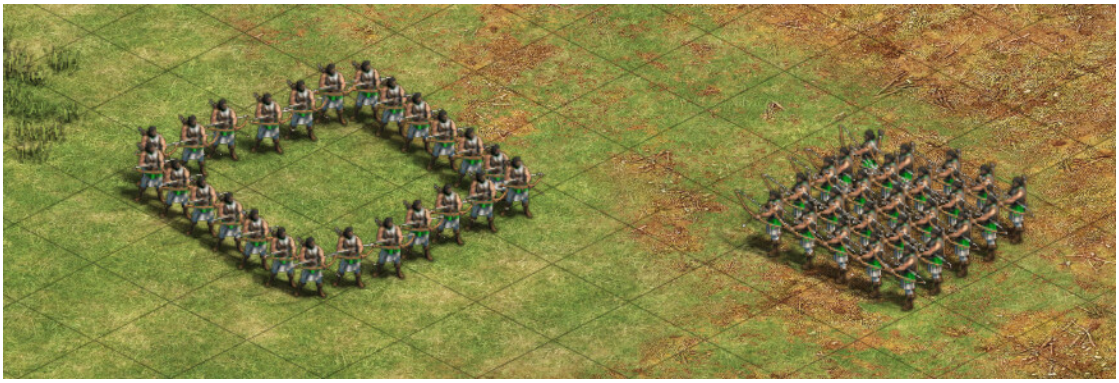
Velkou výhodou oproti aplikacím, které pouze vizualizují zadaný problém, bude možnost modifikovat zadání. Uživatel bude schopný upravovat mapu přidáváním či odebíráním kusů terénu a tím modifikovat problém dle své potřeby. Uživatel bude dále schopný si přizpůsobovat zadání úlohy přidáním nebo odebráním agentů, kteří mají udržovat jím zadanou formaci, a dostat se ze zadaných startovních pozic do cílových. Simulaci bude možné kdykoliv pozastavit a upravit scénář, aplikace následně přepočítá cesty agentů a po opětovném spuštění bude pokračovat ve vizualizaci. Možnost úpravy cesty v „reálném“ čase navíc umožňuje vizualizaci změn prostředí a dopadu těchto změn na výsledné cesty a možnosti udržení zadané formace.

V naší aplikace se také bude moci testovat aktuální průběh simulace. Bude možné si zobrazit statistiky cest agentů ze startu do cíle, jejich soudržnost (schopnost udržet se po cestě v zadané formaci) apod.

V práci se nezaobíráme vymýšlením nových nebo úpravou již existujících algoritmů pro udržování formací v multi-agentním prostředí, pouze využíváme již existující. Jediné modifikace algoritmů jsou pouze jejich integrace do struktury a funkčnosti aplikace jako celku.

Práci jsem si zvolil z důvodu mého zájmu o dané téma. Hledání optimálních přístupů a cest k dosažení určitého cíle je mou zálibou. Zajímá mě také tvorba aplikace jako celku spojujícího mnoho komponentů dohromady, v tomto případě algoritmu, jeho vizualizace a interaktivity s uživatelem.

Obrázek 1.1 [2] Ukázka využití formací při letu dronů.



Obrázek 1.2 [3] Ukázka využití formací ve videohře Age of Empires.

Kapitola 2
Cíl

Cílem práce je vytvoření interaktivního grafického prostředí pro vizualizaci a testování udržování formací v diskrétní variantě multi-agentního hledání cest v dvourozměrných mřížkových mapách.

Je předpoklad, že prostředí bude umožňovat zadávání instance problému, tj. počáteční a cílové pozice agentů, a stejně tak bude umožňovat editovat mapu, kde se agenti pohybují. U vizualizační fáze se předpokládá schopnost agentů reagovat na dynamické změny prováděné uživatelem, tj. například na změny v mapě (nové překážky) nebo změny cílových pozic agentů.

V první části bakalářské práce bude cílem prozkoumání existujících algoritmů pro udržování formací v multi-agentním prostředí, dále, pokud to bude nezbytné, rozšíření zvoleného algoritmu tak, aby zohledňoval dynamicky se měnící pozice ostatních agentů a interakci uživatele s mapou v podobě změny polohy, přidání či odebrání překážek a změny cílových pozic agentů.

Cílem další části práce je navrhnout interaktivní prostředí pro ovládání aplikace. Bude potřeba použít vhodnou grafickou knihovnu a zvolit si vhodný programovací jazyk na implementaci.

Součásti práce bude výsledkem vytvoření dostatečně obsáhlé dokumentace aplikace. Součástí dokumentace bude část uživatelská, která bude obsahovat instrukce pro uživatele pro ovládání aplikace. Druhou částí dokumentace bude část programátorská, jež bude podrobně vysvětlovat konkrétní implementaci jednotlivých částí aplikace.

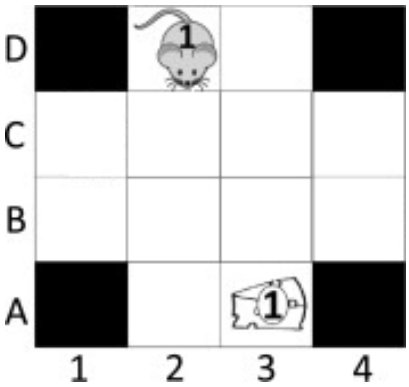
V poslední části práce ukážeme, že naše aplikace správně testuje obvykle zadávané problémy v obvyklých scénářích. Ukážeme záznamy obrazovky zadání a poté výsledky testování.

Kapitola 3
Teoretická východiska

Nejprve začneme s představením problematiky hledání cest pouze pro jednoho agenta a uvedeme několik algoritmů pro to používaných 3.1. Dále se přesuneme na hledání cest pro více agentů zároveň, vysvětlíme čtenáři rojové a konfliktní algoritmy 3.3 a poté představíme téma udržování formací v multi-agentním hledání cest. Následující sekce 3.6 vysvětluje možné řešení udržování formací v multi-agentním hledání cest, a to algoritmus pro udržování formací v diskrétním multi-agentním prostředí ze sbírky Formation in Congested Environments [1]. V poslední sekci literární rešerše 3.7 zvažujeme výběr programovacího jazyka a grafické knihovny použité pro naši bakalářskou práci.

3.1 Problém hledání cest

[4] Hledání cest je problém nalézání cesty mezi dvěma vrcholy v grafu. Je to jeden ze základních a nejdůležitějších problémů v odvětví umělé inteligence (AI), který byl velmi důkladně zkoumán. Tento problém se objevuje v mnoha reálných aplikacích, například v GPS navigaci [5], navádění robotů [6], síťovém provozu [7] a také v mnoha kombinatorických problémech [8, 9]. Problémy hledání optimálních cest se obvykle řeší pomocí vyhledávacích algoritmů založených na A* algoritmu [10]. Takové algoritmy provádějí tzv. „best-first“ vyhledávání, která používají prioritní fronty a heuristiky. Best-first vyhledávání se řídí funkcí pro odhad ceny $f(n) = g(n) + h(n)$, kde $g(n)$ je cena nejkratší nalezené cesty z počáteční pozice do pozice n a $h(n)$ je heuristická funkce, která odhaduje cenu cesty z pozice n do nejbližší cílové pozice. Pokud je funkce h *přípustná*, to znamená, že nikdy nepřecení nejkratší cestu z pozice n do cílové pozice, potom A* (a další algoritmy řízené stejnou funkcí pro vyhodnocení ceny) garantuje nalezení optimální cesty z počáteční do cílové pozice, pokud taková cesta existuje [11].



Obrázek 3.1 Ukázka problému hledání cest na myši a sýru [4]

3.2 Hledání cest pomocí algoritmu A*

Algoritmus A* patří mezi heuristické algoritmy. Používá tedy heuristickou (vyhodnocovací) funkci k ohodnocení a porovnání jednotlivých uzlů a nalezení optimální cesty. Abychom při hledání museli projít a rozšířit co nejméně uzlů pro získání optimální cesty, musí hledací algoritmus konzistentně dělat dobře informovaná rozhodnutí o tom, který uzel rozšířit jako další. Pokud rozšíří uzly, které jasně nemohou být součástí optimální cesty, je to zbytečná práce. Na druhou stranu, pokud bude ignorovat uzly, které mohou být součástí optimální cesty, může se stát, že občas nenalezne takovou cestu. Efektivní algoritmus potřebuje způsob ohodnocení dostupných uzlů, aby rozhodl, který jako další rozšíří. Předpokládejme, že pro libovolný uzel n lze vypočítat nějakou *vyhodnocovací funkci* $f(n)$. Navrhneme takovou funkci níže, ale nejdříve popíšeme, jak by funkci používal vyhledávací algoritmus. [10]

Definujeme vyhodnocovací funkci $f(n)$ tak, že dostupný uzel s nejmenší hodnotou f je ten uzel, který by měl být rozšířený jako další. Potom můžeme vyhledávací algoritmus navrhnout takto. 1

Algorithm 1 A* [10]

Vstup: PF instance
$s.state = OPEN$
$s.cost = f(s)$
while $OPEN$ *not empty* **do**
  $n \leftarrow$ nejlepší $OPEN$ uzel ▷ Remízy preferuj cílový uzel
  if n je cílový uzel **then**
    $s.state = CLOSED$
    return
  end if
  $s.state = CLOSED$
  for each n_i následník n **do**
    if $n_i.state \neq CLOSED$ **then**
      Označ n_i jako OPEN
    end if
    if $n_i.state = CLOSED \wedge f(n_i) < n_i.cost$ **then**
      $n_i.cost = f(n_i)$
      $s.state = OPEN$
    end if
  end for
end while

3.2.1 Vyhodnocovací funkce

Můžeme $f(n)$ rozepsat na součet dvou částí:

$$f(n) = g(n) + h(n) \tag{3.1}$$

kde $g(n)$ je cena optimální cesty z počátečního uzlu s do uzlu n a $h(n)$ je cena optimální cesty z n do cílového uzlu.

Nyní, pokud bychom měli odhady g a h, mohli bychom je sečíst k vytvoření odhadu$f$. Nechť $\hat{g}(n)$ je odhad $g(n)$. Jasnou volbou pro $\hat{g}$ je cena cesty z s do n s nejmenší dosavadní cenou nalezenou algoritmem. Toto implikuje $\hat{g}(n) \geq g(n)$.

Dále musíme získat odhad $\hat{h}(n)$ funkce $h(n)$. Zde budeme spoléhat na informace z domény problému. Mnoho problémů může být reprezentováno jako problém nalezení cesty s nejmenší cenou skrze graf, který má nějaké „fyzické“ informace, které mohou být použity na odhad $\hat{h}$. Například pro města spojená silnicemi, může být $\hat{g}(n)$ vzdálenost vzdušnou čarou mezi městy n a cílovým městem. [10]

3.2.2 Varianta A* s lokální opravou

A* s lokální opravou (Local Repair A*, dále LRA*) [12] popisuje třídu algoritmů často užívaných v video-herním průmyslu. Každý agent hledá cestu do cílové pozice pomoci A* algoritmu [10], ignorujíce všechny ostatní agenty kromě svých aktuálních sousedů. Agenti se poté začnou pohybovat po svých naplánovaných cestách, dokud nemá nastat kolize. Pokaždé, když by se agent měl přesunout do již zabrané pozice, namísto toho přepočítá zbytek své cesty. Základní algoritmus odpovídá plánování hrubou silou popsaném Zelinskym (1992) [13].

Zacyklení je možné (dokonce časté) při použití těchto algoritmů, tudíž je obvyklé přidat nějakou modifikaci pro vyhnutí se takovým problémům. Jedna možnost, použitá zde, je zvyšování agentovy *úrovně agitace* pokaždé, kdy je donucen změnit cestu. Poté je přidán náhodný šum do

vzdálenostní heuristiky v závislosti na úrovni agitace. Agenti se chovají stále více náhodně a tudíž s vysokou pravděpodobností vyhnou problémové oblasti a půjdou jinou cestou. [12]

Je známo, že LRA* má v obtížných podmínkách několik závažných nevýhod [13]. Pokud se v přeplněných oblastech vyskytují úzká místa, může jejich vyřešení trvat libovolně dlouho. Když jsou agenti zachyceni v úzkém místě, neustále mění trasy ve snaze uniknout, což vyžaduje úplný přepočet hledání A* téměř při každém časovém kroku. To vede k vizuálně rušivému chování, které je vnímáno jako neinteligentní. Každá změna trasy se provádí nezávisle, což vede k cyklům, v nichž stejné místo může být opakovaně navštěvováno ve smyčce. Další sekce navrhuje variantu A* algoritmu k překonání těchto problémů pomocí kooperativního vyhledávání. [12]

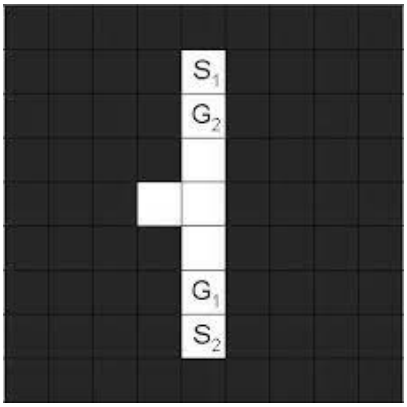
3.2.3 Varianta kooperativního A*

Kooperativní A* (Cooperative A*, dále CA*) [12] je algoritmus pro kooperativní hledání cest. Problém je rozdělen na série jedno-agentních hledání. Jednotlivá hledání jsou dělána ve tří-dimenzionálním časoprostoru a berou v potaz naplánované cesty ostatních agentů. Lze agentovi zadat příkaz *čekej*, aby mohl agent zůstat stát na své aktuální pozici. Poté, co je vypočítána cesta pro agenta, jsou stavy podél cesty označeny do *rezervační tabulky*. Záznamy v rezervační tabulce jsou považovány za neprůchodné a následní agenti se jim při vyhledávání vyhýbají.

Rezervační tabulka reprezentuje společné povědomí agentů o jejich individuálně plánovaných cestách. Je to jednoduchá datová struktura zaznamenávající pozice v časoprostoru. Volba datové struktury je nezávislá na prostorovém stavu agentů jako takových. Obecně, jednotlivý agenti se mohou lišit velikostí, nebo rychlostí a rezervační tabulka musí být schopná zaznamenat jakékoliv zabrané pozice.

Jednoduchá implementace je použití tří-dimenzionální tabulky (dvě prostorové dimenze a jedna časová). Každé políčko tabulky, které je zároveň součástí agentovi cesty, je označeno jako neprůchozí přesně po dobu, jakou na ní agent stráví, což zamezí jinému agentovi si naplánovat kolizní cestu. Jen velmi malá část tabulky bude vyplněna, proto může být tabulka efektivně implementována hašovací tabulkou s uspořádanou trojicí (x, y, t) jako klíčem.

Je důležité připomenou, že tento algoritmus nebude schopný vyřešit určitou třídu problémů. K tomu může dojít, když hladové řešení pro jednoho agenta zabrání jakémukoli řešení pro jiného agenta, viz například obrázek 3.2. Obecně jsou takové algoritmy citlivé na pořadí agentů, což vyžaduje, aby byly pro dobrý výkon zvoleny rozumné priority, například pomocí Latombeho plánování podle priorit. [14]



Obrázek 3.2 Tento jednoduchý problém nemůže být vyřešen pomocí CA* algoritmu. Jeden agent se musí přesunout z S_1 do G_1, zatímco druhý z S_2 do G_2. [12]

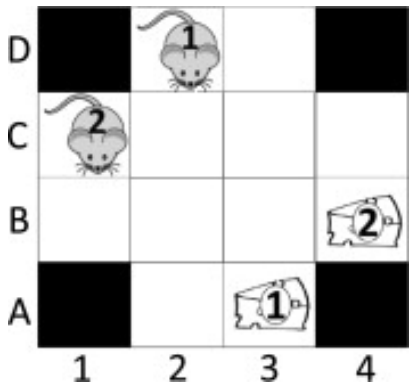
3.3 Problém multi-agentního hledání cest

Problém multi-agentního hledání cest (Multi-agent pathfinding, dále MAPF) je zobecněním problémů hledání cest pro $k \geq 1$ agentů. Obsahuje graf a počet agentů. Pro každého agenta je dána unikátní počáteční pozice a unikátní cílová pozice a úkolem je najít cestu pro všechny agenty z jejich počátečních pozic do jejich cílových pozic s podmínkou, že nesmí vzniknout kolize agentů během jejich přesunů. V mnoha případech je dán ještě další úkol, a to minimalizovat celkovou cenu řešení, například počet časových kroků potřebných k tomu, aby všichni agenty dosáhli svých cílových pozic. MAPF má praktické aplikace například ve videohrách [15], řízení provozu [16], robotice [17] a letectví [18].

Algoritmy řešící MAPF lze rozdělit na dvě třídy: *optimální* a *sub-optimální*. Nalezení optimálního řešení MAPF problému je NP-těžké [19], jelikož počet stavů roste exponenciálně s počtem agentů. Sub-optimální algoritmy jsou obvykle používány při vysokém počtu agentů. V těchto případech je cílem rychle najít cestu pro různé agenty a často je obtížné zaručit, že je řešení optimální. [4]

3.4 MAPF pomocí konfliktově založených algoritmů

„Conflict based search algorithm“ (dále CBS) řeší problém MAPF rozkladem na velké množství jedno-agentových problémů hledání cest s omezením. Každý z těchto problémů může být vyřešen v čase závislém na velikosti mapy a délce řešení, ale počet takových jedno-agentových problémů může být exponenciální. [4]



Obrázek 3.3 Ukázka multi-agentního hledání cest na myších a sýrech [4]

3.4.1 Definice pro CBS

Následující definice používáme ve zbytku této sekce.

- *Cesta* je pojem, který používáme pouze v kontextu jednoho agenta a pojem *řešení* k označení množiny k cest pro daných k agentů.
- *Omezení* je uspořádaná trojice a_i, v, t, kde se agent a_i nesmí nacházet na vrcholu v v časovém kroku t. Během průběhu algoritmu agentům budou přiřazena omezení. *Konzistentní cesta* pro agenta a_i je taková cesta, která splňuje všechna omezení. Obdobně *konzistentní řešení* je takové řešení, které se skládá z cest, pro které platí, že cesta pro jakéhokoliv agenta a_i splňuje omezení agenta a_i.

- *Konflikt* je uspořádaná čtveřice a_i, a_j, v, t kde jsou agent a_i a agent a_j, oba jsou na vrcholu v v časovém kroku t. Řešení (k cest) je *validní*, pokud jsou všechny jeho cesty bez konfliktů. Konzistentní řešení může být *nevalidní*, pokud navzdory tomu že individuální cesty jsou konzistentní s omezeními jejich agentů, mají tyto cesty stále konflikty.

Klíčovou ideou CBS je vytvořit sadu omezení a nalézt cesty, které dodržují tato omezení. Pokud mají tyto cesty konflikt, a tudíž jsou nevalidní, jsou tyto konflikty vyřešeny přidáním nových omezení. CBS pracuje na dvou úrovních. Na vyšší úrovni jsou nacházeny konflikty a přidávány omezení. Nižší úroveň nachází cesty, které dodržují omezení, pro jednotlivé agenty. V dalších sekcích představíme jednotlivé části celého procesu podrobněji. [4]

3.4.2 Vyšší úroveň CBS

V následující sekci popíšeme proces CBS na vyšší úrovni a ukážeme vyhledávací strom, který prohledává. [4]

3.4.2.1 Strom omezení

CBS na vysoké úrovni prohledává strom nazvaný *strom omezení* („constraint tree“, dále CT). CT je binární strom. Každý uzel N v CT se skládá z:

- **Množiny omezení** ($N.constraints$). Každé z těchto omezení patří jednomu agentovi. Kořen CT obsahuje prázdnou množinu omezení. Potomek uzlu v CT dědí omezení jeho rodiče a přidá jedno nové omezení pro jednoho agenta.
- **Řešení** ($N.solutions$). Množina k cest, jedna cesta pro každého agenta. Cesta pro agenta a_i musí dodržovat omezení agenta a_i. Cesty jsou hledané CBS nižší úrovně.
- **Celkové ceny** ($N.cost$) aktuálního řešení (sečtená cena všech cest jednotlivých agentů). Tato cena je také označována za f-hodnotu uzlu N.

Uzel N v CT je cílový uzel, pokud $N.solutions$ je validní, tzn. že množina cest pro všechny agenty nemá konflikty. CBS vyšší úrovně provede hledání nejlepšího výsledku na uzlech CT seřazených podle jejich cen. Remízy jsou vyhodnoceny ve prospěch CT uzlů s méně konflikty. Pokud je i nadále remíza, je vyřešena přístupem FIFO. [4]

3.4.2.2 Zpracování uzlu v CT

Nízkoúrovňové hledání (popsáno níže) je spuštěno se zadaným seznamem omezení a vrátí nejkratší cestu pro a_i v uzlu N. Jakmile je nalezena konzistentní cesta pro každého agenta (respektující jejich omezení), cesty jsou poté *validovány* s ohledem na cesty ostatních agentů. *Validace* probíhá iterováním skrze všechny časové kroky a porovnáváním pozic rezervovaných všemi ostatními agenty. Pokud žádní dva agenti nemají naplánováno být na stejné pozici ve stejný čas, je tento CT uzel N prohlášen za cílový uzel a jeho řešení ($N.solution$), které obsahuje tuto množinu cest, vráceno jako výsledek. Pokud ale během provádění *validace* je nalezen konflikt $C = (a_i, a_j, v, t)$ pro dva nebo více agentů, je validace pozastavena a uzel je prohlášen za necílový. [4]

3.4.2.3 Řešení konfliktů

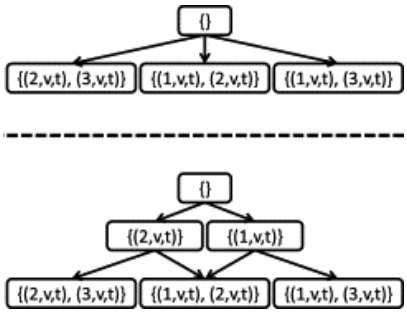
Řešení konfliktu necílového CT uzlu N, jehož řešení $N.solutions$ obsahuje nějaký *konflikt* $C_n = (a_i, a_j, v, t)$, víme že v jakémkoliv validním řešení může nanejvýš jeden z agentů v kolizi (a_i a a_j) být na pozici v v časovém kroku t. Tudíž alespoň jedno omezení (a_i, v, t) nebo

(a_j, v, t) musí být přidáno k množině omezení $N.constraints$. Abychom mohli garantovat optimálnost, obě možnosti jsou prozkoumány a uzel N je rozdělen do dvou synů. Oba nově vzniklé uzly zdědí množinu omezení z N. Levý syn řeší konflikt přidáním omezení (a_i, v, t) a pravý syn přidáním omezení (a_j, v, t). [4]

Všimněme si, že pro daný uzel CT N není třeba ukládat všechna jeho kumulativní omezení. Namísto toho stačí, aby si zapamatoval jen poslední omezení a zbytek načetl průchodem z uzlu N ke kořenovému uzlu přes své předky. Obdobně, kromě kořenového uzlu, nízkoúrovňové hledání by mělo být prováděno pouze pro agenta a_i, na kterého se vztahuje nové přidané omezení. Cesty ostatních agentů zůstávají stejné, jelikož na ně nebyla přidána žádná nová omezení. [4]

3.4.2.4 Konflikt $k > 2$ agentů

Může se stát, že při validaci mezi několika cestami je nalezen *k-agetní* konflikt pro $k > 2$. Existují dva způsoby jak takový k-agentní konflikt vyřešit. Můžeme vygenerovat k synů, přičemž každému přidáme omezení na $k - 1$ agentů (tzn. každý syn povoluje jen jednomu agentovi být na vrcholu konfliktu v v časovém kroku t). Nebo je ekvivalentní formalizací zaměřit se pouze na první dva agenty, u nichž byl nalezen konflikt, a větvit pouze podle jejich konfliktu. Tím se další konflikty ponechají pro hlubší úrovně stromu. To je znázorněno na obrázku 3.4. Horní strom představuje variantu CT, kde je povoleno k-cestné větvení pro jediný konflikt, který zahrnuje k-agentů pro případ, kdy $k = 3$. Každý následník přidá $k - 1 (= 2)$ nových omezení (pro všechny agenty kromě jednoho). Spodní strom představuje binární CT pro stejný problém. Všimněte si, že spodní prostřední stav je duplicitní stav, a pokud není použita detekce duplicit, budou v tomto uzlu dva výskyty místo jednoho. Jak je vidět, velikost nejhlubší vrstvy je v obou stromech stejná. Složitost obou přístupů je podobná, protože oba skončí s k uzly, každý s $k - 1$ s novými omezeními. Pro zjednodušení jsme implementovali a popsali pouze druhou možnost. [4]



Obrázek 3.4 [4] $(k = 3)$-cestné větvení CT (horní) a binární CT pro stejný problém (dolní).

3.4.2.5 Konflikt na hranách

Pro zjednodušení popisujeme pouze konflikty na vrcholech. Ale podle definice nemají agenti povoleno pohybovat se po stejné hraně v opačném směru ve stejný časový krok, pokud toto nastane, nazveme to *konflikt na hraně*. Definujeme *konflikt na hraně* jako uspořádanou pětici (a_i, a_j, v_1, v_2, t), kde si dva agenti „prohodí“ své pozice (a_i se přesune z v_1 na v_2, zatímco a_j se přesune z v_2 na v_1) mezi časovým krokem t a krokem $t + 1$. Na vyšší úrovni je s konflikty na hranách, tam kde je to možné, zacházeno jako s konflikty na vrcholech. [4]

3.4.3 Nižší úroveň CBS: nalezení cest pro CT uzly

Nízkoúrovňové hledání dostane na vstupu agenta a_i a množinu omezení spojené s a_i. Provede hledání v grafu pro nalezení optimální cesty pro agenta a_i, které splňuje všechna omezení, zatímco

Algorithm 2 CBS [4]

Vstup: MAPF instance
$Root.constraints = 0$
$Root.solution$ = najdi jednotlivé cesty pomocí low_level_CBS()
$Root.cost = estimate\_cost(Root.solution)$
Vlož *Root* do OPEN
while OPEN *not empty* **do**
  $P \leftarrow$ nejlepší uzel z OPEN ▷ řešení s nejnižší cenou
  Validuj cesty P dokud nenastane konflikt.
  if P nemá konflikt **then**
    return $P.solution$ ▷ P je cílové řešení
  end if
  $C \leftarrow$ první konflikt a_i, a_j, v, t v P
  for each agent a_i v C **do**
    $A \leftarrow$ nový uzel
    $A.constraints \leftarrow P.constraints + (a_i, v, t)$
    $A.solution \leftarrow P.solution$
    $A.solution$ je aktualizováno voláním low_level_CBS(a_i)
    $A.cost = estimate\_cost(A.solution)$
    if $A.cost <$ inf **then** ▷ řešení nalezeno
      Vlož A do OPEN
    end if
  end for
end while

kompletně ignoruje ostatní agenty. Prohledávaný prostor pro nízkoúrovňové hledání má dvě dimenze: prostorovou dimenzi a časovou dimenzi[1]. Jakýkoliv algoritmus na hledání cest pro jednoho agenta může být použit pro nacházení cest pro agenta a_i, zatímco ověřuje, že jsou splněny všechna omezení. Navrhli jsme nízkoúrovňové vyhledávání CBS pomocí A* algoritmu 3.2, který zpracovává omezení následovně. Vždy pokud je stav v, t vygenerován, kde v je pozice a t je časový krok a existuje omezení a_i, v, t v aktuální CT (vysokoúrovňovém) uzlu, je tento stav zahozen. Heuristika, kterou používáme, je podle nejkratší vzdálenosti v prostorové dimenzi, ignorujíce ostatní agenty a jejich omezení. [4]

Pro případy, kdy mají dva stavy nízkoúrovňového A* stejnou f-hodnotu, používáme na rozhodnutí remízy následující metodu. Preferovány jsou stavy obsahující konflikt s menším počtem agentů. Například, pokud stavy $s_1 = (v_1, t_1)$ a $s_2 = (v_2, t_2)$ mají stejnou f-hodnotu, ale na vrcholu v_1 se v časovém kroku t_1 nacházejí dva další agenti, zatímco na vrcholu v_2 se nenacházejí žádní další agenti v časovém kroku t_2, potom bude dříve rozšířen stav s_2. Tato metoda rozhodování zlepší celkový čas běhu až dvakrát oproti běžnému řešení remíz. Detekce duplicitních stavů a jejich čištění také zrychlí hledání. Na rozdíl od hledání cest pro jednoho agenta, stavový prostor zahrnuje i časovou dimenzi a dynamické „překážky“ tvořené omezeními. Tudíž dva stavy jsou brány jako duplicitní, pokud je identická jak pozice a_i, tak časový krok. [4]

[1] Prostorová dimenze může obsahovat více vnitřních dimenzí. Například 2D mapa obsahuje dvě dimenze. k-dimenzionální graf spolu s časovou dimenzí má ve výsledku vyhledávací prostor o $k + 1$ dimenzích.

3.5 Problém udržování formací v multi-agentním hledání cest

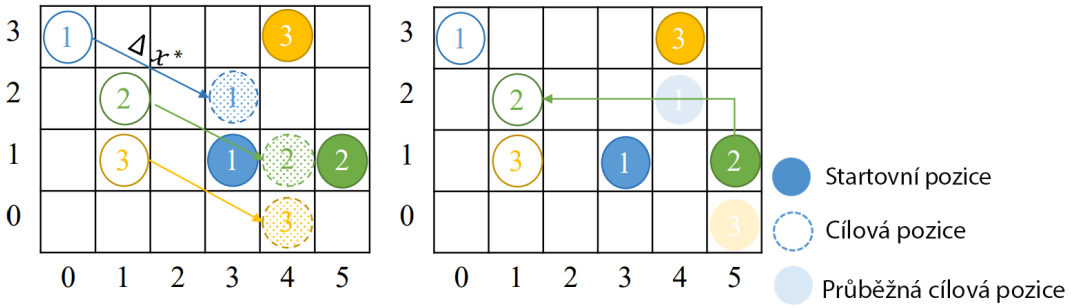
Plánování cest pro více agentů ve známém zhuštěném prostředí je důležitý problém, který vychází z mnoha multi-agentních aplikací v reálném světě, například vozidla přesunující letadla [20], roboti ve skladu [21] a videoherní postavy [15]. V mnoha aplikacích je zároveň důležité, aby agenti udržovali během přesunu zadanou formaci zároveň s vyhýbáním se překážkám. Například autonomní vozidla se musejí pohybovat ve specifické formaci, aby přesunuli velké objekty nebo si mezi sebou udrželi komunikační síť. Herní postavy nebo armádní personál se musí pohybovat ve specifické formaci, aby ochránil zranitelné agenty. Tyto aplikace obsahují 2 klíčové úkoly: (1) plánování bezkolizní cesty pro více agentů a (2) udržování agentů ve formacích. Úkol (1) může být vyřešen multi-agentním hledáním cest (MAPF) algoritmy, které typicky minimalizují jednu z několika možných metrik, kterou stanoví cenu nalezené cesty. MAPF je NP-těžký na vyřešení optimálně [22, 23]. Tento úkol může být vyřešen redukcí na ostatní dobře probádané kombinatorické problémy nebo dedikovaným MAPF algoritmem. Úkol (2) se dá řešit formaci-kontrolujícími algoritmy, které se snaží obnovit zadanou formaci, pokud je narušena nějakou překážkou (například algoritmy vedoucí-následník, grafové, nebo rojové algoritmy). Nicméně, tyto algoritmy obvykle nefungují dobře ve zhuštěných prostředích a negarantují úplnost. Níže v textu 3.6 představujeme optimální řešení problému, který řeší oba dva úkoly a zároveň garantuje úplnost.[1]

3.6 Pohyb agentů ve formacích v diskrétním prostředí

V této sekci se budeme zabývat vysvětlením algoritmu popsaného ve výzkumném článku [1] o algoritmu vyvinutém v USA, v Kalifornii, na Univerzitě Jižní Kalifornie. Algoritmus řeší problém Udržování formací v multi-agentním hledání cest (Moving Agents in Formations), dále jen MAiF. Problém kombinuje úlohu nalezení krátké a bezkolizní cesty pro více agentů a úlohu udržení jich co nejblíže zadané formaci. Algoritmus řeší problém systematicky ve dvou fázích. První fáze (v otevřených částech cesty) je inspirována tzv. rojovými algoritmy a druhá fáze (ve zhuštěných částech cesty) algoritmy pro multi-agentní hledání cest. V první fázi algoritmus vybere vedoucího agenta a najde pro něj cestu, která je dostatečně vzdálená od překážek tak, aby mohli ostatní agenti udržet okolo něj zadanou formaci. Také identifikuje kritická místa, ve kterých nejsou agenti schopni udržet formaci, a tudíž jejich konkrétní cesta musí být přenechána na vyhodnocení druhou fází. Druhá fáze zpřesní cestu agentů pro tyto kritické segmenty.[1]

3.6.1 Definice problému

Nejprve si formalizujeme MAiF v Kartézském systému souřadnic. Je dán neorientovaný graf $G = (V, E)$ v d-dimenzionálním Kartézském systému souřadnic. Vrcholy V korespondují s pozicemi a hrany E korespondují s přechody mezi pozicemi. Pozice $v_i \in V$ je definována svými souřadnicemi $v_i = (v_{i1}, \ldots, v_{id}) \in \mathbb{R}^d$. Je také daná množina M agentů $\{a_i | i = 1, \ldots, M\}$, každý s unikátní startovní pozicí $s_i \in V$ a unikátní cílovou pozicí $g_i \in V$. Čas je diskretizován do kroků. Mezi po sobě jdoucími kroky se může agent buď pohnout, nebo setrvat na aktuální pozici. *Cesta* π_i pro agenta a_i je sekvencí pozic, jedna pro každý krok, která posune agenta a_i z jeho startovní pozice s_i do jeho cílové pozice g_i. $\pi_i(t) \in V$ je pozice agenta a_i v kroku t. Agenti zůstávají navždy na své pozici po dokončení jejich cesty. *Kolize* mezi cestami agentů a_i a a_j je buď kolizí na vrcholu v v kroku t, nebo kolizí na hraně (a_i, a_j, u, v, t), tj. agent a_i se pohybuje z pozice u do pozice v a agent a_j se pohybuje z pozice v do pozice u v kroku t. Řešením je množina M cest $\{\pi_i | i = 1, \ldots, M\}$, pro každého agenta jedna, taková, že nenastane kolize mezi žádnými dvěma cestami. *Rozpětí* řešení je maximální délka všech cest v řešení, tj. $\max_{1 \leq i \leq M} |\pi_i|$.[1]



Obrázek 3.5 Ukázka MAiF problému. Levá část zobrazuje vytečkované pozice agentů po optimální transformaci $\Delta\mathbf{x}^* = (3, -1)$ do cílových pozic. V pravé části zelená čára zobrazuje cestu vedoucího agenta.[1]

Formace popisuje relativní pozice agentů a je definována souřadnicemi všech agentů ze skupiny. *Formace* v kroku t je M-tice $l(t) = \langle \pi_1(t), \ldots, \pi_M(t) \rangle$ definovaná souřadnicemi pozic všech agentů v kroku t. *Požadovaná formace* je M-tice $l_g = \langle g_1, \ldots, g_M \rangle$ definovaná souřadnicemi cílových pozic všech agentů. *Vzdálenost formací* $\mathcal{F}(l, l')$ mezi dvěma formacemi l a l' popisuje nejmenší možný počet kroků potřebných k transformaci formace l na formaci l'. Je definována jako suma vzdáleností všech agentů mezi jejich původními pozicemi a pozicemi po $\Delta\mathbf{x}$ transformaci do l', minimalizována přes všechny takové transformace. Formálně, nechť $l = \langle u_1, \ldots, u_M \rangle$ a $l' = \langle v_1, \ldots, v_M \rangle$. Potom,

$$\begin{aligned} \mathcal{F}(l, l') &= \min_{\Delta\mathbf{x}} \sum_{i=1}^{M} \|u_i - (v_i + \Delta\mathbf{x})\|_1 \\ &= \sum_{j=1}^{d} \min_{\Delta\mathbf{x}_j} \sum_{i=1}^{M} |(u_{ij} - v_{ij}) - \Delta\mathbf{x}_j| \\ &= \sum_{j=1}^{d} \sum i = 1^{M} |(u_{ij} - v_{ij}) - \Delta\mathbf{x}_j^*|, \end{aligned} \tag{3.2}$$

kde pro každou dimenzi j, $\Delta\mathbf{x}_j^*$ je medián všech rozdílů $u_{ij} - v_{ij}$[2]. *Odchylka formací* $\mathcal{F}(t)$ v kroku t je vzdálenost formací mezi formacemi $l(t)$ v kroku t a požadované formace l_g, tj. $\mathcal{F}(t) = \mathcal{F}(l(t), l_g)$. *Celková odchylka formací* je sumou všech odchylek formací přes všechny kroky, to jest, $\sum_{t=0}^{T} \mathcal{F}(t)$, kde $T = \max_{1 \le i \le M} |\pi_i|$. Obrázek 3.5 (levá část) ukazuje příklad: Rozdíl mezi souřadnicemi startovní pozice a cílové pozice pro každého agenta v každé dimenzi $\{s_{i1} - g_{i1} | i = 1, 2, 3\}$ a $\{s_{i2} - g_{i2} | i = 1, 2, 3\}$ jsou $\{3, 4, 3\}$, respektive $\{-2, -1, 2\}$. Tudíž v kroku 0 je optimální transformace $\Delta\mathbf{x}^* = (3, -1)$ a rozptyl formace v kroku 0 je tedy $\mathcal{F}(0) = 5$.[1]

Kvalita řešení problému MAiF $\{\pi_i | i = 1, \ldots, M\}$ je hodnocena jak podle jeho rozpětí, tak podle jeho celkového rozptylu formace. Není jednoduché optimalizovat obě dvě požadované metriky zároveň. Nicméně minimalizování pouze jedné z nich není dostatečné. MAPF algoritmy mohou produkovat MAiF řešení, která minimalizují rozpětí, ale nesnaží se udržet požadovanou formaci. Naproti tomu triviální řešení držení všech agentů nehybně v zadané formaci minimalizuje celkový rozptyl formace, ale nedostane agenty do požadovaných cílových pozic.[1]

[2] Využíváme toho, že medián minimalizuje sumu absolutních rozdílů k M číslům.

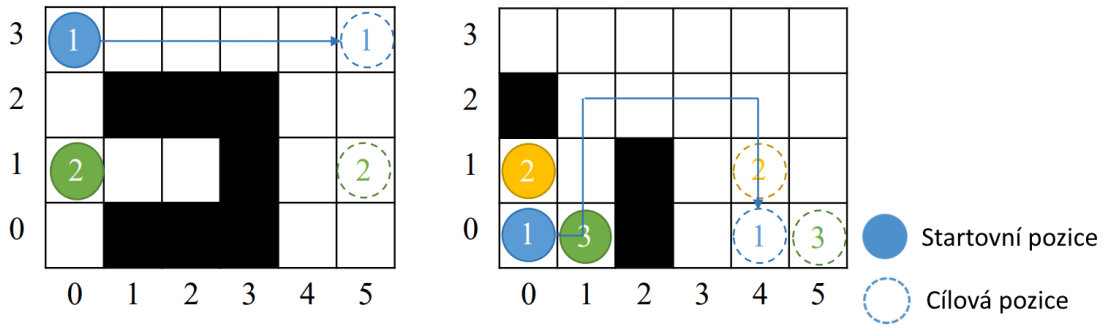
3.6.2 Řídící jednotka Vedoucí-následník

Řídící jednotka vedoucí-následník je klasická třída algoritmů, které se využívají pro udržování a navádění formací a již byla hluboce a podrobně zkoumána a aplikována v robotice. Jak již jméno naznačuje, řídící jednotka vedoucí-následník vybere jednoho z agentů jako vedoucího a zbytek agentů označí jako následníky. Vedoucí agent se pokusí najít cestu k cílové pozici, zatímco každý následovník následuje vedoucího agenta, udržuje svou relativní pozici k němu a těsně dodržuje zadanou formaci.[1]

Adaptujeme jednoduchou vedoucí-následník řídící jednotku do MAiF. Nejprve libovolně vybereme agenta a^* jako vedoucího a naplánujeme nejkratší cestu π^*, během tohoto plánování ignorujeme ostatní agenty. V každém časovém kroku t se vedoucí agent a^* posune o jeden krok po své naplánované cestě π^*. Jakmile je určena další pozice $\pi^*(t+1)$ vedoucího agenta a^*, následníci (tzn. všichni agenti kromě a^*) se pokusí posunou na svou další pozici vzhledem k vedoucímu agentovi a své pozice ve formaci. Konkrétněji je další pozice následníka a_i

$$\pi_i(t+1) = arg\min_{v \in S} ||(v - \pi^*(t+1)) - (g_i - g^*)||_1, \tag{3.3}$$

kde g^* je cílová pozice agenta a^* a $S \subseteq V$ obsahuje aktuální pozice $\pi_i(t)$ agenta a_i a jeho sousední pozice.[1]



Obrázek 3.6 Dvě MAiF instance na mřížce se 4 sousedy. Černá políčka jsou překážky. První dimenze je horizontální, druhá dimenze je vertikální. V obou instancích modrá čára ukazuje cestu vedoucího agenta[1]

Řídící jednotka vedoucí-následník ale není úplná. Obrázek 3.6 ukazuje dva příklady. Předpokládejme že agent a_1 je zvolen jako vedoucí v obou příkladech. Obrázek 3.6 (vlevo), agent a_2 se posune o jeden krok doprava, při následování agenta a_1 v každém časovém kroku, ale zasekne se na pozici (2, 1) poté co agent a_1 dorazí na pozici (2, 3) v časovém kroku 2. Zatímco agent a_1 se bude dále přesouvat ke své cílové pozici, agent a_2 bude navždy čekat na pozici (2, 1), protože tato pozice je nejblíž k jeho další pozici. Obrázek 3.6 (vpravo), agent a_1 se přesouvá z pozice (0, 0) na pozici (1, 0) v časovém kroku 0, podle své naplánované cesty. Pokud přiřadíme posun agentovi a_2 před agentem a_3, tak se agent a_2 přesune na pozici (1, 1), aby dodržel formaci. Následně ale nemá agent a_3 kam se přesunout.[1]

3.6.3 A* na spojeném stavu

A* na spojeném stavu je přímočarý algoritmus, který optimálně řeší problém MAPF. Aplikuje algoritmus A* na M-agentní spojený stavový prostor. Stavy přiřadí každému z M agentů různou pozici. Agentu a_i je přiřazena startovní pozice s_i a cílová pozice g_i ve startovním a cílovém stavu. Dále je přiřazen každému agentovi nekolizní posun nebo čekací příkaz. [1]

Adaptujeme A* na spojeném stavu do MAiF tak, že vezmeme v úvahu jak rozpětí, tak odchylku od formace při výpočtu g-hodnoty a přípustné h-hodnoty každého uzlu ve vyhledávacím stromu A*. Pro uzel n v časovém kroku t, hodnota rozpětí g-hodnoty je

$$C_{mg}(n) = t, \tag{3.4}$$

zatímco g-hodnota celkové odchylky od formace je

$$C_{fg}(n) = \sum_{k=0}^{t} \mathcal{F}(k) \tag{3.5}$$

Navrhneme přípustnou h-hodnotu $C_{mh}(n)$ pro rozpětí, jako minimální počet časových kroků potřebných pro všechny agenty k dosažení jejich cílových pozic, který můžeme odhadnout jako maximální vzdálenost z aktuální pozice $\pi_i(t)$ každého agenta a_i k jeho cílové pozici g_i, tzn.

$$C_{mh}(n) = \max_{1 \leq i \leq M} dist(\pi_i(t), g_i), \tag{3.6}$$

kde $dist(u, v)$ je vzdálenost mezi pozicemi u a v v grafu G. Navrhujeme přípustnou h-hodnotu $C_{fh}(n)$ pro celkovou odchylku od formace, jako sumu odchylek od formace po časovém kroku t v nejlepším případě, tedy že všichni agenti se posunou o jeden krok k požadované formaci každý časový krok. Například, když graf G je mřížka se 4 sousedy se stejně velikými políčky, nejlepší případ je, pokud se odchylka od formace zvyšuje o M s každým časovým krokem, tzn.

$$C_{fh}(n) = \sum_{j=1}^{\lfloor \mathcal{F}(t)/M \rfloor} (\mathcal{F}(t) - M \cdot j). \tag{3.7}$$

Proto, jsou dvě f-hodnoty pro každý uzel n, konkrétně rozpětí f-hodnota $C_m(n) = C_{mg}(n) + C_{mh}(n)$ a celková odchylka formace f-hodnota $C_f(n) = C_{fg}(n) + C_{fh}(n)$. [1]

K získání Paretovy fronty tohoto optimalizačního problému použijeme algoritmus ϵ-constraint [24]. Konkrétně nahradíme A* fokálním hledáním [25]. Stejně jako A* používá fokální hledání seznam OPEN, v němž jsou uzly n seřazený podle priorit vzestupně podle jejich hodnot rozpětí f-hodnoty $C_m(n)$. Narozdíl od A*, fokální hledání používá navíc seznam FOCAL, ve kterém jsou všechny uzly ze seznamu OPEN, jejichž rozpětí f-hodnot nejsou vyšší než $\epsilon \min_{n \in OPEN} C_m(n)$, kde $\epsilon \geq 1$ je parametr poskytnutý uživatelem. Fokální seznam dává vyšší prioritu jeho uzlům n podle jejich zvyšující se $f$hodnoty $C_f(n)$ celkového odchýlení od formace. V každé iteraci rozšíří fokální hledání uzel v seznamu FOCAL (ne v seznamu OPEN) minimální f-hodnotou celkového odchýlení od formace. Fokální hledání zaručuje nalezení řešení, jehož rozpětí C_m je nejvýše ϵ krát větší než minimální rozpětí a jehož celková odchylka od formace je nejmenší celková odchylka od formace mezi všemi řešeními se stejným rozpětím C_m. Některé body Paretovy fronty můžeme získat změnou ϵ. [1]

A* na společném stavu je úplný a Pareto optimální. Bohužel díky exponenciálnímu stavovému prostoru je A* na společném stavu extrémně neefektivní. Existuje mnoho technik na zrychlení A* na společném stavu pro MAPF problémy, ale většinu těchto optimalizačních technik nelze použít na problém MAiF. Používáme pouze techniku dekompozice operátorů [26] pro A* na společném stavu. Ale i s touto optimalizací A* na společném stavu se tento algoritmus škáluje špatně při zvyšování počtu agentů.[1]

3.6.4 Roj multi-agentní hledání cest

V této sekci budeme prezentovat náš dvoufázový MAiF algoritmus nazvaný SWARM-MAPF, který využívá myšlenky jak navádění formací, tak MAPF a je úplný a efektivní. Řeší kombinovanou úlohu MAiF rozdělením problému na podproblémy a dvoufázové plánování. V první fázi

je *vedoucí agent* s nejmenším počtem časových kroků k utvoření formace. Ostatní agenti jsou označeni jako *následníci*. Poté je nalezená cesta vedoucího agenta rozdělena na segmenty dvou typů – na segmenty, kde se mohou všichni agenti pohybovat v zadané formaci (tzv, otevřené segmenty), SWARM-MAPF zde používá rojový algoritmus a na segmenty (tzv. zhuštěné segmenty), kde agenti nemohou udržet zadanou formaci (např. v oblastech s překážkami). SWARM-MAPF používá nový hierarchický algoritmus založený na MAPF, nazvaný CBS-M k přesunu agentů okolo překážek jak nejrychleji to jde a zároveň udržení celkové odchylku od formace co nejnižší. CBS-M může být taky použitý jako samostatný MAiF algoritmus, který pouze minimalizuje rozpětí, ale upřednostňuje řešení s nejmenší odchylkou od formace. SWARM-MAPF negarantuje optimální řešení pro ani jeden z cílů, ale je experimentálně demonstrováno [1], že produkuje řešení, která udržují agenty blízko zadané formace s pouze malou ztrátou optimality rozpětí. [1]

3.6.4.1 1. fáze: Hledání cesty pro vedoucího agenta

V 1. fázi, SWARM-MAPF vybere vedoucího agenta ze všech agentů a rozdělí svojí cestu do otevřených a zhuštěných segmentů. V každém otevřeném segmentu agenti utvoří zadanou formaci a následující vedoucího agenta, zatímco ve zhuštěných segmentech jsou jejich cesty plánované v 2. fázi.[1]

3.6.4.1.1 Výběr vůdčího agenta a jeho cesty *Formaci-blokující* pozice je taková, že dokud na ní agent je, nemůže být dodržena zadaná formace zbytkem agentů. Celková *formaci-blokující hodnota* cesty je počet formaci-blokujících pozic na ní. V 1. fázi, SWARM-MAPF vybere vedoucího agenta a^*, jehož cesta má nejnižší formaci-blokující hodnotu ze všech cest agentů, ne delších než wB, kde $w \geq 1$ je parametr zadaný uživatelem a B je spodní hranice na rozpětí. V této publikaci SWARM-MAPF používá $B = \max_{1 \leq i \leq M} dist(s_i, g_i)$. [1]

Aby toto dokázal, SWARM-MAPF udělá best-first hledání pro každého agenta a_i, aby našel cestu π_i s nejmenší formaci-blokující hodnotou s omezením,že cesta není delší než wB. Remízy řeší preferencí kratších cest. Poté SWARM-MAPF vybere vedoucího agenta a^* a jeho cestu π^* s nejmenší formaci-blokující hodnotou. Remízy řeší preferencí kratších cest.[1]

3.6.4.1.2 Rozdělení cesty vedoucího agenta do segmentů Jakmile je zvolen vedoucí agent a^*, jeho cesta π^* je rozdělena na alternující *otevřené* a *zhuštěné* segmenty. Každý otevřený segment je maximální segment $[\pi^*(t_b), \pi^*(t_e)](t_b \leq t_e)$ takový, že pro každé $t_b \leq t \leq t_e$, pozice $\pi^*(t)$ není formaci-blokující. Všechny ostatní segmenty jsou zhuštěné.[1]

Předpokládejme, že existuje K otevřených segmentů. K musí být alespoň 1, protože cílová pozice vedoucího agenta není formaci-blokující. Nechť $p_1^*, \ldots, p_{2K}^*$ jsou první a poslední pozice všech otevřených segmentů, tzn. k-tý otevřený segment je $[p_{2k-1}^*, p_{2k}^*]$. Nechť p_0^* je počáteční pozice vedoucího agenta. p_0^* a p_1^* jsou identické právě tehdy, když počáteční (startovací) pozice všech agentů jsou v zadané formaci. Existuje také K zhuštěných segmentů a k-tý zhuštěný segment je $[p_{2k-2}^*, p_{2k-1}^*]$.[1]

Nechť l_0^* je M-skupina seřazených počátečních pozic všech agentů a $l_k^*(1 \leq k \leq 2K)$ je M-skupina seřazených pozic všech agentů, která utváří zadanou formaci okolo pozice p_k^* vedoucího agenta. Každý otevřený segment specifikuje dílčí MAiF instanci, kde se všichni agenty z pozic l_{2k-1}^* přesunou na pozice l_{2k}^*. SWARM-MAPF získá dílčí řešení pro každou takovou dílčí instanci MAiF zdarma, protože všichni agenti se pohybují v zadané formaci podél segmentu cesty $[p_{2k-1}^*, p_{2k}^*]$ vedoucího agenta. V každém zhuštěném segmentu $[p_{2k-2}^*, p_{2k-1}^*]$ jakákoliv pouice p_{2k-2}^* a p_{2k-1}^* je formaci-blokující. Každý zhuštěný segment specifikuje dílčí MAiF instanci, kde se všichni agenty musejí přesunout z *průběžných počátečních pozic* l_{2k-2}^* na *průběžné cílové pozice* l_{2k-1}^*. SWARM-MAPF získá dílčí řešení pro každou takovou dílčí MAiF intanci v 2. fázi. Konečně, SWARM-MAPF napojí jednotlivá dílčí řešení pro všechny dílčí MAiF instance obou typů segmentů pro získání řešení celé MAiF instance problému.[1]

Obrázek 3.5 (pravá část) demonstruje generaci cesty vedoucího agenta pro tento příklad. SWARM-MAPF položí spodní hranici B rozpětí rovnou 5. Nezávisle na uživatelem zadaném parametru w, SWARM-MAPF vždy najde cestu o délce 5 pro agenta a_2 a vybere ho jako vedoucího, protože agent a_2 má cestu délky 5 s celkovou hodnotou odchylky formace rovnou 0. Zelená čára na obrázku je taková cesta. Jsou zde 2 segmenty, konkrétně zhuštěný segment z (průběžných) počátečních pozic $l_0^* = \langle(3,1),(5,1),(4,3)\rangle$ k (průběžným) cílovým pozicím $l_1^* = \langle(4,2),(5,1),(5,0)\rangle$ a otevřený segment z počátečních pozic l_1^* k cílovým pozicím $l_2^* = \langle(0,3),(1,2),(1,1)\rangle$. Řešení pro celou MAiF instanci problému je napojení dílčích řešení jednotlivých MAiF instancí obou segmentů.[1]

3.6.4.2 2. fáze: CBS-M pro zhuštěné části cesty

Ve 2. fázi SWARM-MAPF používá MAPF algoritmus k výpočtu cest pro dílčí instance MAiF pro každý zhuštěný segment, které přesunou agenty z jejich průběžných počátečních pozic na jejich průběžné cílové pozice. Pro snazší představu budeme v popisu 2. fáze označovat s_i a d_i dané průběžné startovní a cílové pozice agenta a_i. Vyvineme novou variantu vyhledávání založeného na konfliktech (CBS) [4] zvanou CBS-M, která minimalizuje rozpětí a motivuje agenty k těsnému dodržování zadané formace.[1]

3.6.4.2.1 CBS Původní CBS [4] je dvouúrovňový MAPF algoritmus, který minimalizuje *dobu toku*, tj. součet délek cest pro všechny agenty. Na vyšší úrovní CBS provádí nejprve prohledávání *stromu omezení* (CT), aby vyřešil kolize mezi agenty. Každý uzel CT N obsahuje sadu časoprostorových omezení ($N.constraints$), cesty všech agentů ($N.paths$), které dodržují $N.constraints$, ale mohou vyústit v kolize a cenu ($N.constraints$), která se rovná době toku všech cest v $N.paths$. CBS vždy rozšíří uzel CT s nejmenší cenou. Kořenový CT uzel nemá žádná omezení a obsahuje nejkratší cesty na grafu G (bez ohledu na další agenty) pro všechny agenty. Když CBS rozšiřuje uzel CT N, zkontroluje, zda jsou cesty v $N.paths$ bezkolizní. Pokud ano, pak je N cílovým uzlem CT a CBS úspěšně skončí. Jinak CBS vybere kolizi, kterou chce vyřešit a vygeneruje dva potomky – CT uzly N_1 a N_2 z N, které dědí $N.constraints$ a $N.paths$. Pokud je kolize, kterou je třeba vyřešit vrcholovou kolizí $\langle a_i, a_j, v, t\rangle$, potom CBS přidá omezení $\langle a_i, v, t\rangle$ do $N_1.constraints$ k omezení agenta a_i, aby nebyl na pozici v v časovém kroku t a obdobně přidá vrcholové omezení $\langle a_j, v, t\rangle$ do $N_2.constraints$. Pokud je kolize hranová $\langle q_i, a_j, u, v, t\rangle$, potom CBS přidá hranové omezení $\langle a_i, u, v, t\rangle$ do $N_1.constraints$ k omezení agenta a_i, aby se nemohl přesunout z pozice u na pozici v v časovém kroku t a obdobně přidá hranové omezení $\langle a_j, v, u, t\rangle$ do $N_2.constraints$.[1]

Pro každý uzel CT potomka, řekněme N_1, CBS provede nízkoúrovňové A* vyhledávání pro výpočet nové nejmenší cesty pro agenta a_i. Každý nízko-úrovňový uzel (ve vyhledávacím stromě A*) obsahuje pozici v v časovém kroku t. A* hledání určí, kdy rozšířený uzel splňuje následující dvě podmínky: (a) jeho pozice je pozice g_i a (b) jeho časový krok je vyšší nebo stejný jako poslední časový krok kteréhokoliv omezení relevantního k agentovi a_i (pro zajištění, aby agent a_i mohl na své cílové pozici čekat do nekonečna).[1]

3.6.4.2.2 CBS-M CBS-M je podobné CBS, akorát že jeho cílem je minimalizovat rozpětí. Obecně existuje mnoho kombinací cest pro všechny agenty, které mají stejně rozpětí, CBS-M využívá tuto skutečnost pro zrychlení obou vysoko i nízko úrovňových hledání, zatímco se snaží udržet zadanou formaci.[1]

Na vysoké úrovni CBS-M používá rozpětí cest v $N.paths$ jako cenu $N.cost$ CT uzlu N. Každý uzel CT N si také udržuje celkovou odchylku od formace těchto cest. Vyhledávání nejvyšší úrovně vždy rozšiřuje CT uzel s nejnižší cenou. Primární kriterium pro vyřešení remíz upřednostňuje CT uzel N s nejmenším počtem dvojic kolizních cest v $N.paths$, což bylo empiricky dokázáno, zvyšuje rychlost vysokoúrovňového hledání [27]. Sekundární kriterium upřednostňuje uzel CT s nejmenší celkovou odchylkou od formace, což vede agenty k těsnějšímu udržování zadané formace.[1]

Algorithm 3 CBS-M [1]

Vstup: MAPF instance
$Root.constraints = 0$
$Root.solution$ = najdi jednotlivé cesty pomocí low_level_CBS()
$Root.cost = makespan(Root.paths)$ ▷ použití rozpětí cest pro ohodnocení uzlů
Vlož $Root$ do OPEN
while OPEN *not empty* **do**
  $P \leftarrow$ nejlepší uzel z OPEN ▷ řešení s nejnižší cenou
  Validuj cesty P dokud nenastane konflikt.
  if P nemá konflikt **then**
    return $P.solution$ ▷ P je cílové řešení
  end if
  $C \leftarrow$ první konflikt a_i, a_j, v, t v P
  for each agent a_i v C **do**
    $A \leftarrow$ nový uzel
    $A.constraints \leftarrow P.constraints+(a_i, v, t)$
    $A.solution \leftarrow P.solution$
    $A.solution$ je aktualizováno volaním low_level_CBS(a_i)
    $Root.cost = makespan(Root.paths)$
    if $A.cost <$ inf **then** ▷ řešení nalezeno
      Vlož A do OPEN
    end if
  end for
end while

Když je z rodičovského uzlu CT N vygenerován potomek uzel CT N_1 s novým omezením pro agenta a_i, CBS-M provede nízkoúrovňové vyhledávání s cílem vypočítat cestu pro agenta a_i, která splňuje omezení v $N_1.contrainsts$ relevantní pro agenta a_i a těsně se drží zadané formace. Na nízké úrovní používá CBS-M k nalezení cest pro agenty fokální vyhledávání, namísto A* vyhledávání, což mu umožňuje najít (ne nejkratší) cesty, které mají nižší výsledné odchylky od formace než nejkratší cesty. Fokální vyhledávání používá OPEN seznam, který dává vyšší prioritu všem nízkoúrovňovým uzlům ve vzestupném pořadí jejich f-hodnot (které odpovídají délkám cest z počáteční pozice s_i agenta a_i do pozice těch nízkoúrovňových uzlů) a FOCAL seznam všech uzlů aktuálně v OPEN seznamu, jejichž f-hodnoty nejsou vyšší než fokální hranice $\max\{N.cost, \min_{n \in OPEN} f(n)\}$. Pro kořenový uzel CT, který nemá žádný rodičovský CT uzel, nízkoúrovňové vyhledávání používá A* algoritmus. Kromě pozice a časového kroku si každý nízkoúrovňový uzel také udržuje sumu odchylek od formace (s ohledem na cesty ostatních agentů v $N_1.paths$) podél cesty z pozice s_i do pozice nízkoúrovňového uzlu. Nízkoúrovňové vyhledávání vždy rozšiřuje nízkoúrovňový uzel n z FOCAL seznamu, jehož cesta má nejméně kolizí s cestami ostatních agentů v $N_1.paths$, což bylo empiricky dokázáno, že zrychluje vysokoúrovňové vyhledávání [27]. Kriterium pro vyřešení remízy upřednostňuje nízkoúrovňový uzel s nejmenší celkovou odchylkou od formace. Ve srovnání s nízkoúrovňových A* vyhledáváním popsaným v sekci 3.6.4.2.1, ukončovací kriterium nízkoúrovňového fokálního vyhledávání má třetí podmínku a to, že časový krok rozšířeného nízkoúrovňového uzlu je alespoň $N.cost$. Tato podmínka je důležitá, protože umožňuje nízkoúrovňového vyhledávání zakomponovat do svého výpočtu kolize a odchylky od formace ve všech časových krocích, včetně časových kroků poté, co agent dosáhne své cílové pozice g_i.[1]

▶ **Tvrzení 3.1.** *Cena $N.cost$ každého CT uzlu N je nanejvýš rozpětí jakéhokoliv řešení, jehož cesty dodržují omezení v $N.constraints$. [1]*

Důkaz. Pomocí indukce dokážeme, že platí tvrzení: $N.cost$ každého CT uzlu N je nejvýše

rozpětí **jakékoliv množiny cest** (s možnými kolizemi) všech agentů, kteří dodržují omezení $N.constraints$. Tvrzení platí, protože jakékoliv řešení, jehož cesty dodržují $N.constraints$, je množina cest všech agentů, kteří dodržují $N.constraints$. Kořenový CT uzel R obsahuje nejkratší cestu pro každého agenta, tudíž tvrzení platí pro R. Předpokládejme, že tvrzení platí pro rodičovský CT uzel všech CT uzlů N'. V porovnání s $N.paths$, CBS-M mění cestu pouze jednoho agenta, řekněme agenta a_i, v $N'.paths$ provedením nízkoúrovňového vyhledávání s omezeními $N'.constraints$.[1]

(1) Pokud je délka cesty vypočítaná nízkoúrovňovým vyhledáváním nanejvýš $N.cost$, potom výsledná $N'.cost$ je nanejvýš $N.cost$, která je nanejvýš rozpětím jakékoliv množiny cest všech agentů, kteří dodržují $N.constraints$, jak bylo řečeno v indukčním předpokladu. Následně je toto rozpětí nanejvýš rozpětím jakékoliv množiny cest všech agentů, kteří dodržují $N'.constraints$, jelikož $N'.constraints$ je nadmnožina $N.constraints$. (2) V opačném případě je délka cesty vypočítána nízkoúrovňovým vyhledáváním větší než $N.cost$ a tato cesta je nejkratší cestou agenta a_i, který dodržuje $N'.constraints$. Výsledná $N'.cost$ je rovna délce nejkratší cesty, která je nanejvýš rozpětím jakékoliv množiny cest všech agentů, kteří dodržují $N'.constraints$. Tudíž, tvrzení platí pro CT uzel N' v obou případech.[1] ◀

Pro zajištění úplnosti, v prvním kroku CBS-M spustí efektivní MAPF algoritmus, který určí řešitelnost dílčí MAiF instance a vrátí horní hranici rozpětí. Pokud je instance řešitelná, CBS-M vždy vrátí řešení. V opačném případě jakmile rozšíří první CT uzel s cenou vyšší, než horní hranice, vrátí informaci, že žádné řešení neexistuje. Důkaz úplnosti CBS-M je stejný jako v [4]. Důkaz optimálnosti CBS-M je také stejný jako v [4], akorát je adaptovaný na minimalizaci rozpětí a používá 3.1. Tudíž platí následující tvrzení:

▶ **Tvrzení 3.2.** *CBS-M je úplné pro všechny instance MAiF a minimalizuje rozpětí.[1]*

3.6.4.3 Iterativní úprava cíle pro úplnost

V 1. fázi, SWARM-MAPF nedokáže určit řešitelnost dílčí MAiF intance, tudíž úplnost SWARM-MAPF vyžaduje další techniku zvanou *iterativní aktualizace cíle*: V 2. fázi, SWARM-MAPF zavolá CBS-M k řešení dílčách MAiF instancí ze zhuštěných segmentů, a to postupně od prvního zhuštěného segmentu po poslední zhuštěný segment. Pokud CBS-M zahlásí, že pro k-tou instanci MAiF neexistuje žádné dílčí řešení s průběžnými počátečními pozicemi l^*_{2k-2} a průběžnými cílovými pozicemi l^*_{2k-1}, potom pozice p^*_{2k-1} je označena jako formaci-blokující. SWARM-MAPF se vrátí do 1. fáze, znovu přerozdělí cestu π^* po pozici p^*_{2k-2} a aktualizuje l^*_k. SWARM-MAPF iterativně volá CBS-M pro řešení nových dílčích MAiF instancí s průběžnými počátečními pozicemi l^*_{2k-2} a aktualizuje průběžné cílové pozice $(l^*_{2k-1})'$, dokud (1) CBS-M nevrátí řešení dílčího problému nebo (2) pozice $(l_{2k-1})'$ jsou cílové pozice a CBS-M hlásí, že žádné řešení dílčího problému neexistuje. V případě (1), SWARM-MAPF si uloží vypočítané řešení dílčího problému z l^*_{2k-2} do $(l^*_{2k-1})'$ a přejde k dalšímu zhuštěnému segmentu z přerozdělené cesty. V případě (2), SWARM-MAPF nahlásí, že žádné řešení celkové MAiF instance neexistuje. [1]

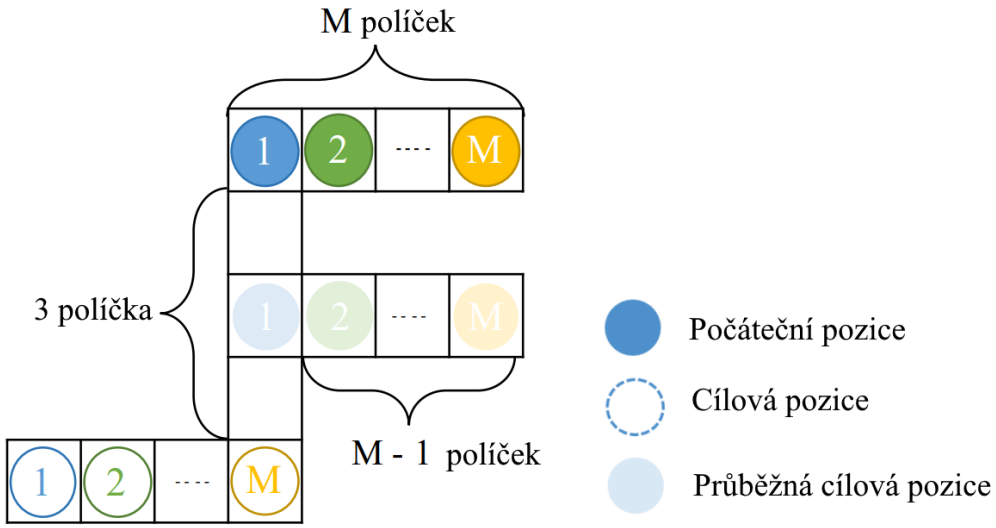
▶ **Tvrzení 3.3.** *SWARM-MAPF s iterativní aktualizací cílů je úplný pro všechny instance MAiF [1]*

Důkaz. Poté co je zavoláno CBS-M na vyřešení dílčí MAiF instance tak buď (1) vrátí řešení, nebo (2) nevrátí řešení a tudíž SWARM-MAPF aktualizuje průběžné cílové pozice a pokusí se vyřešit nově vzniklou MAiF instanci. V obou případech SWARM-MAPF neskončí dokud se nepokusí vyřešit poslední dílčí MAiF instanci vycházející z posledního zhuštěného segmentu, který posune agenty z nějaké průběžné počáteční pozice l^*_{2k-2} na cílové pozice l_g. Tvrdíme, že SWARM-MAPF buď (1) vrátí řešení celé MAiF instance, pokud nějaké existuje nebo (2) správně zahlásí, že řešení neexistuje.[1]

V případě (1) existuje řešení celé MAiF instance, které posune všechny agenty z jejich počátečních pozic l_s do jejich cílových pozic l_g. Nejprve tvrdíme, že zde musí existovat řešení dílčí

MAiF instance plynoucí z posledního zhuštěného segmentu s průběžnými počátečními pozicemi l^*_{2k-2} a cílovými pozicemi l_g. Složením vypočtených dílčích řešení až k průběžným počátečním pozicím l^*_{2k-2} vyústí v bezkolizní cesty, které přesunou všechny agety z pozic l_s na pozice l^*_{2k-2}. Obrácením všech akcí vyprodukuje bezkolizní cesty, které je přesunou z pozic l^*_{2k-2} na pozice l_s. Z předpokladu víme, že existují bezkolizní cesty na přesunutí agentů z pozic l_s na pozice l_g. Tudíž, existuje řešení, které přesune všechny agenty k pozic l^*_{2k-2} na pozice l_g (konkrétně přes pozice l_s). Díky tomu, že je CBS-M úplný, musí vrátit dílčí řešení poslední dílčí MAiF instance. Toto dílčí řešení je připojené ke zbytku všech vypočítaných dílčích řešení a do l^*_{2k-2} nám dá řešení celé MAiF instance.[1]

V případě (2) řešení celé MAiF instance neexistuje. Neexistuje také žádné dílčí řešení k dílčí MAiF instanci vycházející z posledního zhuštěného segmentu s průběžnými počátečními pozicemi l^*_{2k-2} a cílovými pozicemi l_g, jelikož by toto řešení připojené k dosavadním dílčím řešením až k l^*_{2k-2} produkovalo protichůdné řešení celé MAiF instanci. Vzhledem k tomu, že je CBS-M úplný, tak následně SWARM-MAPF musí správně hlásit, že žádné řešení celé MAiF instance neexistuje. [1] ◀



Obrázek 3.7 MAiF instance na mřížce se 4 sousedy[1]

3.6.4.4 Limity Roj multi-agentního hledání cest

Nyní ukážeme, že v extrémních případech SWARM-MAPF vypočítá řešení s velkým rozpětím. Obrázek 3.7 ukazuje příklad, kde M agentů se musí přesunout skrze úzkou vertikální uličku o délce 3 s jednou horizontální uličkou o délce $M-1$. Minimální rozpětí je $M+3$, což vyústí v přesun agentů skrze vertikální uličku jeden po druhém. SWARM-MAPF vybere agenta a_1 jako vedoucího protože jeho pozice m_1 ve prostředku vertikální uličky (naznačeno modrým kruhem označeným „1“ ve světlejším odstínu) není formaci-blokující a všichni ostatní agenty jsou naplánovaní, že obnoví formaci podél horizontální uličky veprostřed obrázku. Tudíž cesta vedoucího agenta je rozdělena na 5 segmentů, jmenovitě otevřený segment $[s_1, s_1]$, zhuštěný segment $[m_1, g_1]$ a otevřený segment $[g_1, g_1]$. Rozpětí dílčích řešení pro všechny otevřené segmenty je 0, zatímco rozpětí pro všechny oba dílčí zhuštěné segmenty je určeno délkou nejkratší cesty agenta a_M z jeho (průběžné) počáteční pozice do jeho (průběžné) cílové pozice, a to jest rozpětí dílčích zhuštěných segmentů je $2M$, respektive $M+1$. Rozpětí řešení celé MAiF instance je tedy $3M+1$. [1]

Nyní zobecníme tento příklad na M agentů přesouvající se skrze vertikální uličku délky $2C+1$ s C horizontálními uličkami, každá o délce $M-1$. Nejmenší rozpětí je $M+2C+1$, což vychází z přesunu agentů skrze vertikální uličky jednu po druhé. SWARM-MAPF si stále vybere agenta a_1 jako vedoucího a rozdělí svou cestu na $C+2$ otevřených segmentů $[m_k, m_k](0 \leq k \leq C+1)$ a $C+1$ zhuštěných segmentů $[m_k, m_{k+1}](0 \leq k \leq C)$, kde m_0 je počáteční pozice s_1 agenta a_1, m_{C+1} je cílová pozice g_1 agenta a_1 a $m_k(1 \leq k \leq C)$ je pozice agenta a_1 v k-té horizontální uličce. Rozpětí dílčích řešení pro všechny otevřené segmenty je 0, zatímco rozpětí dílčích řešení všech zhuštěných segmentů je určeno podlé délky nejkratší cesty agenta a_M z jeho (průběžné) počáteční pozice do jeho (průběžné) cílové pozice, tzn. $2M$ pro každý z prvních $C+1$ zhuštených segmentů a $M+1$ pro poslední zhuštěný segment. Rozpětí řešení celé MAiF instance je tedy $2M(C+1)+M+1$, které je mnohem větší, než minimální rozpětí $M+2C+1$ pro velké C a M.[1]

3.7 Volba implementačních nástrojů

V této sekci odůvodníme použití našich zvolených nástrojů, kterými jsou herní engine Godot a programovací jazyk C++.

3.7.1 Volba herního enginu Godot

Godot je 2D a 3D multiplatformní open source herní engine s licencí MIT vyvinutý komunitou Godot Engine a interně používaný několika společnostmi v Latinské Americe, než byl otevřen a uvolněn pro veřejnost. Vývojové prostředí běží na systémech Windows, OS X a Linux (32 i 64 bitů) a lze v něm vytvářet aplikace určené pro PC, konzole, mobilní a webové platformy. [28]

Cílem Godotu je nabídnout plně integrované prostředí pro vývoj interaktivních prostředí. Umožňuje vývojářům vytvářet prostředí od nuly a nepotřebuje žádné další nástroje kromě těch, které se používají pro tvorbu obsahu (tvorba grafických prostředků, hudby atd.). Architektura návrhu hry je postavena na konceptu stromu vnořených „scén“. Všechny herní zdroje, od skriptů po grafické prostředky, jsou uloženy na disku jako součást souborového systému počítače (nikoli například v databázi). Toto řešení ukládání má usnadnit týmům vývojářů her spolupráci na kontrole verzí kódu skriptů. [29]

Aplikace v Godotu se vytvářejí buď v jazyce C++, nebo pomocí vlastního skriptovacího jazyka GDScript, což je dynamicky typovaný programovací jazyk vysoké úrovně velmi podobný jazyku Python. Narozdíl od Pythonu má GDScript přísné typování proměnných a je optimalizován pro architekturu Godotu založenou na scénách. [28]

Z těchto důvodů jsme se rozhodli pro použití enginu Godot a to konkrétně verze Godot v3.5.1.

3.7.2 Volba programovacího jazyka C++

C++ je vysokoúrovňový, univerzální programovací jazyk, který vytvořil dánský počítačový vědec Bjarne Stroustrup. Poprvé byl vydán v roce 1985 jako rozšíření programovacího jazyka C a od té doby se výrazně rozšířil. Moderní C++ má v současnosti kromě prostředků pro nízkoúrovňovou manipulaci s pamětí také objektově orientované, generické a funkcionální funkce. Téměř vždy je implementován jako kompilovaný jazyk a mnoho dodavatelů poskytuje kompilátory C++. [30]

Jazyk C++ byl navržen s ohledem na systémové programování, vestavěný software s omezenými zdroji a pro rozsáhlé systémy, přičemž jeho hlavními rysy jsou výkonnost, efektivita a flexibilita použití. Jazyk C++ byl shledán užitečným i v mnoha dalších kontextech, přičemž jeho hlavními přednostmi jsou softwarová infrastruktura a aplikace s omezenými zdroji, včetně desktopových aplikací, videoher, serverů a aplikací kritických z hlediska výkonu. [31] Proto jsme shledali, že implementace výpočtově náročnějších částí našeho prostředí lze dobře udělat v jazyce C++.

C++ a Godot mají též velmi dobře zdokumentovanou vzájemnou provázatelnost. Z těchto důvodů jsme si tuto kombinaci zvolili.

Kapitola 4

Návrh prostředí pro testování

V této kapitole nejprve shrneme důvody pro vytvoření interaktviního prostředí. Dále rozepíšeme požadavky, které na toto prostředí budeme mít. V následujících sekcích navrhneme takové prostředí, které splňuje vytyčené požadavky.

4.1 Důvody pro vytvoření prostředí

Hlavní účel interaktivního prostředí (dále aplikace) je vizualizovat uživatelem zadaný problém ve scénáři. Dále také možnost ho upravovat a přizpůsobovat požadavkům. Zároveň je účelem aplikace vizualizovat průběh těchto simulací a reakci agentů na dynamicky se měnící prostředí. Na závěr aplikace testuje řešení uživatelem zadaného scénáře vygenerovaného zvoleným algoritmem hledání cest a sepíše výsledky těchto testů.

Aplikace je nástrojem pro vizualizaci MAPF, MAiF a dalších problémů. Po přečtení dat scénáře je vizualizuje, následně uživatel zadá problém, zvolený algoritmus pro hledání cest ho vyřeší, aplikace toto řešení simuluje, následně ho otestuje a výsledky testů předá uživateli.

4.1.1 Porovnání s již existujícími řešeními

GraphRec [32] je vizualizační nástroj zaměřený na animaci pohybujících se entit na obecném grafu střední velikosti obsahujícím desítky až stovky uzlů. Tento nástroj poskytuje animační engine pro pohyb entit spolu s funkcemi určenými pro podporu sledování časové linie řešení. Graf je třeba vložit na obrazovku ještě předtím, než dojde k vizualizaci. Nástroj navíc dokáže animaci zachytit do různých obrazových nebo video formátů. Vizualizuje jsou pouze pro diskrétní prostředí.

ContinuousViz [33] je vizualizační nástroj, jehož cílem je ohodnotit zadaný MAPF-R problém a najít problémy, které mohou být odhaleny pouze vizualizací řešení. Přečtení dat řešení MAPF-R problému vygeneruje animaci řešení, čímž poskytne uživateli lepší porozumění řešení a poskytne lepší podmínky pro další analýzu. Jeho hlavní účelem je animace a grafické uživatelské rozhraní.

MAPF Visualizer [34] (dále Visualizer) je online nástroj pro vizualizaci multi-agentního hledání cest v mřížce. Nástroj dokáže vizualizovat MAPF problémy a průběh jejich řešení. Dále umožňuje uživateli editovat mapu, přidávat agenty, zadávat jejich cílové pozice a řešit i takto zadané problémy. Umožňuje také importování mapy ze souboru. Poté co nástroj nalezne řešení, ho vizualizuje pomocí animace agentů. Tento nástroj umožňuje i výběr algoritmu pro vyřešení problému zadaného uživatelem.

Porovnání vlastností řešení				
Vlastnost	GraphRec	ContinuousViz	Visualizer	Naše řešení
Vizualizace MAPF	A	A	A	A
Vizualizace MAPF-R	N	N	N	N
Vizualizace MAiF	A	N	N	N
Testování MAPF	A	A	A	A
Testování MAPF-R	N	A	N	N
Testování MAiF	N	N	N	A
Načítání scénářů ze souborů	A	A	A	A
Úprava scénářů v aplikaci	N	N	A	A
Dynamická změna problému	N	N	N	A
Export scénářů do souboru	N	N	N	A
Zadání problému a vyřešení v aplikaci	N	N	A	A
Změny algoritmu hledání cest	N	N	A	A
Přidání algoritmu hledání cest	N	N	N	A

4.1.2 Důvody pro vytvoření našeho prostředí

Předchozí implementace neumožňují dynamické změny v mapě a zadání pro agenty. Dále také neumožňuje zadávání problému udržování formací v multi-agentním hledání cest. Kvůli tomu jsme

se rozhodli vytvořit interaktivní prostředí, které tyto požadavky dokáže naplnit. Naše prostředí umožňuje zadaný problém po nahrání upravovat, jak před spuštěním simulace, tak během ní.

4.2 Jmenná konvence

- Aplikace - celé interaktivní prostředí.
- Mapa - mřížka reprezentující graf, po kterém se agenti pohybují.
- Terén - jedno políčko v mapě.
- Scénář - mapa a agenti dohromady.
- Zadaný problém - uživatelem zadané počáteční a cílové pozice agentů ve scénáři.
- Simulace - obsahuje zadaný problém, řešení vygenerované algoritmem a metadata o simulaci.
- Aplikační módy / módy aplikace - editační a simulační mód.
- Editační mód - mód, ve kterém probíhají úpravy scénáře a celkové nastavení následných simulací.
- Simulační mód - mód, ve kterém uživatel zadává problémy v rámci scénáře a aplikace z nich vytváří simulace.

4.3 Požadavky na prostředí

V této sekci rozepíšeme jednotlivé požadavky na Aplikaci. Konkrétní požadavky jsou označeny **[PX]**, kde **X** je číslo požadavku. Některé požadavky dále rozepisujeme do dílčích částí označených **[PX.Y]**, kde **Y** je číslo dílčí části požadavku **X**.

4.3.1 [P1] Vizualizace scénářů a ovládacích prvků aplikace

Aplikace bude vizualizovat a animovat problémy zadané uživatelem nebo scénáře načtené ze souboru. Zvolíme vhodnou jednoduchou a přehlednou grafickou reprezentaci ovládacích prvků aplikace, jednotlivých polí terénu, agentů, jejich startovních a cílových pozic a průběhu simulace.

4.3.2 [P2] Úprava problémů

Aplikace bude umožňovat změny a úpravy scénářů pomocí nástrojů. Volba nástrojů na úpravu bude součástí ovládacích prvků aplikace, bude vizualizována a bude možno je vybírat myší. Všechny tyto úpravy budou možné za běhu simulace a simulace se jim přizpůsobí.

4.3.2.1 [P2.1] Přidání a odebrání agentů

Mezi úpravy scénářů bude patřit přidávaní a odebírání agentů.

4.3.2.2 [P2.2] Změny v mapě

Úprava terénu/mapy bude součástí aplikace. Budou dostupné různé druhy terénu s různou průchodností (kterou bude možné následně využít v rámci algoritmy pro hledání cest). Dále bude možné změnit rozměry mapy, tím ji zvětšit nebo zmenšit, a tudíž přidat nebo odebrat kousky terénu.

4.3.2.3 [P2.3] Adaptace simulace na změny scénáře

Veškeré změny, které uživatel provede, ať už na mapě nebo mezi agenty, budou komunikovány zvolenému algoritmu pro vyhledávání cest. Simulace se přepočítá a přizpůsobí novému scénáři.

4.3.2.4 [P3] Změna algoritmu pro hledání cest

Uživatel si bude moci zvolit, který algoritmus pro hledání cest bude použit při hledání řešení zadaného problému.

4.3.3 [P4] Ovládání prostředí

Aplikaci bude možné ovládat primárně myší. Funkce budou zvolitelné kliknutím myší na tlačítka. Alternativně budou některé funkce dostupné i klávesovými zkratkami.

4.3.4 [P5] Načítání scénářů

Scénáře bude možné načítat ze souboru. Soubor bude mít formát JSON.

4.3.5 [P6] Export scénářů

Scénáře bude možné exportovat do uživatelem zvoleného souboru. Soubor bude mít formát JSON.

4.3.6 [P7] Testovací schopnosti a metody

Aplikace bude testovat u řešení zadaných problémů:

- délku trvání výpočtu řešení
- odchylky od formace v jednotlivých časových krocích
- celkovou odchylku od formace

4.3.7 [P8] Export výsledků testování

Výsledky testování simulace budou exportovány spolu se seznamem agentů a jejich cestami do souboru. Soubor bude mít formát JSON.

4.4 Návrh

V této sekci představíme návrh aplikace na splnění požadavků z minulé sekce.

4.4.1 Vizualizace scénářů

Základem vizualizace bude zobrazení scénářů. Scénář bude možné načíst ze souboru a následně upravit pomocí nástrojů nebo bude také možné začít „na zelené louce“ a vytvořit celý scénář v aplikaci.

Scénář se skládá z mapy, složené z jednotlivých políček neboli terénů. Bude k dispozici mnoho druhů terénu, každý individuálně ohodnocený váhou, kterou bude možné využít v algoritmech pro hledání cest. Mezi terény bude alespoň jeden druh terénu, který bude mít absolutní průchodnost, tedy váhu 1 a terén, který nebude průchodný vůbec, tedy s váhou 0.

Mapa bude zobrazena jako mřížka terénů, ze kterých bude možné poznat, co za druh terénu to je. Pro jednoduchost prohlížení mapy ji bude možné přibližovat a oddalovat pomocí myši a také se po ní pohybovat, a to jak pomocí myši, tak pomocí mačkání kláves.

Součástí scénáře budou také jednotliví agenti. Agenti budou vizualizováni pomocí jednoznačných textur tak, aby bylo zřejmé, na jakém terénu aktuálně stojí.

Při úpravě scénáře bude ošetřeno, aby uživatel nemohl „položit“ políčko terénu mimo mapu a aby nebylo možné umístit agenta na neprůchodné políčko.

4.4.2 Vizualizace simulace

Simulace, neboli spustitelná instance scénáře a zadaného problému, bude vizualizována animací agentů mezi jejich individuálními pozicemi, počínaje jejich startovními pozicemi, až do cílových pozic, na jejich cestě v jednotlivých časových krocích. Tato animace bude probíhat pouze pokud bude aplikace v simulačním módu a bude spuštěná. Jakmile agent dojde do své finální pozice, tak se zastaví, přepne se do nečinného stavu *idle* a čeká na další příkazy. Při pozastavení simulace se pozastaví i animace agentů na jejich nejbližší následné pozici.

4.4.3 Vizualizace uživatelského rozhraní

Uživatelské rozhraní bude rozděleno na vrchní a spodní panel. Na každém z nich budou různé ovládací prvky. Na vrchním panelu budou tlačítka, která budou přepínat módy aplikce mezi editačním a simulačním, dále tam budou tlačítka na spuštění nebo pozastavení simulace a také další tlačítka, která budou spouštět panely pro další nastavení a manipulaci se scénářem nebo návod na ovládání aplikace.

HELP tlačítko zobrazí panel s nápovědou ohledně ovládání a funkčnosti aplikace. Toto tlačítko bude dostupné kdykoliv. Alternativně bude možné zmáčknout klávesu pro zobrazení či zavření panelu nápovědy. Panel nápovědy bude obsahovat shrnutí všech ovládacích prvků aplikace a krátké vysvětlení každého z nich, u kterých je to relevantní bude obsahovat i obrázek.

Tlačítka *EDIT* a *SIM* přepínají mezi simulačním a editačním módem aplikace. Alternativně bude možné přepínat pomocí klávesy. Při přepnutí módů se bude měnit i vizualizace uživatelského rozhraní. V simulačním módu budou na vrchním panelu dostupná tlačítka *HELP*, *EDIT*, *PLAY*, *PAUSE* a na spodním budou zobrazeni označení agenti. V editačním módu budou na vrchním panelu tlačítka *HELP*, *SIM*, *NEW*, *IMPORT*, *EXPORT*, *RESIZE* a *ALGO*.

PLAY a *PAUSE* tlačítko budou sloužit ke spouštění a pozastavení simulace. Během spuštěné simulace nebude možné kliknout na tlačítko *PLAY* a během pozastavené simulace nebude možné kliknout na tlačítko *PAUSE*. Alternativně půjde přepínat běh simulace pomocí klávesy.

Jak už název napovídá bude tlačítko *NEW* sloužit pro vytvoření nové mapy o velikost zadané uživatelem.

Tlačítko *IMPORT* otevře panel, ve kterém bude moci uživatel zadat cestu ke scénáři, který chce načíst ze souborového systému.

Po kliknutí na tlačítko *EXPORT* se otevře panel, ve kterém bude moci uživatel zadat cestu, kam se má uložit aktuální scénář.

Pro změnu velikosti aktuálního scénáře slouží panel otevřený pomocí tlačítka *RESIZE*.

Pokud bude chtít uživatel změnit algoritmus, který se používá na hledání cest, bude tak moci udělat v panelu, který se otevře tlačítkem *ALGO*

Na spodní panelu budou dostupné různé informace a ovládací prvky podle toho, v jakém módu se aplikace nachází. V simulačním módu zde budou zobrazení právě označení agenti. V editačním módu zde budou na výběr všechny dostupné terény a agenti. Při kliknutí na ně se spustí vizualizace jejich pokládání.

Při pokládání nových terénů nebo agentů bude vždy na místo, kam ukáže uživatel myší, zobrazen nový prvek a bude na něm poznat, zdali je možné ho sem položit, či ne.

Součástí vizualizace prostředí bude i možnost zobrazovat zprávy uživateli, například proč nelze na daném místě položit agenta nebo terén.

4.4.4 Nástroje pro úpravu problémů

V editačním módu aplikace ve spodním panelu budou zobrazeny všechny dostupné druhy terénu, které je možné zvolit kliknutím na ně a následně položit kam uživatel chce. Terén bude zbarven červeně, pokud ho na dané místo není možné položit.

Obdobně bude fungovat pokládání agentů. Na spodním panelu bude možnost zvolit pokládání agentů a poté stejně jako u terénu bude zobrazena silueta, kam by při kliknutí byl agent položen. Silueta bude zbarvena červeně, pokud na dané místo na mapě není možné agenta položit.

Změny v podobě počtu agentů a změny terénu budou dynamicky zpracovány aplikací a simulace problémů budou přepočítány podle změn. Případné úpravy scénáře, které zapříčiní příliš velkém změny v simulacích bude znamenat, že výsledek simulace jíž nebude relevantní pro testování a tudíž na ně nebudou provedené testy.

4.4.5 Načítání a ukládání scénářů

Načítání a ukládání scénářů bude dostupné skrze panely, které se otevřou po kliknutí na tlačítka popsaná v předchozí sekci 4.4.3. Panel bude možné otevřít pouze v editačním módu aplikace.

V rámci *IMPORT* panelu bude moci uživatel zadat cestu k souboru, ze kterého chce načíst scénář. Po zadání cesty a odsouhlasení se aplikace pokusí ze zadané cesty načíst scénář. Aplikace zkontroluje, zdali je scénář ve správném formátu, načte ze souboru data scénáře, zobrazí je uživateli a aktualizuje potřebná data pro další simulace.

V rámci *EXPORT* panelu bude moci uživatel zadat cestu na místo v souborovém systému, kam se má exportovat aktuální scénář. Aplikace převede aktuální mapu a pozice agentů do formátu pro uložení do souboru a následně je do něj uloží.

Scénáře jsou uložené jako soubory formátu JSON pro možnost jednoduché úpravy i mimo aplikaci.

4.4.6 Simulační mód

Jeden ze dvou módů, ve kterých bude aplikace fungovat, je simulační mód. V simulačním módu budou probíhat simulace zadaných problémů a jejich následné testování. Pro přepnutí do simulačního módu z editačního bude moci uživatel kliknout na tlačítko, viz sekce 4.4.3, nebo zmáčknout klávesu.

V simulačním módu může uživatel zadávat problémy v aktuálním scénáři následujícím způsobem: označí agenty, které chce použít v problému, zvolí cílové pozice označených agentů, spustí simulaci.

Nejprve musí označit agenty, které chce aby byli součástí zadaného problému. Označit agenty bude moci pomocí myši. První způsob je kliknutím na agenta, kterého chce označit. Agent se následně přidá mezi označené. Pro zrušení označení agenta na něj stačí opět kliknout. Druhý způsob bude kliknutí a potažení myší přes všechny agenty, které chce uživatel označit. Toto označí agenty v zadaném obdélníku a zruší označení všech ostatních. Označení agenti budou zobrazeni jako tlačítka ve spodním panelu. Při kliknutí na tlačítko s agentem, se zruší jeho označení.

Formace agentů je dána jejich počátečními pozicemi. Tudíž jejich cílové pozice budou určeny podle ní.

Výběr cílových pozic probíhá opět myší. Aplikace zobrazí cílové pozice agentů okolo myši. Na cílových pozicích bude poznat, zdali na daný terén může agent vstoupit.

Po odsouhlasení cílových pozic agentů se aplikace pokusí pomocí zadaného algoritmu pro hledání cest nalézt cesty pro označené agenty z jejich počátečních pozic do jejich cílových pozic. Pokud se jí to nepovede, zobrazí se na obrazovce hlášení o neúspěchu. Pokud se jí to povede a nalezne cesty do cílových pozic, tak po spuštění simulace tlačítkem nebo klávesou, aplikace začne animovat agenty po jejich cestách. Simulace se spustí nebo pozastaví tlačítky nebo klávesami, viz sekce 4.4.3.

4.4.7 Editační mód

Druhý mód, ve kterém bude aplikace fungovat, je editační mód, ve kterém bude moci uživatel načítat scénáře ze souborového systému, ukládat aktuální scénář do jím zvoleného souboru 4.4.5, vytvořit úplně nový scénář „na zelené louce“, změnit velikost mapy aktuálního scénáře, změnit algoritmus použitý pro hledání cest pro zadané problémy a také upravovat scénář změnou konkrétních terénů v mapě nebo přidáním či odebíráním agentů. Pro přepnutí do editačního módu ze simulačního bude moci uživatel kliknout na tlačítko, viz sekce 4.4.3, nebo zmáčknout klávesu.

Vytvoření nového scénáře smaže aktuální scénář a vytvoří novou mapu bez agentů, o výšce a šířce zadané uživatelem do příslušného panelu. Mapa bude vyplněna výchozím terénuem s plnou průchodností, váhou 1.

Změna velikosti mapy aktuálního scénáře změní rozměry (výšku a šířku) mapy podle uživatelem zadaných nových rozměrů. Všechny terény, které jsou mimo nové rozměry, budou smazány. Všechny nově vzniklé terény budou nastaveny na výchozí terén s plnou průchodností (váhou 1).

Algoritmus pro hledání cest půjde změnit v příslušném panelu. Uživatel dostane na výběr z předem registrovaných algoritmů pro hledání cest. Následně se tento algoritmus bude používat pro všechny další výpočty ve všech dalších simulacích.

Úprava scénáře bude probíhat nejprve zvolením typu terénu nebo agenta ve spodní panelu a následně bude moci uživatel klikat na místa na mapě, kam chce terén nebo agenta umístit. Při pokládání terénu nebo agentů bude aplikace vizualizovat, zdali se na dané místo dá terén nebo agent položit. Kliknutím uživatel potvrdí svou volbu. Pokud se uživatel pokusí položit terén nebo agenta na místo, kam se nedá položit, aplikace zobrazí zprávu proč to nejde.

4.4.8 Rozhraní algoritmů pro testování

Nyní definujeme rozhraní, které budou muset všechny algoritmy pro hledání cest implementovat a dodržovat.

První důležitou součástí rozhraní bude metoda *setup*(), která příjme na svém vstupu data o mapě scénáře. Šířku mapy, výšku mapy a mapu samotnou, která bude reprezentovat jednotlivá pole (terény) mapy. Součástí informací bude i váha jednotlivých terénů na mapě.

Druhou důležitou metodou bude *generate_paths*(), která přijme pozice označených agentů, jejich formaci, vedoucího agenta formace, cílové pozice agentů a pozice ostatních, neoznačených agentů. S těmito informacemi pak bude moci algoritmus pracovat.

Algoritmus bude vracet cesty agentů, informaci o tom, zda bylo hledání úspěšné a čas, jaký algoritmu zabralo vypočítat cesty agentů v mikrosekundách. Tento výstup vrátí metoda *generate_paths*().

4.4.9 Testování řešení

Poté, co doběhne simulace řešení uživatelem zadaného problému, vygenerované zvoleným algoritmem, bude toto řešení otestováno. Testování bude spuštěno až po doběhnutí simulace z důvodu možnosti upravovat scénář během průběhu simulace.

Testovat se bude odchylka od formace. Jak moc se v každém časovém kroku agenti odchylují od zadané formace vzhledem k vedoucímu agentovi a součástí testů bude i doba, kterou algoritmus potřeboval na vygenerování řešení. Tato informace je součástí řešení, které vrátí algoritmus.

4.4.10 Export výsledků testování

Výsledky testů ze sekce 4.4.9 budou po úspěšném doběhnutí simulace, tedy poté co všichni agenti původně přiřazení do simulace dojdou do své cílové pozice, zpracovány a exportovány do souboru pojmenovaném podle interního identifikátoru simulace.

Součástí těchto výsledků budou jak výsledné hodnoty testů řešení, tak další informace o simulaci. Tyto informace se budou skládat z výčtu agentů, výpis jejich kompletních cest, vedoucího agenta, formace, doby jakou trvalo vygenerovat řešení, odchylky od formace v každém časovém kroku a celkové odchylky od formace během celého průběhu simulace.

4.4.11 Komunikační datové struktury

V této sekci popíšeme, pomocí jakých datových struktur bude probíhat komunikace mezi jednotlivými komponentami aplikace. Datové struktury budou obsahovat informaci relevantní pro splnění funkce komponenty nebo budou obsahovat výstupní data pro předání výsledků vygenerovaných komponentami zpět do hlavní části.

4.4.11.1 Datová struktura scénářů pro načítání a ukládání

Při načítání a ukládání nových map budeme využívat strukturu, která v sobě bude nést informace o výšce mapy a šířce mapy v podobě dvou celočíselných hodnot. Dále bude obsahovat řetězec, ve kterém budou uloženy všechny terény, zakódované jejich příslušnými znaky. Struktura bude také obsahovat pozice agentů na mapě uložené jako pole celočíselných souřadnic.

4.4.11.1.1 Zakódování terénů

- . - pro výchozí terén s nejvyšší průchodností s váhou 1, vizualizovaný jako travnatá oblast.
- : - pro terén s nejvyšší průchodností s váhou 1, vizualizovaný jako hliněný plac.
- @ - pro výchozí neprůchodný terén s váhou 0, vizualizovaný jako skála.
- *s* - pro terén sněhu s váhou 3.

- ; - pro terén vizualizovaný jako písek s váhou 4.
- i - pro terén ledu s váhou 6.
- t - pro další neprůchodný terén s váhou 0, vizualizovaný jako pár stromů.
- T - pro další neprůchodný terén s váhou 0, vizualizovaný jako mnoho stromů.

4.4.11.2 Struktura dat pro nastavení algoritmu

Struktura dat pro nastavení algoritmu bude obsahovat následující informace: obdobně jako struktura pro načítání a ukládání scénářů bude obsahovat celočíselné hodnoty výšky a šířky mapy. Na rozdíl od struktury pro načítání a ukládání scénářů nebude tato struktura obsahovat pozice agentů a řetězec terénu bude transformovaný do řetězce obsahující místo znaků terénu jejich váhy, tedy průchodnosti.

4.4.11.3 Struktura dat pro zadání problému

Při zadávání problému aplikace zpracuje data scénáře a uživatelem zadané parametry a naplní následující strukturu. První položka struktury budou počáteční celočíselné souřadnice agentů v mapě, dostupné pod identifikačním jménem agenta. Další položkou bude pole pozic ostatních agentů, dále identifikační jméno vedoucího agenta a formace ve tvaru pozice vzhledem k vedoucímu agentovi, uložené pod identifikačním jménem každého agenta. Jako poslední informace budou součástí datové struktury cílové pozice agentů uložené opět pod identifikačními jmény agentů.

4.4.11.4 Struktura dat pro předání řešení zadaného problému

Poté, co algoritmus ze zadaných parametrů pomocí struktur pro nastavení algoritmu a zadání problému vypočítá řešením, předá ho zpět aplikaci v následujícím formátu. Struktura bude obsahovat binární hodnotu úspěchu v závislosti na tom, zdali byl algoritmus schopný najít cestu pro zadaný problém v zadaném scénáři. Pokud bude algoritmus úspěšný, bude obsahovat také cesty jednotlivých agentů, uložené pod jejich identifikačními jmény. Poslední informace, kterou bude struktura obsahovat, bude celkový čas, jak dlouho trval výpočet cest v mikrosekundách.

4.4.11.5 Datová struktura pro výsledky testování

Struktura dat pro výsledky testování, která je použita po doběhnutí simulace a otestování na zápis do souboru je následující: obsahuje řešení zadaného problému, tedy strukturu viz. sekce 4.4.11.4, dále pole odchylek od formace v jednotlivých časových krocích a celkovou odchylku od formace za celý průběh simulace. V neposlední řadě obsahuje formaci, jakou měli agenti udržovat po celou dobu simulace vzhledem k vedoucímu agentovi a identifikační jméno vedoucího agenta.

Kapitola 5

Programátorská dokumentace implementace řešení

V této kapitole popíšeme celou architekturu, implementaci a kód řešení. Nejprve vysvětlíme co je všechno potřeba na kompilaci a poskládání aplikace do jejího spustitelného stavu a projdeme jednotlivé kroky potřebné pro vytvoření spustitelné aplikace. V dalších sekcích vysvětlíme jednotlivé komponenty řešení a jaká je celková struktura našeho řešení, ve které popíšeme, jak spolu komponenty komunikují a jak spolupracují. Dále také ukážeme uživatelské rozhraní a dopodrobna rozepíšeme jeho funkcionality a ovládání. Součástí této kapitoly bude i podrobná dokumentace části řešení napsané v jazyce C++, která se zaobírá algoritmy pro hledání cest, které jsou následně používány aplikací. Dokumentaci doporučujeme procházet spolu se zdrojových kódem pro lepší pochopení a názornou ukázku, ne pouze text v tomto dokumentu.

5.1 Kompilace řešení

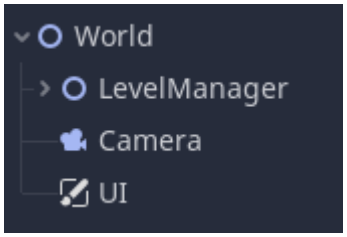
Kompilace řešení má více kroků. Na kompilaci bude zároveň potřeba více nástrojů, zvlášť pro kompilaci části aplikace napsané v jazyce C++, následně propojit vzniklou dynamicky linkovanou knihovnu se zbytkem aplikaci napsané v Godot enginu, prostřednictvím Godot editoru.

Pro zkompilování budeme potřebovat program *SCons* [35], což je open source nástroj pro kompilaci a kompletaci softwaru. SCons potřebuje mít dané instrukce v souboru *SConstruct*. Soubor SConstruct se nachází v hlavní složce aplikace a je připravený ke kompilaci a kompletaci aplikace. Spuštěním *SCons* s přepínačem *platform=windows* se C++ část aplikace zkompiluje a vytvoří dynamicky linkovanou knihovnu ve složce *bin*, připravenou na použití zbytkem aplikace.

Dále je potřeba Godot engine, bakalářská práce je psaná v Godotu v3.5.1, tudíž je nutné mít staženou tuto verzi. Po otevření zdrojových kódů v editoru enginu Godot je možné aplikaci připravit pro vydání navigováním do menu *Project* - *Export...* a zde si vybrat parametry, které chceme, aby naše výsledná aplikace měla.

5.2 Struktura řešení

Struktura celé aplikace je vysoce ovlivněna strukturou, jaká je typická pro herní engine Godot. Celá je rozdělena do scén obsahující skripty a uzly. Tyto scény mezi sebou navzájem interagují, posílají si data a signály o proběhlých událostech.



Obrázek 5.1 Struktura hlavní scény

Hlavní scéna obsahuje řídící skript *World manager* a všechny ostatní komponenty aplikace a má strukturu zobrazenou na obrázku 5.1. Scéna obsahuje komponentu *LevelManager*, která má na starosti průběh veškerých operací a funkcí nad scénáři. Dále obsahuje instanci Godot třídy *Camera2D* jménem *Camera*.

Camera má na sobě navěšený skript *camera_controller.gd*. Tento skript se stará o pohyb kamery v aplikaci. Reaguje na vstupy od uživatele v podobě pohybu myší a mačkání kláves. Při zmáčknutí kláves šipek provede kamera pohyb tím směrem. Pokud uživatel přesune myš poblíž okraje obrazovky, tak se kamera pohne tím směrem. Dále kamera obstarává i funkčnost přibližování a oddalování, kterou může uživatel vyvolat kolečkem myši. Kamera zároveň umí nastavit svoje limity, které slouží k omezení pohybu mimo okraje aktuálního scénáře. Kamera si kromě aktuální pozice, úrovně přiblížení a dalších hodnot udržuje i aktuální globální a lokální pozici myši na obrazovce. Zároveň zprostředkovává informace o zmáčknutí a pouštění tlačítek myši, které propaguje do ostatních komponent, kde se využívají na označovaní agentů a vizualizaci označovacího obdélníku.

5.3 Hlavní scéna (WorldManager)

Skript, a tedy zároveň třída *world_manager.gd*, je komponenta na vrcholu stromu scén a tudíž má nejlepší možnost předávání informací směrem dolů ke kořenům stromu scén. Komponenta reaguje na vstup uživatele v podobě klávesy *TAB* pro změnu módu aplikace, klávesy *mezerník* pro spuštění nebo pozastavení aplikace a kláves *plus* a *minus* pro zvýšení nebo snížení rychlosti průběhu simulací. Vždy po obdržení těchto vstupů od uživatele aktualizuje svůj vnitřní stav a zároveň změny propaguje dále do aplikace pomocí signálů nebo voláním metod ostatních komponent.

5.4 Manažer levelu (LevelManager)

Obrázek 5.2 Struktura komponenty Level manager

Manžer levelu (dále *LevelManager*) je jedná z největších funkčních částí celé aplikace. Mezi její komponenty patří mapa, manažer agentů, silueta pokládaného terénu nebo agenta a *Sprite*,

který ukazuje, na které políčko na mapě právě uživatel míří na myší. Všem těmto komponentám *LevelManger* posílá informace a volá jejich funkce při zpracovávání vstupů od uživatele. Mezi jeho funkcionality, které skript zajišťuje patří nastavování dostupných algoritmů, zpracovávání uživatelem zadaných problémů, vytváření nových simulací, aktualizování běžících simulací, testování simulací po jejich doběhnutí, přepočítávání simulací při změně ve scénáři, zpracování požadavků na vytvoření nové mapy nebo na změnu velikosti aktuální mapy, kontrola možnosti položení agenta nebo terénu a import/export scénářů.

V rámci skriptu *LevelManager* lze přidávat algoritmy pro hledání cest. Nový algoritmus je možné přidat právě do metody *setup_pathfinder_options()*. Tato metoda je volána na začátku, při vytvoření instance *LevelManager*. Metoda přidává jednotlivé algoritmy do struktury. *LevelManager* poté předává informace o algoritmech zbytku aplikace (například uživatelskému rozhraní 5.7.2.4). Pokud *LevelManger* přijde požadavek na změnu algoritmu, využívá k tomu metodu *change_pathfinder()*.

Po potvrzení cílových pozic uživatelem *LevelManager* si od manažera agentů 5.4.2 zjistí aktuálně označené agenty. Vybere mezi nimi vedoucího agenta, podlo toho, který z agentů je nejblíž pozici myši ve scénáři. Podle vedoucího agenta vypočítá formaci označených agentů. Formace je pozice agentů vzhledem k vybranému vedoucímu agentovi. Následně vytvoří strukturu 5.10.3 pro předání informací aktuálnímu algoritmu pro hledání cest, předá informace algoritmu a počká, než algoritmus vrátí řešení. Pokud nebylo řešení nalezeno, pošle komponentě uživatelského rozhraní 5.7 žádost o zobrazení hlášky, že se nepovedlo nalézt řešení pro uživatelem zadaný problém. Pokud ale řešení nalezeno bylo, vytvoří simulaci.

Simulace v sobě obsahuje informace o řešení, seznam agentů, kteří byli původně součástí řešení, seznam agentů, kteří stále jsou součástí řešení, formaci, identifikační jméno vedoucího agenta a informaci, zdali se má simulace po doběhnutí testovat a výsledky exportovat do souboru. Simulace je následně uložena do běžících simulací pod interním identifikačním číslem. Všichni agenti, kteří se zúčastni této nové simulace, jsou odstranění z ostatních simulací, protože se jich již neúčastí.

Během celého běhu aplikace se aktualizují simulace a informace uložené v nich. Aktualizují se zároveň i v případě odebírání agentů nebo změn ve scénáři. Pro každou aktuálně běžící simulaci se provede několik kroků. První krok je kontrola počtu agentů stále se účastnících simulace/řešení. Pokud se již neúčastní žádný agent, je simulace smazána. Druhý krok proběhne, pokud má stále simulace povolené exportování a pokud jsou již všichni agenti v simulaci v nečinném stavu *idle*. Pokud jsou splněné obě podmínky je simulace poslána na testování a následný export do souboru pomocí skriptu *JSON_evaluation_exporter.gd*.

Testování simulací přijme simulaci k testování. Následně u ni pro vede testování v podobě výpočtu odchylek od formace jednotlivých časových krocích simulace. Po dokončení výpočtů je naplněna příslušná struktura 5.10.5 a pomocí skriptu *JSON_evaluation_exporter.gd* je exportována do složky *evaluations* pod identifikačním jménem simulace. Soubor je ve formátu JSON.

LevelManager zpracovává požadavky o smazání agenta. Zde se tomuto požadavku přizpůsobí simulace, samotné smazání agenta provede manažer agentů. Při smazání agenta je potřeba přepočítat všechny simulace, kterých se tato změna týká, to znamená ty simulace, kterých se smazaný agent účastnil. Pro každou simulace, které se toto týká je provedeno několik kroků. Nejprve se simulace zakáže exportování, jelikož se již agent nezúčastní simulace nemá smysl pro ni počítat některé testy (například odchylku od formace) a tudíž se zruší i testování celé simulace. Dalším krokem je aktualizování seznamu agentů, kteří se simulace zúčastní. Pokud se simulace již neúčastní žádný agent, je simulace smazána z bežících simulací.

Při změně scénáře je potřeba přepočítat všechny bežící simulace. Toto se děje posláním nastavujících informaci algoritmu v příslušné struktuře 5.10.2 a následným posláním nového problému k vypočítání. Nový problém je vytvořen z aktuálních pozic agentů, stejné formace, stejného vedoucího agenta a stejných cílových pozic. Po obdržení výsledku nového řešení se aktualizuje řešení v simulaci. Pokud nové řešení nebylo nalezeno je simulace smazána, jelikož již dál nemůže pokračovat. Pokud ale řešení bylo úspěšně nalezeno, následuje zkombinování staré simulace a nové

do aktualizované simulace.

Spojení simulací probíhá v několika krocích. Prvním krokem je zkombinování starého a nového řešení. To probíhá postupným spojováním nových a starých cest agentů. Aktualizovaná cesta začíná starou cestou až do aktuální pozice agenta a poté je na ní navázána nová cesta. Dalším krokem je úprava metadat simulace s použitím nového řešení. Povolení exportu a identifikační číslo simulace bude ze staré simulace. Agenti, agenti účastnící se simulace, identifikační jméno vedoucího agenta a formace bude také ze staré simulace.

V panelu změny velikosti mapy 5.7.2.5 může uživatel změnit velikost mapy. Tento požadavek je poslán do *LevelManageru*, kde se provede samotná změna velikosti. Všechny pole terénu, které jsou mimo nový rozsah, jsou smazány a všechna nově vniklá pole jsou nahrazena výchozím terénem s nejvyšší průchodností (tedy váhou 1).

Vytvoření nové mapy kompletně smaže aktuální scénář. Dále nastaví nové rozměry mapy a vytvoří mapu o zadané velikosti, která se bude skládat z výchozích terénů s nejvyšší průchodností, tedy váhou 1. Takto nově vzniklou mapu může uživatel dál upravovat již standardním způsobem úpravy scénáře 6.2.2.5.

Metody na kontrolu pokládání terénu a agentů jsou využívány v několika případech. Mezi případy patří zobrazování cílových pozic vybraných agentů, pokládání nových terénů do mapy a pokládání nových agentů do scénáře. Kontrola položení terénu kontroluje, jestli je pozice nového terénu na mapě. Kontrola položení nového agenta kontroluje, stejně jako kontrola pro položení terénu, zdali je pozice nového agenta uvnitř rozmezí mapy. Navíc kontroluje, jestli nová pozice agenta nekoliduje s již existujícím agentem, nebo jestli je terén na pozici, na kterou by byl nový agent přidán, průchodný (tedy nemá váhu 0). Pokud kontroly selžou, zobrazí se siluety pokládaných terénů nebo agenta zbarvené červeně.

Jakmile je alespoň jeden agent označený, objeví se u kurzoru silueta cílových pozic označených agentů. Siluety jsou následně zkontrolovány pomocí metod z předchozího odstavce a pokud to nejsou validní cílové pozice, zbarví se červeně.

Importování scénářů probíhá pomocí skriptu *JSON_level_importer.gd*, který načte ze zadané cesty scénář. Na scénáři je provedena validace přímo ve skriptu, a pokud je nalezena nějaká chyba, je zahlášena uživateli. Výsledek je předán v příslušné struktuře 5.10.1 zpět do *LevelManageru*. Exportování scénáře je provedeno naplněním příslušně struktury 5.10.1, která je následně spolu s cestou k souboru odeslána skriptu *JSON_level_exporter.gd*, který scénář zapíše na disk. Importování i exportování pracuje se soubory ve formátu JSON.

5.4.1 Komponenta mapy (Env)

Komponenta mapy (Env) je instance třídy *GridMap* herního enginu Godot. Slouží pro udržování a manipulaci informací o mřížkové mapě, která ve scénáří reprezentuje mapu. Jejími stavebními bloky jsou jednotlivé terény 5.9 uložené v další Godot komponentě *TileSet*. Tuto komponentu, ve které jsou terény, využívá skript pro uživatelské rozhraní k dynamickému vygenerování tlačítek pro volbu terénu. Veškerá naše logika zvýraznění komponenty mapy probíhá ve skriptu komponenty *LevelManager*.

5.4.2 Manažer agentů (AgentManager)

Manažer agentů (dále *AgentManger*) slouží k obsluze agentů. Mezi obsluhu agentů patří jejich vytváření, mazání a předávání informací mezi nimi a zbytkem aplikace. Přijímá signály od instancí scény agenta 5.4.2.1 a zpracovává je. Udržuje si informace o aktuálně označených agentech a tuto informaci na vyžádání komunikuje uživatelskému rozhraní. Dále propaguje informace o změně módu aplikace nebo spuštění či pozastavení simulace. *AgentManager* také komunikuje a přiřazuje nové cesty agentů poté, co je vygenerováno nové řešení.

5.4.2.1 Agent

Scéna *Agent* reprezentuje agenty v naší aplikaci. Skládá se z mnoha částí sloužící k vizualizaci a jednoznačnému označení agenta. Jestli je agent označený nebo ne pomocí, lze poznat podle zeleného obdélníku.

Agent dostane od komponenty *AgentManager* 5.4.2 přiřazenou cestu. Jakmile má agent přiřazenou cestu a simulace je spuštěna, začne se po své cestě pohybovat. Pohyb je realizovaný pomocí komponenty *Timer*, která v závislosti na nastavené rychlosti simulace odpočítává jednotlivé časové kroky simulace. Během každého časového kroku se agent přesune na další pozici cesty. Jakmile dojde na konec, zastaví se a nastaví proměnnou *idle* na hodnotu pravda. Následně čeká na další příkazy.

Agent vysílá signály, když je na něj kliknuto. Označení nebo zrušení označení je realizováno zviditelněním označujícího zeleného obdélníku.

5.4.3 Silueta pro pokládání terénů a agenta

Komponenta z herního enginu Godot *Sprite*, která slouží ke komunikaci mezi aplikací a uživatelem se zobrazí vždy když uživatel pokládá nové terény nebo agenty. Silueta na sebe vezme podobu aktuálně zvoleného terénu nebo agenta. Silueta se zbarví červeně, pokud je nemožné položit terén nebo agenta na zvolené místo. Veškerá naše logika siluety probíhá ve skriptu komponenty *LevelManager*.

5.4.4 Zvýraznění vybraného políčka

Instance Godot komponenty *Sprite*. V aplikaci je používána na zvýraznění políčka na mapě, na které uživatel právě míří myší. Pozice je upravována podle globální pozice myši, převedené na pozici na mapě v *LevelManageru*. Přidali jsme tuto komponentu pro zjednodušení a přehlednost při vybírání políček na mapě, ať už při pokládání terénů a agentů, nebo zadávání problému, nebo pro obecně lepší orientaci uživatele na mapě. Veškerá naše logika zvýraznění vybraného políčka probíhá ve skriptu komponenty *LevelManager*.

5.5 Úprava scénáře

Změna scénáře je možná pouze v editačním módu aplikace. Pro změnu velikosti mapy lze využít příslušný panel 5.7.2.5, spuštěný příslušným tlačítkem na horním panelu 5.7.1.

Pro změnu destiček terénu je potřeba si zvolit nový typ terénu 5.7.3.2 na spodním panelu 5.7.3, následně namířit na políčko, kam chce uživatel položí terén a kliknou levým tlačítkem myši. Toto pošle požadavek o změně terénu a pokud je tato změna možná, provede se. Pokud v tom ovšem něco brání, nebude destička terénu změněna a bude vypsána hláška o tom proč tomu tak nelze.

Pro přidání agenta je potřeba kliknout na tlačítko pro přidání nového agenta 5.7.3.2, namířit na požadované místo a kliknout levým tlačítkem myši. Tato akce pošle požadavek na přidání agenta. Pokud je toto validní pozice pro nového agenta, bude agent přidán do scénáře. Pokud to ovšem něco znemožňuje, nebude agent přidán do scénáře, a bude vypsána hláška, proč se tomu tak nestalo.

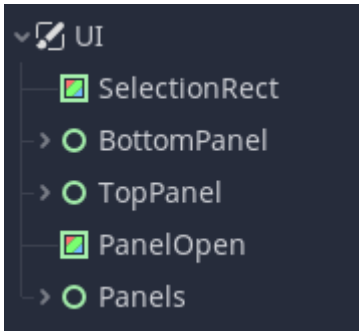
Pro odebrání agenta na něj stačí kliknout pravým tlačítkem myši.¨Pokud chce uživatel zrušit výběr aktuálně pokládaného terénu nebo agenta, zmáčkne pravé tlačítko myši.

5.6 Globálně dostupná data

Součástí aplikace v souboru *AppData.gd* jsou převodníky mezi reprezentacemi terénu. Tyto převodníky jsou dostupné v celé aplikaci, díky tomu, že je třída *AppData* autoload singleton a může je tudíž využívat libovolná scéna. Mezi převodníky je *RAW_MAP_CONVERSION* pro převod mezi textovou reprezentací terénů na mapě při uložení v souboru a identifikačního čísla terénu v mřížkové komponentě 5.4.1, *CELL_CONVERSION* pro převod nazpět z identifikačních čísel terénů v mřížkové mapě do textové formy pro uložení do souboru, převod *PATHFINDING_CONVERSION* na převod z identifikačních čísel terénů v mřížkové mapě na text reprezentující váhy jednotlivých terénů na mapě.

Pro úpravu hodnot vah terénu stačí změnit jejich hodnoty zde v tomto souboru. Je tudíž jednoduché si přizpůsobit terény, ze kterých se skládají mapy scénářů svým potřebám.

5.7 Uživatelské rozhraní (UI)



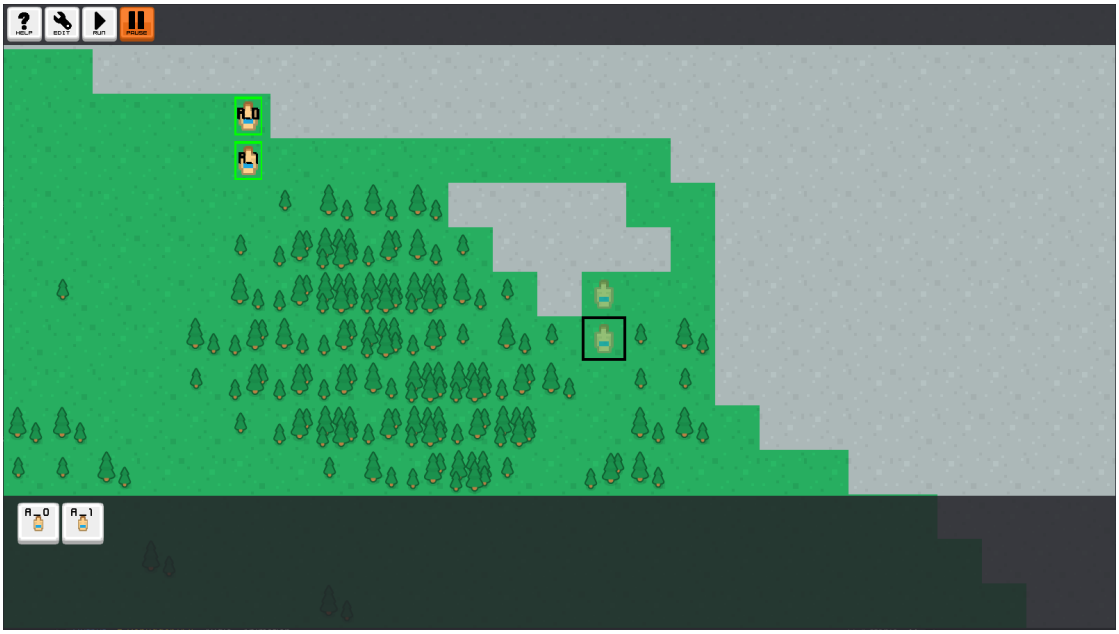
Obrázek 5.3 Struktura komponenty uživatelského rozhraní

Komponenta uživatelského rozhraní je samostatná scéna obsahující mnoho vlastních komponent, v této sekci je popíšeme. Zároveň součástí této scény je skript *ui.gd*, který všechny komponenty spojuje dohromady, dává jim komplexní funkcionality a celkově zprostředkovává vizuální ovládání aplikace. Jednotlivým komponentám je přiřazená skupina (*group*) podle toho, zdali se mají zobrazí v editačním, simulačním, nebo v obou módech. Toto zobrazování a skrývání částí uživatelského rozhraní obstarává samotný skript.

Skript obstarává i generování dynamických částí uživatelského rozhraní. Pokud je aplikace v simulačním módu, přijímá komponenta uživatelského rozhraní od komponenty manažera agentů informace o označených agentech a zobrazuje je pomocí tlačítek 5.7.3.1. Pokud je aplikace v editačním módu, přijme komponenta informace o dostupných terénech a zobrazí je spolu s tlačítkem pro pokládání agentů 5.7.3.2.

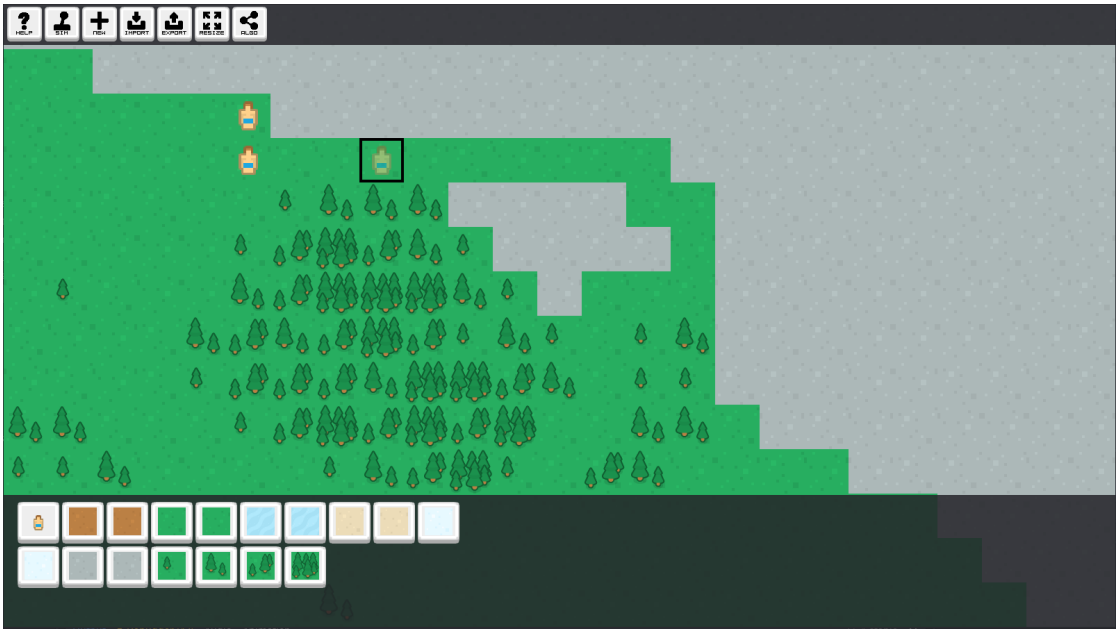
Součástí scény jsou i další komponenty. *SelectionRect*, který slouží k vizualizaci označovacího obdélníku při volbě agentů a *PanelOpen*, který slouží k zvýraznění aktuálně zobrazeného panelu a znemožnění uživateli interagovat s jinými komponentami, než je panel. Obě komponenty jsou skryté, pokud se zrovna nepoužívají.

UI komponenta také zpracovává všechny interakce s panely, tlačítky a dalšími komponentami. Přeposílá žádosti o změny a předává jím informace k zobrazení od zbytku aplikace.



Obrázek 5.4 Ukázka rozhraní v simulačním módu

Součásti uživatelského rozhraní v simulačním módu jsou následující komponenty: v horním panelu tlačítko pro zobrazení nápovědy 5.7.2.1, dále tlačítko pro přepnutí do editačního módu a tlačítka pro spuštění nebo pozastavení simulace viz 5.7.1. V dolním panelu 5.7.3 potom budou již zmíněná vygenerovaná tlačítka pro jednotlivé označené agenty 5.7.3.1. Ukázku takového rozhraní je možné vidět na obrázku 5.4.

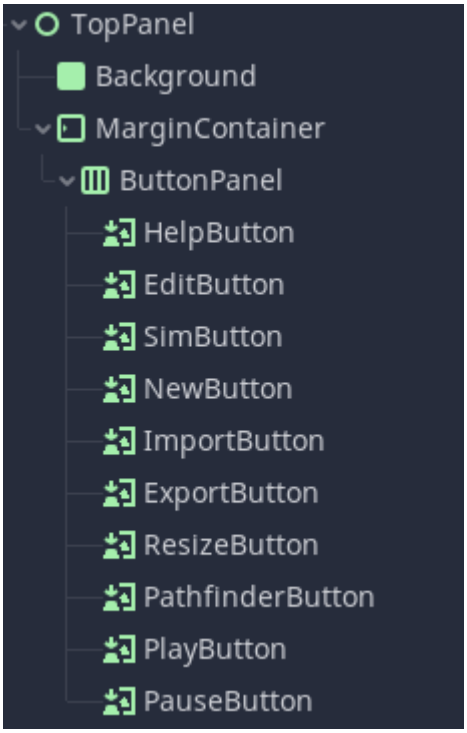


Obrázek 5.5 Ukázka rozhraní v editačním módu

Uživatelské rozhraní v editačním módu obsahuje tyto komponenty. V horním panelu tlačítka pro zobrazení nápovědy 5.7.2.1, dále tlačítko pro přepnutí do simulačního módu, tlačítko pro

vytvoření nové mapy 5.7.2.2, tlačítka pro importování a exportování scénářů 5.7.2.3, tlačítko pro zobrazení panelu změny velikosti mapy scénáře 5.7.2.5 a tlačítko pro změnu algoritmu 5.7.2.4. Na spodním panelu 5.7.3 bude vygenerovaný seznam terénů, které jsou k dispozici, a tlačítko pro pokládání nových agentů 5.7.3.2.

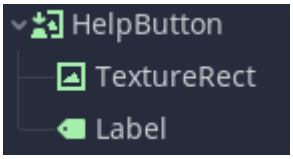
5.7.1 Horní panel



Obrázek 5.6 Struktura komponenty horního panelu

Horní panel se skládá z pozadí, které je pouze pro barvu. Dále obsahuje *ButtonPanel*, ve kterém jsou všechny tlačítka 5.7.1.1. Která tlačítka jsou zrovna viditelná a k dispozici, určuje mód v jakém se právě aplikace nachází.

5.7.1.1 Tlačítka horního panelu



Obrázek 5.7 Struktura tlačítka v horním panelu

Všechna tlačítka v horním panelu jsou složená ze stejných tří částí 5.7. Stlačitelná komponenta *TextureButton*, která zajišťuje funkčnost tlačítka. Dále textura a štítek, které dohromady informují uživatele o tom, co za tlačítko to je a co od něj může očekávat za funkcionalitu.

Obrázek 5.8 Ukázka tlačítka v horním panelu

Každé tlačítko má unikátní texturu a štítek, ukázka takového tlačítka je na obrázku 5.8, funkčnost tlačítek je ale různorodá.

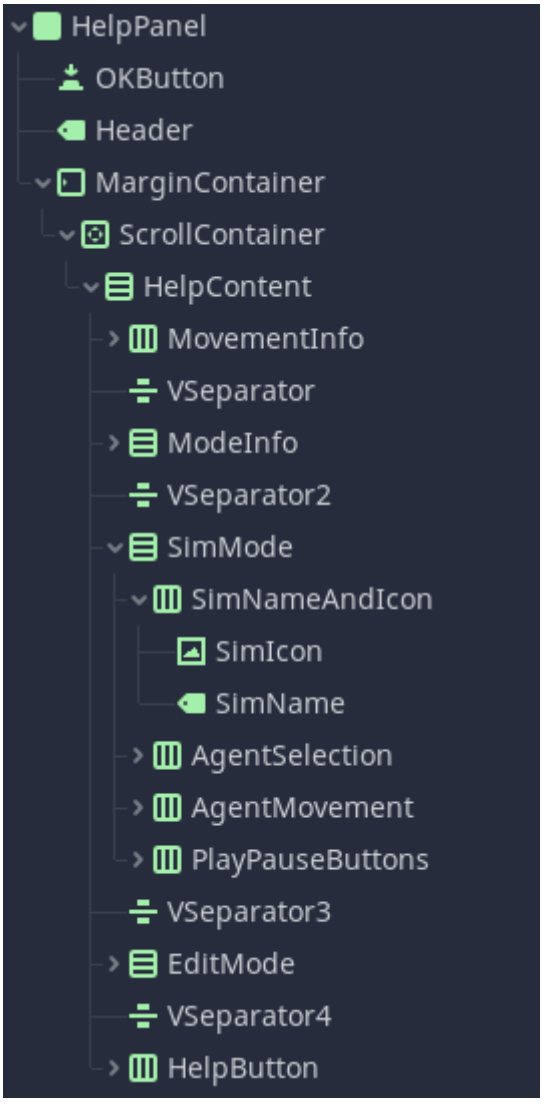
Tlačítko pro zobrazení nápovědy zobrazí panel nápovědy 5.7.2.1. Tlačítko pro přepnutí do simulačního nebo editačního módu přepne aplikaci do zvoleného módu. Tlačítko pro import a tlačítko pro export otevřou panel importování respektive exportování 5.7.2.3. Tlačítko pro změnu velikosti mapy otevře korespondující panel 5.7.2.5. Pro otevření panelu pro změnu algoritmu 5.7.2.4 je možné zmáčknout tlačítko s nápisem *ALGO*. V simulačním módu jsou dostupná tlačítka pro spuštění nebo pozastavení simulace.

Všechna tlačítka mají nastavené posílání signálů do scény UI, kde jsou zpracovány.

5.7.2 Panely

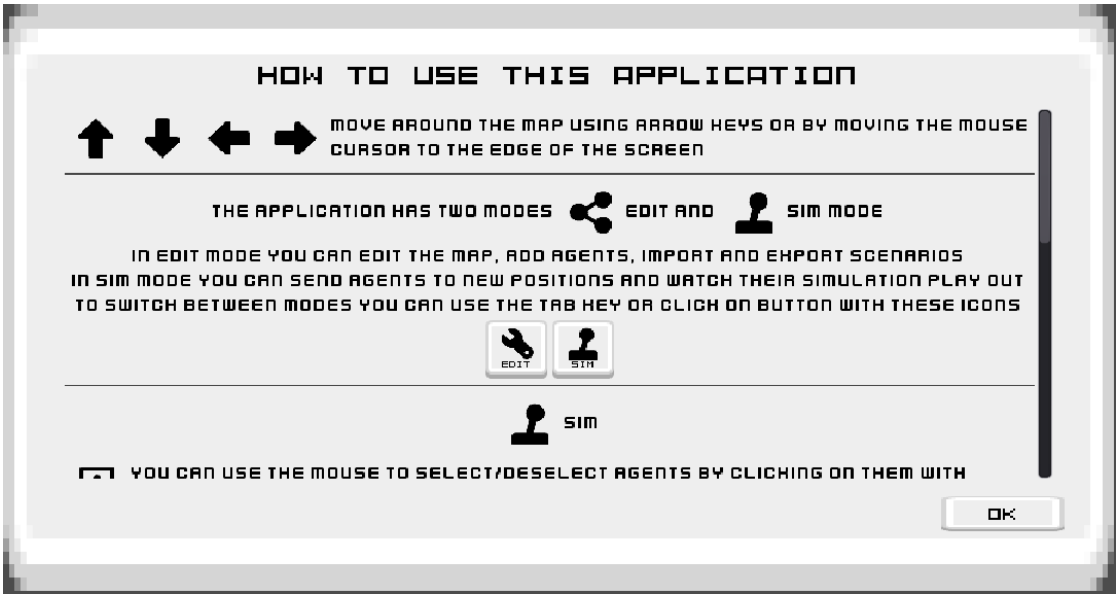
V této sekci popíšeme jednotlivé panely. Všechny panely se při otevření zobrazí na středu obrazovky a budou mít možnost potvrdit změny tlačítkem OK, nebo možnost zrušení změny tlačítkem $CANCEL$, kromě panelu nápovědy, ta má pouze tlačítko potvrzení.

5.7.2.1 Panel nápovědy



Obrázek 5.9 Struktura panelu nápovědy

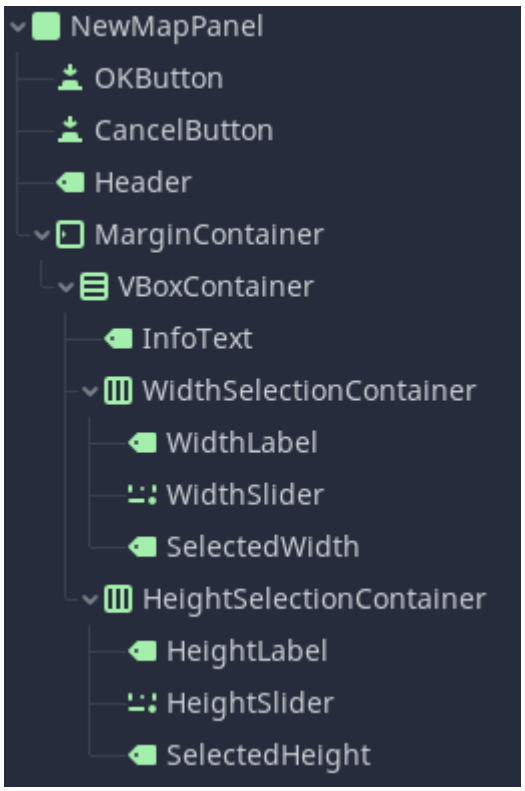
Panel nápovědy je vždy dostupný pod tlačítkem nápovědy nebo lze zobrazit a skrýt klávesou $F1$. Panel se skládá z mnoha do sebe vnořených komponent 5.9, které dohromady utváří zjednodušenou uživatelskou dokumentaci k aplikaci. Panel obsahuje textové vysvětlení funkcí spolu s obrázky tlačítek a popisem jejich funkcí.



Obrázek 5.10 Ukázka panelu nápovědy

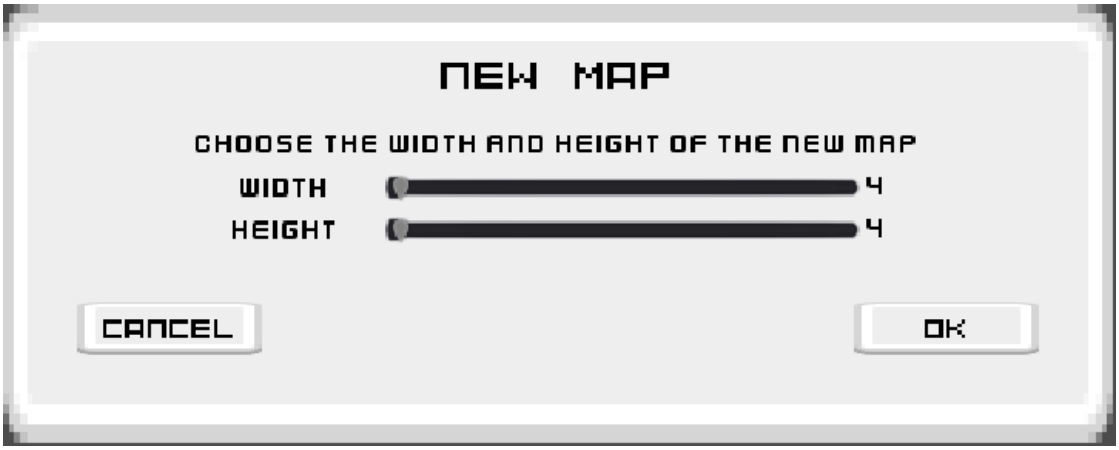
Dostupná nápověda je důležitá v jakémkoliv momentu aplikace, proto jsme se rozhodli jí udělat vždy dostupnou. Ukázku panelu nápovědy lze vidět na obrázku 5.10.

5.7.2.2 Panel vytvoření nové mapy



Obrázek 5.11 Struktura panelu vytvoření nové mapy

Struktura panelu vytvoření nové mapy obsahuje posuvníky, pomocí kterých uživatel zvolí výšku a šířku nové mapy 5.11.Tyto posuvníky mají omezené hodnoty od 4 do 256. Tento panel má v sobě i skript, který slouží pro nastavování hodnot štítků podle hodnot nastavených uživatelem. Dále obsahuje signál. Signál odešle informaci o výšce a šířce, kterou má nová mapa mít.

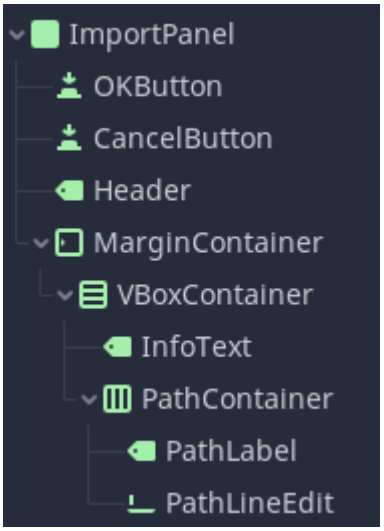


Obrázek 5.12 Ukázka panelu vytvoření nové mapy

Součástí panelu je i vizuální nápověda v podobě štítků, které pomáhají uživateli se zorientovat

a správně zadat přesné hodnoty, které chce 5.12.

5.7.2.3 Panely exportování a importování



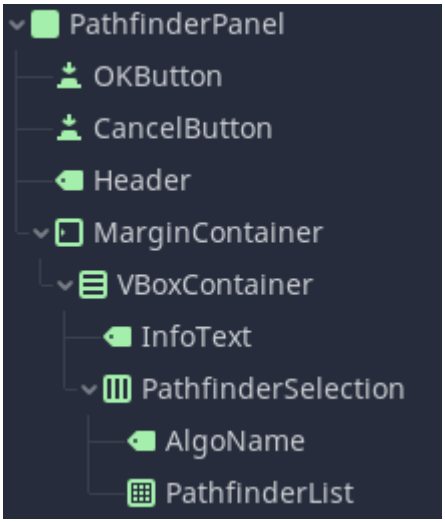
Obrázek 5.13 Struktura panelu importování nebo exportování scénáře

Panely pro exportování a importování jsou si tak podobné, že jsme se rozhodli je popsat dohromady. Oba obsahují textové pole, kam uživatel zadá cestu v souborovém systému 5.13. V případě panelu importování se tato cesta předá manažeru levelu, který ji dál předá komponentě sloužící pro importování scénářů. V případě panelu exportování se cesta předá opět manažeru levelu, který jí použije při exportování aktuálního scénáře.

Obrázek 5.14 Ukázka panelu importování nebo exportování scénáře

Součástí panelu je i vizuální nápověda v podobě štítků, které pomáhají uživateli se zorientovat a správně zadat cesty, které chce 5.14.

5.7.2.4 Panel volby algoritmu pro hledání cest

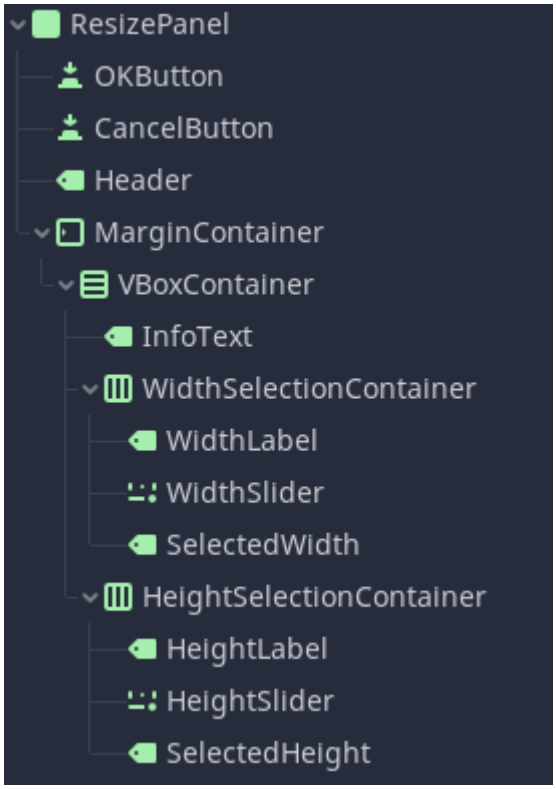


Obrázek 5.15 Struktura panelu změny algoritmu pro hledání cest

Struktura panelu volby algoritmu je jednoduchá 5.15. Panel si při svém otevření zažádá o aktuální informace o dostupných algoritmech a zobrazí jejich výběr uživateli 5.16. Uživatel poté vybere ten, který chce. Po potvrzení výběru tlačítkem se výběr pošle do manažera levelu, který tento požadavek na změnu zpracuje.

Obrázek 5.16 Ukázka panelu změny algoritmu pro hledání cest

5.7.2.5 Panel pro změnu velikosti mapy



Obrázek 5.17 Struktura panelu změny velikosti mapy

Panel pro změnu velikosti mapy vyvolaný stisknutím příslušného tlačítka má následující strukturu 5.17. Po potvrzení jsou informace o nových rozměrech poslány pomocí signálu. Tento panel 5.18 je podobný panelu vytvoření nové mapy 5.7.2.2.

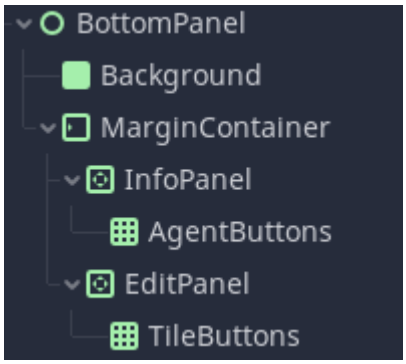
Obrázek 5.18 Ukázka panelu změny velikosti mapy

5.7.2.6 Komponenta pro zobrazování zpráv

Tato komponenta slouží k zobrazovaní různých zpráv, které jsou potřeba komunikovat uživateli. Nastaveni textu a doby zobrazení probíhá ve skriptu komponenty UI. Zprávy jsou omezené délkou a zobrazí se vždy jen na určitý počet vteřin, závislý na délce zprávy. Ukázka komponenty pro zobrazování zpráv je na obrázku 5.19.

Obrázek 5.19 Ukázka komponenty pro zobrazování zpráv

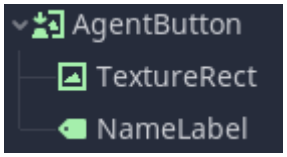
5.7.3 Spodní panel



Obrázek 5.20 Struktura spodního panelu

Spodní panel 5.20 má různý obsah podle toho, v jakém je zrovna aplikace módu. Pokud je aplikace v simulačním módu, zobrazuje právě označené agenty v podobě tlačítek 5.7.3.1. Pokud je aplikace v editačním módu, zobrazuje dostupná políčka terénu a tlačítka agenta 5.7.3.2. Z těchto tlačítek si může uživatel kliknutím na ně vybrat a začít pokládat.

5.7.3.1 Tlačítka označených agentů

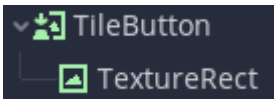


Obrázek 5.21 Struktura tlačítka označeného agenta

Tlačítka označených agentů ukazují agenta, kterého reprezentují a jeho identifikační jméno 5.22. Zároveň je k tlačítku připojen skript, který zařizuje, že při jeho vytváření lze připojit jím vysílaný signál k entitě, která ho vytvořila. To zařizuje metoda *connect_me*(). Když uživatel klikne na toto tlačítko, je vyslán signál pro zrušení označení agenta.

Obrázek 5.22 Ukázka tlačítka označeného agenta

5.7.3.2 Tlačítka volby terénu a agenta k pokládání



Obrázek 5.23 Struktura tlačítka pro volbu terénu

Tato tlačítka zobrazují výběr terénů dostupný pro položení na mapu 5.24, nebo položení nového agenta na mapu 5.25. Obdobně jako tlačítka pro označené agenty 5.7.3.1, mají i tato tlačítka metodu na registrování signálu a vysílají ho při stisknuti. Tato tlačítka jsou generována UI skriptem pro zobrazení aktuálních možností položení terénu a agenta.

Obrázek 5.24 Ukázka tlačítka pro volbu terénu

Obrázek 5.25 Ukázka tlačítka pro volbu přidání agenta

5.8 Nativelib - C++ knihovna algoritmů pro hledání cest

V této sekci popíšeme rozhraní, které implementují algoritmy pro hledání cest používané naší aplikací.

5.8.1 Rozhraní pro algoritmy

Výpis kódu 5.1 IPathfinder rozhraní

```
class IPathfinder: public Node {
    GODOT_CLASS(IPathfinder, Node)
    public:
    inline static void _register_methods() {
        register_method("setup", &IPathfinder::setup);
        register_method(
            "generate_solution", &IPathfinder::generate_solution);
    }
    inline void _init() {}

    virtual ~IPathfinder() {}
    virtual void setup(Ref<PathfinderSetupDTO> pathfidner_setup_DTO) {}
    virtual Ref<SolutionDTO> generate_solution(
        Ref<ScenarioDTO> scenario_DTO)
    { return new SolutionDTO; }
};
```

Rozhraní dědí třídu *Node* z herního enginu Godot. Používá Godot makro *GODOT_CLASS*, které zajistí dobré propojení se zbytkem Godotu. Rozhraní dále má statickou metodu z Godotu *_register_methods*(), která slouží k registraci metod a proměnných, aby byly viditelné pro Godot. Metoda *_init*() slouží jako konstruktor v rámci Godotu. Virtuální destruktor pro bezproblémové dědění. Další dvě metody slouží jíž k funkcionalitě algoritmu a jsou volány zbytkem aplikace.

Metoda *setup*() přijímá na svém vstupu datovou strukturu *PathfinderSetupDTO* 5.10.2, která obsahuje informace, které může algoritmus využít pro nastavení ještě předtím, než bude zpracovávat první zadaný problém. Druhá metoda *generate_solution*() dostává na vstupu datovou strukturu *ScenarioDTO* 5.10.3, která obsahuje zadaný problém pro vyřešení algoritmem. Poté, co algoritmus vyřeší problém, vrátí metoda řešení uložené v datové struktuře *SolutionDTO* 5.10.4.

5.8.2 Aktuálně implementované algoritmy

Aktuálně jsou implementovány dva algoritmy, které využívají rozhraní 5.8.1. Jedním z těchto algoritmů je primitivní algoritmus, který nebere v úvahu překážky ve scénáři a pouze vytvoří tříbodovou cestu mezi počátečními a cílovými pozicemi agentů. Druhý algoritmus je implementace algoritmu CA* 3.2.3.

5.8.3 Implementace vlastního algoritmu

Pro přidání nového vlastního algoritmu je potřeba vytvořit nové soubory *mujalgorimtus.cpp* a *mujalgorimtus.hpp* ve vlastní podsložce složce *NativeLib/src/Pathfinder/MujAlgoritmus*. Poté je potřeba se inspirovat u již hotových implementací. Nový algoritmus musí dědit rozhraní *IPathfinder*. Důležitou součástí je Godot makro *GODOT_SUBCLASS*, které zajišťuje dobrou propojenost se zbytkem Godotu. Poté je potřeba překrýt metody *setup*() a *generate_solution*().

Novou třídu je třeba registrovat v souboru *NativeLib.cpp*, pomocí funkce *register_class()*, opět je dobré se inspirovat již registrovaným třídami. Poté je potřeba překompilovat celou C++ knihovnu pomocí nástroje SCons, viz sekce 5.1.

Přidání nového algoritmu pokračuje jeho přidáním do Godotu. Začneme přidáním Godot komponenty NativeScript do složky *Godot/scripts/pathfinding* a nastavením pro něj vlastnosti *Library* na námi zkompilovanou knihovnu a vlastnosti *Class Name* na jméno námi nově vytvořené třídy.

Posledním krokem je přidat nový algoritmus jako možnost do manažéra levelu. Otevřeme skript *level_manager.gd* a do metody *setup_pathfinder_options()* přidáme nový algoritmus pod námi zvoleným jménem.

Celou aplikaci poté znovu zkompletujeme viz 5.1.

5.9 Druhy terénu

Obrázek 5.26 Ukázka implementovaných terénů a agenta

V aplikaci jsme implementovali terény ukázané v obrázku 5.26. Z těchto terénů si může uživatel vybrat při úpravě map.

5.10 Komunikační struktury

V této sekci podrobně popíšeme implementace jednotlivých komunikačních struktur.

5.10.1 Scénář

```
int height
int width
String raw_map
Array agent_positions
```

Struktura drží informace o výšce, šířce a řetězci znaků reprezentujícím jednotlivé terény a pozice agentů.

5.10.2 Nastavení algoritmu hledání cest

```
int height
int width
String raw_map
```

V této struktuře jsou uloženy informace o výšce a šířce mapy. Také obsahuje řetězec reprezentující mapu, složený z vah jednotlivých terénů.

5.10.3 Zadání problému

```
Dictionary agent_positions
Array other_agent_positions
Dictionary final_agent_positions
String leader_agent
Dictionary formation
```

Struktura drží informace o pozicích jednotlivých agentů pod jejich identifikačním jménem, pozice všech ostatních agentů, kteří se nezúčastní zadaného problému, cílové pozice jednotlivých agentů pod jejich identifikačním jménem, identifikační jméno vedoucího agenta a samotná formace vzhledem k vedoucímu agentovi.

5.10.4 Řešení problému

```
bool success
Dictionary agent_paths
long unsigned int computation_time
```

Struktura obsahuje binární hodnotu úspěchu či neúspěchu hledání cest, cesty jednotlivých agentů pod jejich identifikačním jménem a čas, jak dlouho trvalo řešení nalézt v mikrosekundách.

5.10.5 Výsledky testování

```
SolutionDTO solutionDTO
Array formation_deviations
float total_formation_deviation
Dictionary formation
String leader_agent
```

Struktura obsahuje informace o řešení zadaného problému, odchylku od formace v jednotlivých časových krocích simulace, celkovou odchylku formace, formaci samotnou vzhledem k vedoucímu agentovi a identifikační jméno vedoucího agenta.

Kapitola 6

Uživatelská dokumentace řešení

V kapitole uživatelské dokumentace je sepsán návod pro uživatele na ovládání naší aplikace. Zjednodušená verze je dostupná i přímo v aplikaci v panelu nápovědy 5.7.2.1. Panel nápovědy je uživatelsky přívětivější pro ukázku, co všechno dokáže uživatel v aplikaci dělat, zároveň je rovnou integrovaný v aplikaci a je možné ho kdykoliv otevřít. Tudíž doporučujeme používat panel nápovědy jako referenci a uživatelskou příručku. V následujících sekcích popíšeme jak fungují jednotlivé části aplikace a jak je může uživatel co nejlépe využít.

6.1 Pohyb po mapě scénáře

Obrázek 6.1 K pohybu po mapě slouží šipky

Pro pohyb po mapě slouží klávesy šipek 6.1. Alternativní metoda pohybu je myš. Pro pohyb nahoru musí uživatel přesunout myš poblíž horního kraje obrazovky, obdobně s ostatními směry. Zobrazení lze i přiblížit a oddálit pomocí kolečka myši.

6.2 Módy aplikace

Obrázek 6.2 Aplikace má dva módy

Aplikace má dva módy 6.2, ve kterých se může nacházet, editační a simulační. Mezi módy je možné přepínat klávesou *TAB* nebo pomocí tlačítek 6.3.

Obrázek 6.3 Tlačítka na přepínání aplikačních módů

6.2.1 Simulační mód

První ze dvou módů aplikace je mód simulační. V tomto módu může uživatel označovat agenty, které chce následně použít v zadávání problému. Následně po spuštění simulace se spustí animování agentů, kteří mají zadanou cestu.

6.2.1.1 Označování agentů

Pro označení agentů uživatel použije myš a může si vybrat ze dvou různých způsobů. První způsob je kliknutí levým tlačítkem myši přímo na agenta a tím ho buď označit, pokud označený zatím není, nebo zrušit označení, pokud označený již byl. Druhý způsob je kliknutí levým tlačítkem a potažení myší, což vytvoří označovací obdélník od počáteční pozice, kde uživatel stiskl tlačítko až k aktuální pozici myši. Následně po uvolnění tlačítka se označí všichni agenti, kteří jsou v označovacím obdélníku. Označené agenty lze poznat podle zeleného obdélníku kolem nich 6.4. Označení agenti se zobrazí ve spodním panelu, kliknutí na tlačítko agenta se agentovi zruší jeho označení.

Obrázek 6.4 Ukázka označeného agenta

6.2.1.2 Zadávání cílových pozice agentů

Zadávání cílových pozic agentů probíhá po označení agentů, které chce uživatel, aby byli součástí zadaného problému. Formace agentů je určena jejich vzájemnými počátečními pozicemi. Vedoucí agent je určen podle toho, který agent je nejblíž k pozici myši (Euklidovská vzdálenost).

Uživateli se na pozici myši zobrazí cílové pozice agentů, jako poloprůhledné siluety agentů. Obraz výsledné pozice agenta se zbarví červeně, pokud na toto místo neleze agenta poslat. Při kliknutí pravým tlačítkem myši se potvrdí cílové pozice agentů.

Po potvrzení odešle aplikace požadavek zvolenému algoritmu zadaným problémem. Pokud je hledání cest úspěšné, jsou agentům nastaveny jejich nové cesty a při spuštění simulace se agenti začnou po svých cestách pohybovat. Pokud hledání řešení není úspěšné, agentům se nenastaví žádné nové cesty a zobrazí se hláška, že hledání řešení nebylo úspěšné.

6.2.1.3 Spuštění a pozastavení simulace

Simulace může být spuštěná nebo pozastavená. V pozastavené simulaci se agenti nehýbají po svých cestách, ale stále je lze označovat. Pokud je simulace spuštěna, agenti se pohybují (jsou animování) po svých aktuálních cestách (pokud nějaké mají).

6.2.2 Editační mód

V editačním módu může uživatel měnit terény na mapě, přidávat nebo odebírat agenty a provádět další úpravy celého scénáře.

6.2.2.1 Vytvoření nové mapy

Vytvořit novou mapu lze pomocí panelu 5.12, který se otevře při kliknutí na příslušné tlačítko. Na panelu uživatel zadá požadovanou šířku a výšku nové mapy. Po potvrzení tlačítkem *OK* aplikace vytvoří novou prázdnou mapu o zadaných rozměrech.

6.2.2.2 Import a Export scénářů

Importování a exportování scénářů probíhá podobně. Při kliknutí na příslušná tlačítka se otevře panel 5.14, ve kterém lze zadat cestu k souboru, ze kterého respektive do kterého bude probíhat importování respektive exportování scénáře. Scénáře jsou ve formátu JSON.

6.2.2.3 Změna velikosti mapy

Zadání nových rozměrů pro aktuální mapu scénáře probíhá v panelu změny velikosti 5.18. Obdobně jako v panelu pro novou mapu uživatel zadá nové rozměry výšky a šířky. Po potvrzení se mapa scénáře přizpůsobí nových rozměrům. Všechna pole mimo nově vymezené rozměry jsou smazána a na všechna nová pole je položen výchozí terén trávy s váhou 1.

6.2.2.4 Změna algoritmu

Pro změnu algoritmu uživatel může otevřít panel změny algoritmu 5.16 příslušným tlačítkem. Panel zobrazí nabídku registrovaných algoritmů a uživatel si mezi nimi vybere ten, který chce používat. Po potvrzení je algoritmus nastaven pro hledání všech dalších řešení uživatelem zadaných problémů.

6.2.2.5 Pokládání nových terénů a agentů

Pro pokládání terénů a nových agentů uživatel musí být v editačním módu. Na spodním panelu se zobrazí dostupné terény a agent 5.26. Kliknutím na terén nebo agenta si uživatel vybere co chce pokládat.

Pokládání terénů a agentů probíhá obdobně. Uživatel myší namíří na místo, kam chce terén nebo agenta položit. Během vybírání místa se na cílovém místě zobrazuje silueta pokládaného terénu nebo agenta 6.5, pokud na cílové místo nelze zvolený terén nebo agenta položit, zbarví se silueta červeně. Pro potvrzení cílové pozice se na zvolenou pozici, pokud je to možné, položí terén nebo agent, pokud to není možné zobrazí se hláška, proč to nejde.

Obrázek 6.5 Ukázka siluet cílových pozic agentů

Pro zrušení výběru pokládaného terénu nebo agenta uživatel může zmáčknout pravé tlačítko myši. Pro odstranění agenta ze scénáře na něj může uživatel kliknout pravým tlačítkem myši.

6.3 Testování a výsledky testů

Po úspěšném doběhnutí simulace, to jest, žádný z agentů se nestal součástí jiné simulace a žádný z agentů nebyl odstraněn, se přejde k testování simulace. Testování simulace probíhá na pozadí a výsledek je zapsán do souboru do složky *evaluations* pod interním identifikačním názvem simulace ve formátu JSON. Součástí výsledků testování uložených do souboru je výčet identifikačních jmen agentů, kteří byli součástí simulace, cesty jednotlivých agentů, odchylky od formace v jednotlivých časových krocích, celková odchylka od formace za během simulace a celkový čas, který trval výpočet.

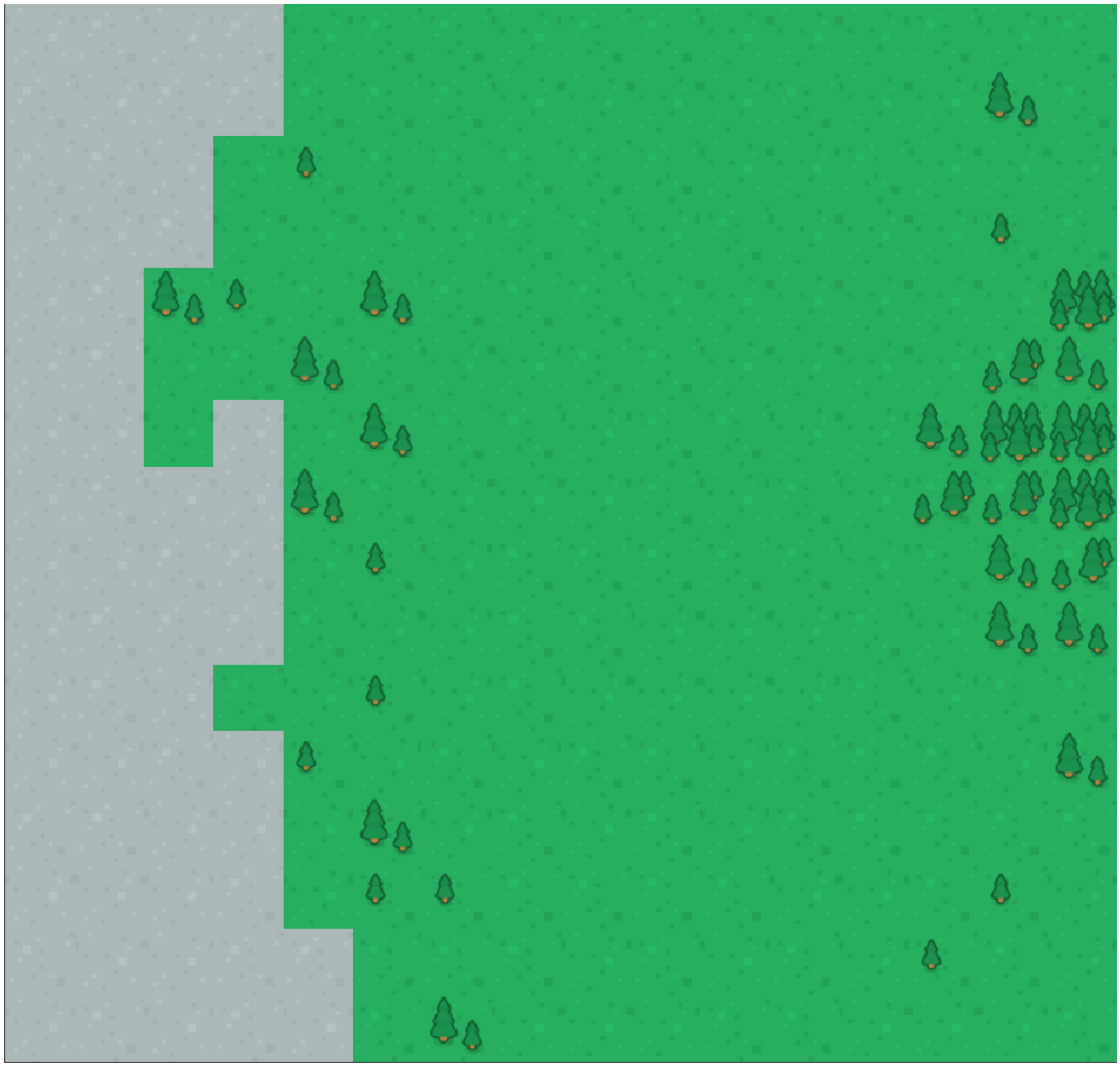
Kapitola 7

Ukázky testovaní scénářů

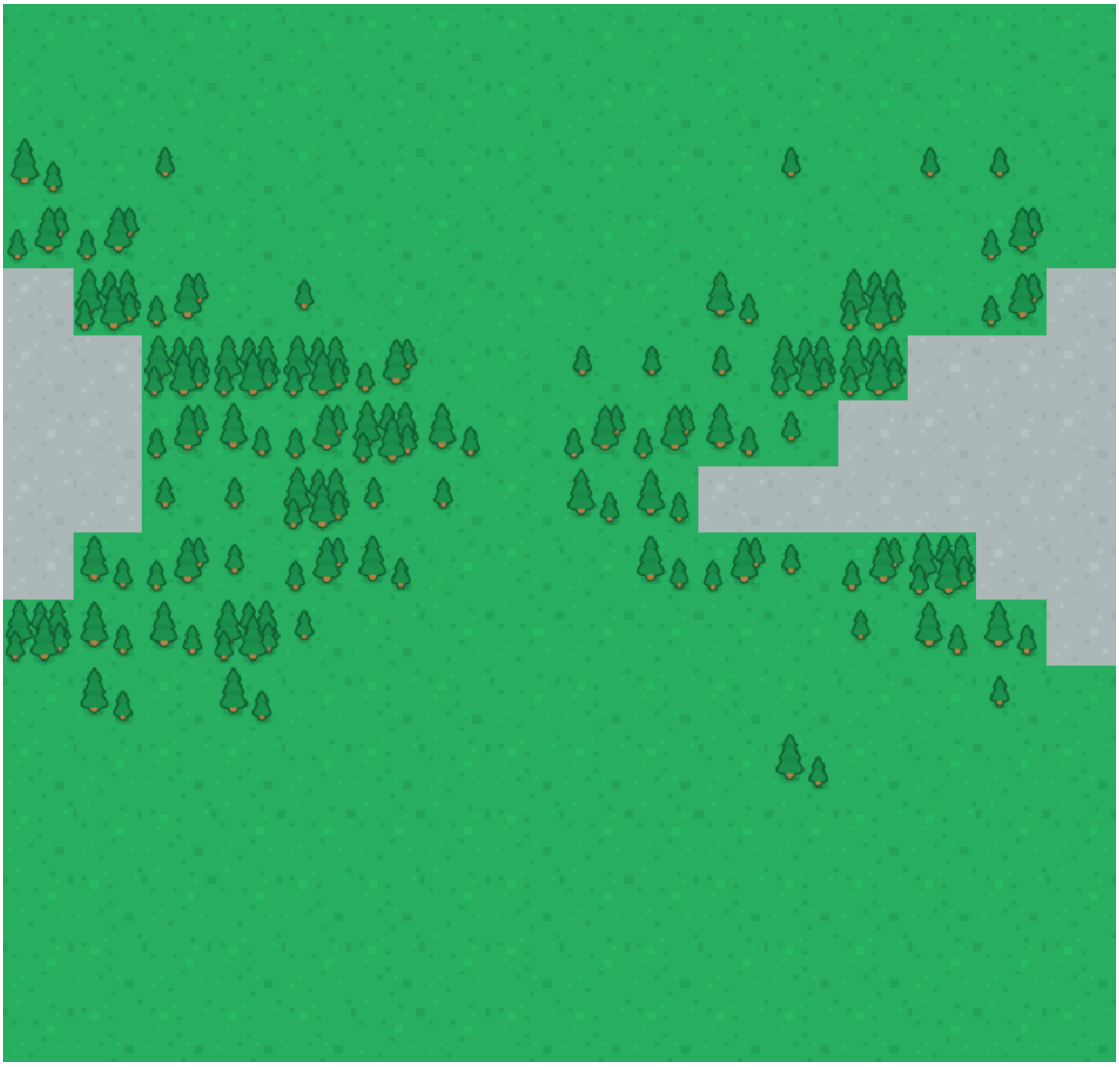
Pro ukázku testovacích schopností naší aplikace otestujeme v této kapitole několik typických problémů v různých scénářích.

7.1 Scénáře pro demonstraci problémů

Zde jsou obrázky scénářů, na kterých budou demonstrovány problémy a testy simulací.



Obrázek 7.1 Scénář údolí, na tomto scénáři by agenti neměli mít problém se přesouvat



Obrázek 7.2 Průsmyk uprostřed lesa, agenti se zde budou muset přizpůsobit zúžení

Obrázek 7.3 Neprůchodný les uprostřed rozděluje scénář na horní a dolní část, agenti nebudou schopni se dostat z horní části do spodní

7.2 Zadaný problém pro jednoho agenta

Ve scénářích vyzkoušíme zadat problémy přesunu po mapě pro jednoho agenta. Aplikace správně otestuje řešení a zapíše je do souboru ve formátu JSON.

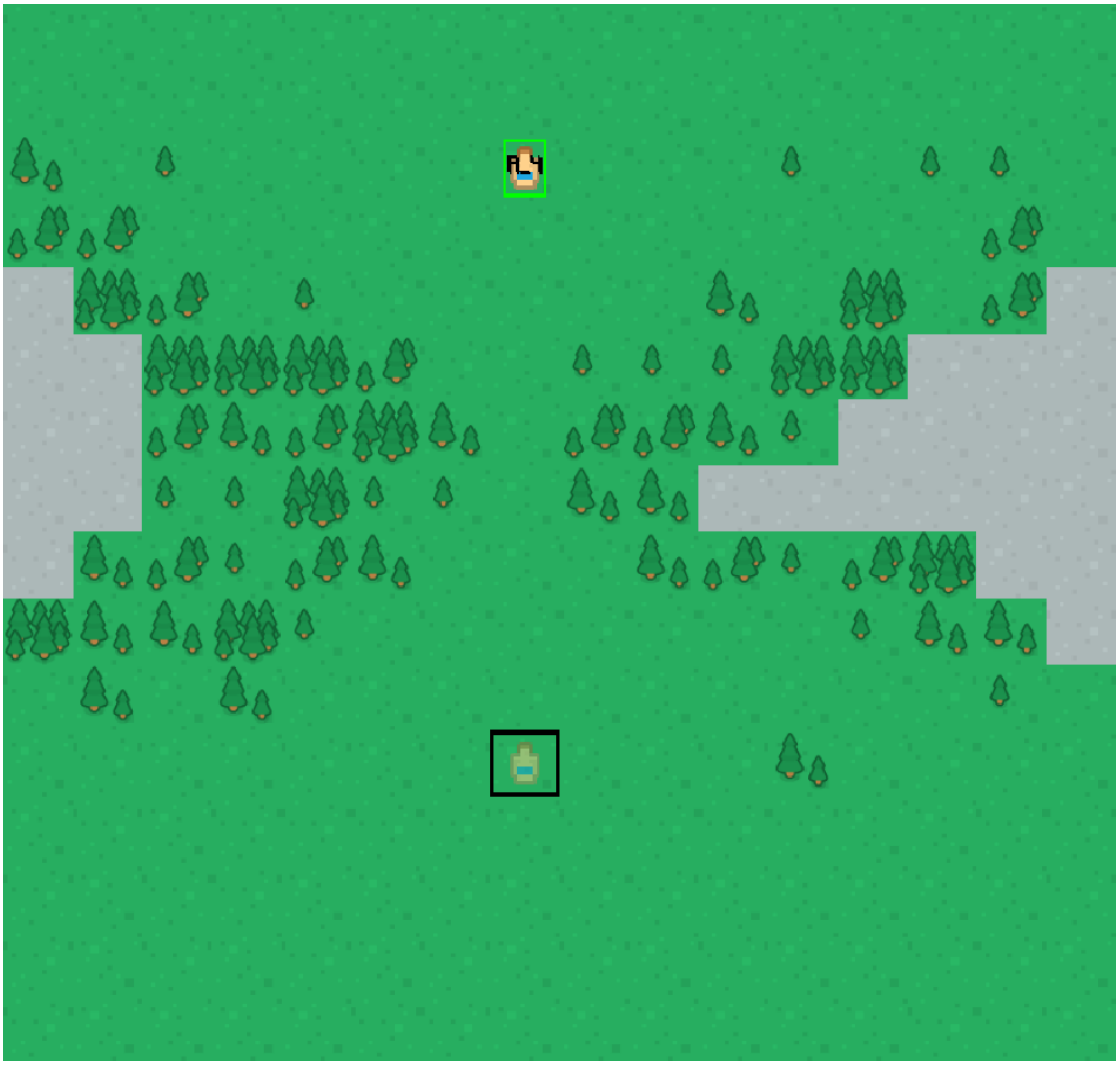
7.2.1 Scénář údolí

Obrázek 7.4 Zadání problému pro jednoho agenta ve scénáři údolí

Po doběhnutí simulace aplikace zapsala do souboru následující výsledky testování:


```
{
    "agents": [ "A_0" ],
    "agent_paths": {
        "A_0": [ "(9, 3)",  "(9, 4)", "(9, 5)", "(9, 6)", "(9, 7)",
            "(9, 8)", "(9, 9)", "(9, 10)" ]
    },
    "computation_time": 346,
    "formation_deviations": [ 0, 0, 0, 0, 0, 0, 0, 0 ],
    "total_formation_deviation": 0
}
```


7.2.2 Scénář průsmyku

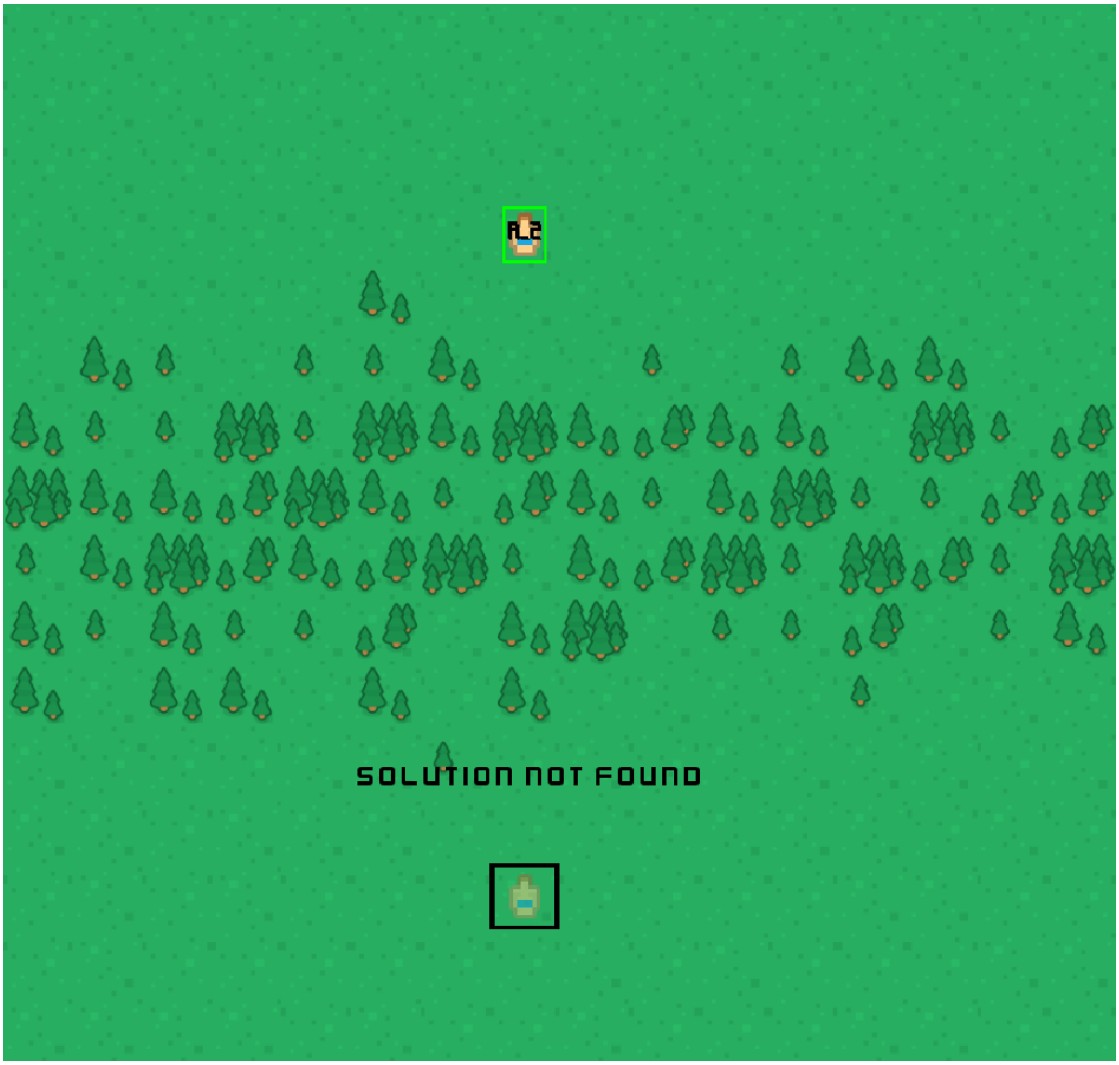


Obrázek 7.5 Zadání problému pro jednoho agenta ve scénáři průsmyku

Po doběhnutí simulace aplikace zapsala do souboru následující výsledky testování:


```
{
"agents": [ "A_4" ],
"agent_paths": {
"A_4": [ "(7, 2)", "(7, 3)", "(7, 4)", "(7, 5)", "(7, 6)",
"(7, 7)", "(7, 8)", "(7, 9)", "(7, 10)", "(7, 11)" ]
},
"computation_time": 334,
"formation_deviations": [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
"total_formation_deviation": 0
}
```


7.2.3 Scénář neprůchodného lesa



Obrázek 7.6 Zadání problému pro jednoho agenta ve scénáři neprůchodného lesa

Algoritmus (správně) nenašel řešení pro zadaný problém a tudíž nebyla započata žádná simulace a neproběhlo žádné testování.

7.3 Zadaný problém pro formaci agentů

Ve scénářích vyzkoušíme zadat problémy přesunu po mapě pro formaci agentů. Aplikace správně otestuje řešení a zapíše je do souboru ve formátu JSON.

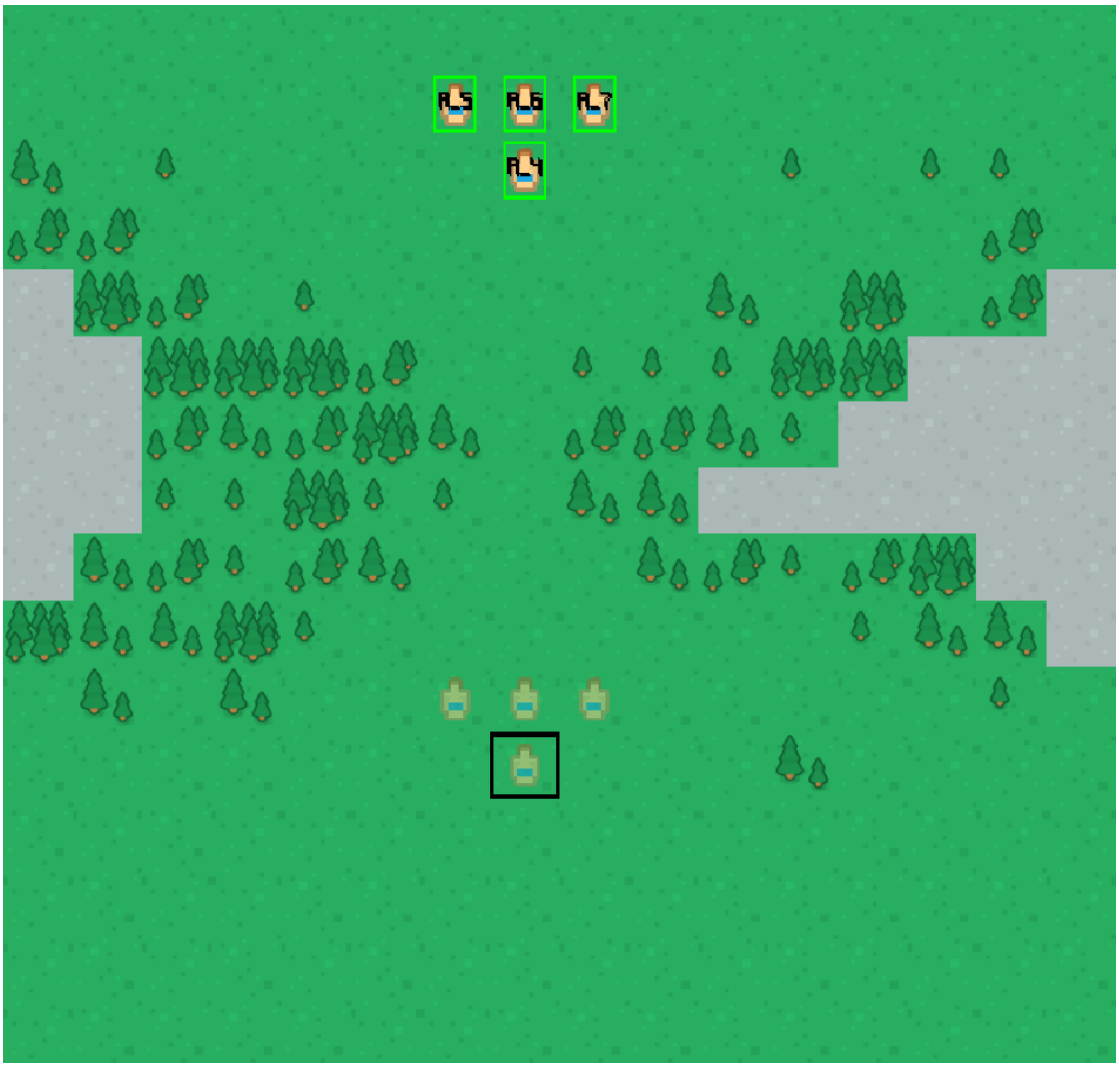
7.3.1 Scénář údolí

Obrázek 7.7 Zadání problému pro formaci agentů ve scénáři údolí

Po doběhnutí simulace aplikace zapsala do souboru následující výsledky testování:


```
{
"agents": [ "A_0", "A_1", "A_2", "A_3" ],
"agent_paths": {
"A_0": [ "(9, 3)", "(9, 4)", "(9, 5)", "(9, 6)", "(9, 7)",
"(9, 8)", "(9, 9)", "(9, 10)", "(9, 11)" ],
"A_1": [ "(8, 2)", "(8, 3)", "(8, 4)", "(8, 5)", "(8, 6)",
"(8, 7)", "(8, 8)", "(8, 9)", "(8, 10)" ],
"A_2": [ "(9, 2)", "(9, 3)", "(9, 4)", "(9, 5)", "(9, 6)",
"(9, 7)", "(9, 8)", "(9, 9)", "(9, 10)" ],
"A_3": [ "(10, 2)", "(10, 3)", "(10, 4)", "(10, 5)", "(10, 6)",
"(10, 7)", "(10, 8)", "(10, 9)", "(10, 10)" ]
},
"computation_time": 1252,
"formation_deviations": [ 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
"total_formation_deviation": 0
}
```

7.3.2 Scénář průsmyku

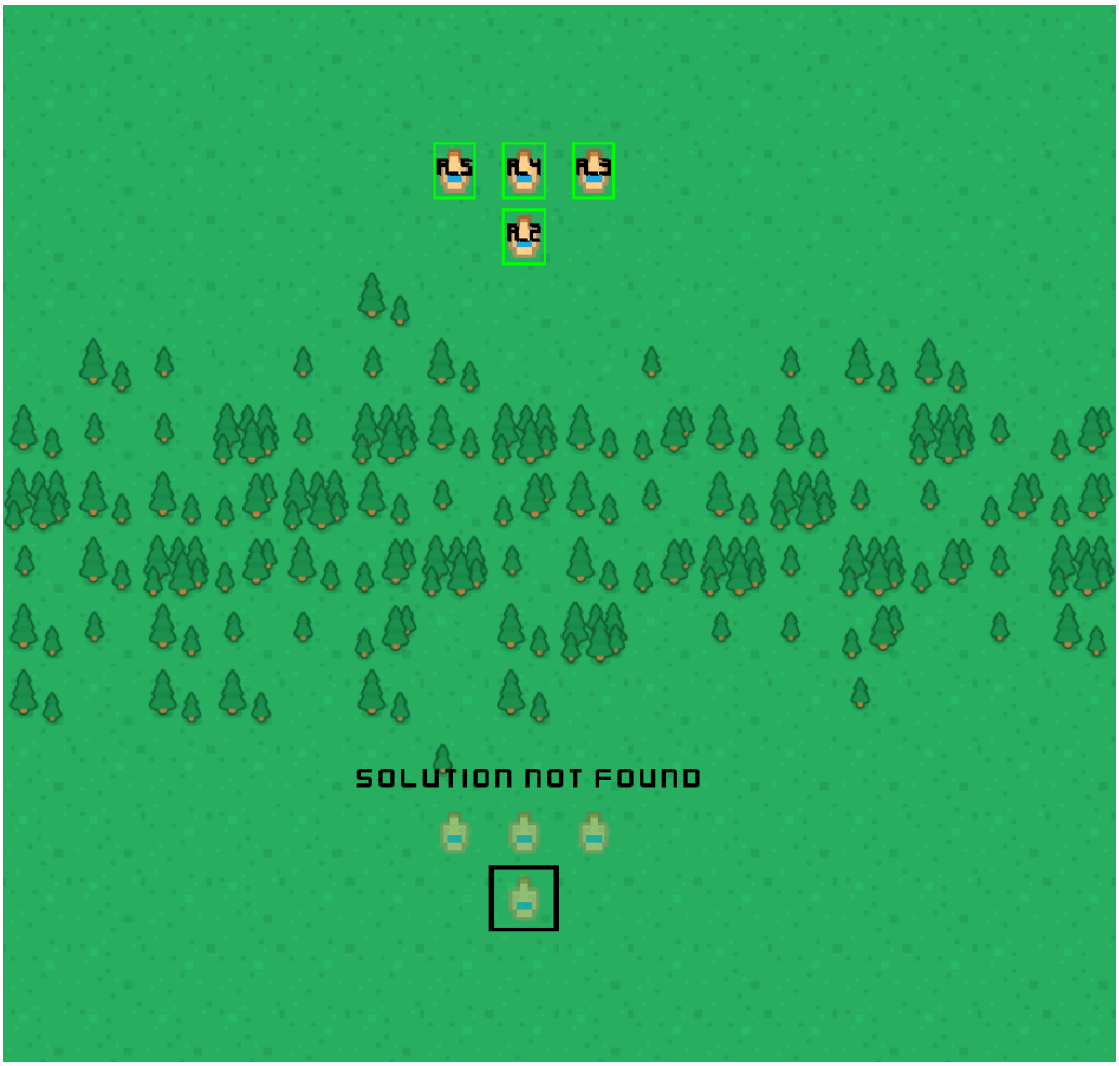


Obrázek 7.8 Zadání problému pro formaci agentů ve scénáři průsmyku

Po doběhnutí simulace aplikace zapsala do souboru následující výsledky testování:


```
{
"agents": [ "A_4", "A_5", "A_6", "A_7" ],
"agent_paths": {
"A_4": [ "(7, 2)", "(7, 3)", "(7, 4)", "(7, 5)", "(7, 6)",
"(7, 7)", "(7, 8)", "(7, 9)", "(7, 10)", "(7, 11)" ],
"A_5": [ "(6, 1)", "(6, 2)", "(6, 3)", "(6, 4)", "(6, 5)", "(7, 5)",
            "(7, 6)", "(7, 7)", "(7, 8)", "(7, 9)", "(7, 10)", "(6, 10)" ],
"A_6": [ "(7, 1)", "(7, 2)", "(7, 3)", "(7, 4)", "(7, 5)",
            "(7, 6)", "(7, 7)", "(7, 8)", "(7, 9)", "(7, 10)" ],
"A_7": [ "(8, 1)", "(8, 2)", "(8, 3)", "(8, 4)", "(7, 4)",
            "(7, 4)", "(7, 5)", "(7, 6)", "(7, 7)", "(7, 8)",
            "(7, 9)", "(7, 10)", "(8, 10)" ]
},
"computation_time": 1270,
"formation_deviations": [ 0, 0, 0, 0, 1.414214, 3.650282, 3.650282,
        3.650282, 3.650282, 3.650282, 2.414214, 1, 0 ],
"total_formation_deviation": 23.079835
}
```

7.3.3 Scénář neprůchodného lesa



Obrázek 7.9 Zadání problému pro formaci agentů ve scénáři neprůchodného lesa

Algoritmus (správně) nenašel řešení pro zadaný problém a tudíž nebyla započata žádná simulace a neproběhlo žádné testování.

Kapitola 8
Závěr

Cílem práce bylo vytvoření interaktivního grafického prostředí (aplikace) pro vizualizaci a dále testování udržování formací v diskrétní variantě multi-agentního hledání cest ve dvourozměrných mřížkových mapách. Byl předpoklad, že prostředí bude umožňovat zadávání instance problému, tj. počáteční a cílové pozice agentů, a stejně tak bude umožňovat editovat mapu, kde se agenti pohybují. Tento cíl jsme splnili vybudováním interaktivní aplikace, dovolující vytvoření či načtení mapy ze souboru, a dále její editace pomocí editačních nástrojů a ovládání myší za účelem zadání konkrétního problému. Dále je v aplikaci možné zadávat počáteční a cílové pozice agentů. V neposlední řadě aplikace reaguje na změny prostředí během simulace udržování formací v multi-agentním hledání cest, například při přidání či odebrání překážek, nebo úpravy cílových pozic agentů a jejich formací.

Cílem první části bakalářské práce bylo prozkoumat existující algoritmy pro udržování formací v multi-agentním prostředí, dále, pokud by to bylo nezbytné, rozšíření zvoleného algoritmu tak, aby zohledňoval dynamicky se měnící pozice ostatních agentů a interakci uživatele s mapou. Splnění tohoto cíle vyžadovalo vytvoření systému, který se změnám zadaným uživatelem přizpůsobí. Systém jsme nakonec uzpůsobili náhlým změnám tak, že se výpočet multi-agentní cesty a následně její simulace provedou při každé úpravě scénáře.

Cílem další části práce bylo navrhnout interaktivní prostředí pro ovládání aplikace, nalézt a použít vhodný programovací jazyk a grafickou knihovnu. Pro tuto práci jsme zvolili jazyk C++ z důvodu jeho možnosti optimálně implementovat algoritmy pro udržování formací v multi-agentním prostředí. Jako grafickou knihovnu jsme použili herní engine Godot kvůli jeho dobré interakci s jazykem C++ a dobře použitelnému rozhraní pro vytváření uživatelských rozhraní.

Jedním z cílů práce bylo vytvořit dostatečně obsáhlou dokumentaci aplikace. Součástí dokumentace je část uživatelská, která obsahuje instrukce pro uživatele pro ovládání aplikace. Druhou částí dokumentace je část programátorská, která podrobně popisuje konkrétní implementaci jednotlivých částí aplikace.

Posledním cílem práce bylo ukázat, že naše aplikace testuje scénáře požadovaných způsobem. Toho jsme docílili sadou testů, které ukazují obvyklé typy scénářů a spustili na nich hledání řešení a následnou simulaci obvyklých typů problémů. S těmito testy můžeme prohlásit, že aplikace skutečně testuje požadované vlastnosti algoritmů.

Výsledná aplikace má mnoho možností rozšíření, které můžeme dále implementovat. V oblasti testování je velký prostor pro rozšíření testů řešení, například kontrola, zdali není více agentů na jednom poli v jednom časovém kroku. Změny scénářů by šlo urychlit a zlepšit přidáním možnosti volby „štětců“, které by umožňovaly pokládat více terénů najednou, to by mohla dělat i volba *Vyplnit*, která by vyplňovala velké plochy stejného terénu najednou. V oblasti vylepšení uživatelského zážitku by bylo užitečné mít možnost skrýt uživatelské rozhraní, pro lepší pořizování záznamů obrazovky.

Bibliografie

1. LI, Jiaoyang; SUN, Kexuan; MA, Hang; FELNER, Ariel; KUMAR, T. K. Satish; KOENIG, Sven. Moving Agents in Formation in Congested Environments. In: *Proceedings of the 19th International Joint Conference on Autonomous Agents and Mult-Agent Systems (AAMAS)*. 2020, s. 726–734.
2. LOBACHAD. *Drones Fly Over The Island Of Mauritius In The Indian Ocean* [online]. 2021 [cit. 2023-04-21]. Dostupné z: `https://www.yayimages.com/39059044/drones-fly-over-the-island-of-mauritius-in-the-indian-ocean-a-natural-landscape-with-drones-flying-over-it-quadrocopter.html`.
3. RENDEREDDOOR695. *Formace ve videohře Age of Empires* [online]. 2021 [cit. 2023-04-21]. Dostupné z: `https://cdn.ageofempires.com/aoeforums/original/3X/e/0/e0c2090e1fd3e1f0772cc80a8dd5c9b98eb47ad1.jpeg`.
4. SHARON, Guni; STERN, Roni; FELNER, Ariel; STURTEVANT, Nathan R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. 2015, roč. 219, s. 40–66.
5. STURTEVANT, Nathan; GEISBERGER, Robert. A Comparison of High-Level Approaches for Speeding Up Pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2010, roč. 6, č. 1, s. 76–82. Dostupné z DOI: `10.1609/aiide.v6i1.12400`.
6. COHEN, Benjamin; CHITTA, Sachin; LIKHACHEV, Maxim. Single- and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research*. 2014, roč. 33, č. 2, s. 305–320. Dostupné z DOI: `10.1177/0278364913507983`.
7. BROCH, Josh; MALTZ, David A; JOHNSON, David B; HU, Yih-Chun; JETCHEVA, Jorjeta. A performance comparison of multi-hop wireless ad hoc network routing protocols. In: *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*. 1998, s. 85–97.
8. KORF, Richard E; TAYLOR, Larry A. Finding optimal solutions to the twenty-four puzzle. In: *Proceedings of the national conference on artificial intelligence*. 1996, s. 1202–1207.
9. KORF, Richard E. Finding optimal solutions to Rubik's Cube using pattern databases. In: *AAAI/IAAI*. 1997, s. 700–705.
10. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: `10.1109/TSSC.1968.300136`.
11. DECHTER, Rina; PEARL, Judea. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*. 1985, roč. 32, č. 3, s. 505–536. ISSN 0004-5411. Dostupné z DOI: `10.1145/3828.3830`.

12. SILVER, David. Cooperative pathfinding. In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. 2005, sv. 1, s. 117–122. Č. 1.

13. ZELINSKY, Alexander. A mobile robot navigation exploration algorithm. *IEEE Transactions of Robotics and Automation*. 1992, roč. 8, č. 6, s. 707–717.

14. LATOMBE, Jean-Claude; LAZANAS, Anthony; SHEKHAR, Shashank. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*. 1991, roč. 52, č. 1, s. 1–47.

15. MA, H.; YANG, J.; COHEN, L.; KUMAR, T. K. S.; KOENIG, S. Feasibility Study: Moving Non-Homogeneous Teams in Congested Video Game Environments. In: *AIIDE*. 2017, s. 270–272.

16. DRESNER, Kurt; STONE, Peter. A multiagent approach to autonomous intersection management. *Journal of artificial intelligence research*. 2008, roč. 31, s. 591–656.

17. BENNEWITZ, Maren; BURGARD, Wolfram; THRUN, Sebastian. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and autonomous systems*. 2002, roč. 41, č. 2-3, s. 89–99.

18. PALLOTTINO, Lucia; SCORDIO, Vincenzo G; BICCHI, Antonio; FRAZZOLI, Emilio. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*. 2007, roč. 23, č. 6, s. 1170–1183.

19. YU, Jingjin; LAVALLE, Steven. Structure and intractability of optimal multi-robot path planning on graphs. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2013, sv. 27, s. 1443–1449. Č. 1.

20. MORRIS, R.; PASAREANU, C.; LUCKOW, K.; MALIK, W.; MA, H.; KUMAR, S.; KOENIG, S. Planning, Scheduling and Monitoring for Airport Surface Operations. In: *AAAI-16 Workshop on Planning for Hybrid Systems*. 2016.

21. WURMAN, P. R.; D'ANDREA, R.; MOUNTZ, M. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. In: *AI Magazine 29, 1*. 2008, s. 9–20.

22. MA, H.; YANG, J.; COHEN, L.; KUMAR, T. K. S.; KOENIG, S. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In: *AAAI*. 2016, s. 3166–3173.

23. YU, J.; LAVALLE, S. M. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In: *AAAI*. 2013, s. 1444–1449.

24. HAIMES, Y.; LASDON, L.; WISMER, D. On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization. In: *IEEE Transactions on Systems, Man, and Cybernetics SMC-1, 3*. 1971, s. 296–297.

25. PEARL, J.; KIM, J. H. Studies in Semi-Admissible Heuristics. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence 4, 4*. 1982, s. 392–399.

26. STANDLEY, T. S. Finding Optimal Solutions to Cooperative Pathfinding Problems. In: *AAAI*. 2010, s. 173–178.

27. BARER, M.; SHARON, G.; STERN, R.; FELNER, A. Conflict-Based Search Algorithm for the Multi-Agent Pathfinding Problem. In: *SoCS*. 2014, s. 19–27.

28. LINIETSKY, Juan. *Godot 2.0: Talking with the Creator* [online]. 2016 [cit. 2023-04-22]. Dostupné z: `https://80.lv/articles/godot2-interview/`.

29. LINIETSKY, Juan; MANZUR, Ariel; COMMUNITY, Godot. *Godot documentation* [online]. 2014 [cit. 2023-04-22]. Dostupné z: `https://docs.godotengine.org/en/3.5/`.

30. STROUSTRUP, Bjarne. *The C++ Programming Language, Third Edition*. 3rd. Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN 0201889544.

31. STROUSTRUP, Bjarne. *C++ Applications* [online]. 2021 [cit. 2023-04-22]. Dostupné z: `https://www.stroustrup.com/applications.html`.

32. KOUPÝ, Petr. Visualization of problems of motion on a graph. 2010.

33. EVGENII, Abdalov. *Vizuální analýza plánů pro multi-agentní hledání cest se spojitým časem (MAPF-R)*. 2021. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

34. LI, Yutong. *MAPF Visualizer* [online]. 2022 [cit. 2023-05-09]. Dostupné z: `http://mapf-visualizer.com/`.

35. SCONSFOUNDATION. *SCons* [online]. 2001 [cit. 2023-05-08]. Dostupné z: `https://scons.org/`.

Obsah přiloženého média

- `application` ..... zdrojové kódy aplikace
  - `Godot` ..... zdrojové kódy Godot engine části aplikace
  - `godot-cpp` ..... zdrojové kódy enginu Godot
  - `NativeLib` ..... zdrojové kódy dynamicky nalinkované C++ knihovny
  - `SConstuct` ..... konfigurační soubor kompilace řešení pomocí nástroje *scons*
- `release` ..... adresář se spustitelnou formou aplikace
  - `evaluations` ..... adresář s výsledky testování simulací
  - `maps` ..... adresář se scénáři
  - `bakalarska-prace.exe` ..... spustitelná forma aplikace
  - `bakalarska-prace.pck` ..... podpůrné soubory aplikace
  - `libnative.dll` ..... zkompilovaná dynamicky nalinkovaná C++ knihovna
- `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `thesis.pdf` ..... text práce ve formátu PDF