FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Hierarchická koordinace multi-robotického konstrukčního týmu ve hře Minecraft
Student:	Vít Šprachta
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem výzkumu bude navrhnout řídící a koordinační metody pro tým virtuálních robotických agentů ve hře Minecraft, jejichž úkolem je postavit určitou stavbu. Nabízí se například hierarchické řízení týmu, které sestává z nadřazeného řídícího multi-agentního systému, který roboty koordinuje a přiřazuje jim úkoly, přičemž samotní roboti používají například klasické plánování pro plnění úkolu z jedno-robotického hlediska.
Úkoly pro uchazeče budou následující:

1. Prozkoumejte metody pro multi-agentní kolektivní konstrukci, které by mohly být aplikovány pro zkoumaný případ ve hře Minecraft.
2. Na základě provedení rešerše navrhněte vlastní hierarchický systém pro koordinaci multi-robotického konstrukčního týmu.
3. Navržený systém implementujte formou softwarového prototypu a ověřte jej na relevantních konstrukčních scénářích ve hře Minecraft.
4. Předpokládáme, že systém bude opatřen grafickým výstupem využívajícím rozšíření tzv. mod hry Minecraft.

[1] Edward Lam, Peter J. Stuckey, Sven Koenig, T. K. Satish Kumar: Exact Approaches to the Multi-agent Collective Construction Problem. CP 2020: 743-758
[2] Guillaume Sartoretti, Yue Wu, William Paivine, T. K. Satish Kumar, Sven Koenig, Howie Choset: Distributed Reinforcement Learning for Multi-robot Decentralized Collective Construction. DARS 2018: 35-49
[3] Kirstin Petersen, Radhika Nagpal, Justin Werfel: TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction. Robotics: Science and Systems 2011

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 13. listopadu 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Hierarchická koordinace multi-robotického konstrukčního týmu ve hře Minecraft

Vít Šprachta

Katedra aplikované matematiky
Vedoucí práce: doc. RNDr. Pavel Surynek, Ph.D.

13. května 2021

Poděkování

Rád bych poděkoval panu doc. RNDr. Pavlu Surynkovi, Ph.D. za vedení této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu 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 předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, 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 zpusobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelum. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Šprachta, Vít. *Hierarchická koordinace multi-robotického konstrukčního týmu ve hře Minecraft*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Minecraft je počítačová sandboxová hra, ve které se hráč ocitne ve světě z kostek a má volnou ruku nad tím co bude dělat. Zejména, co, kde a jak bude stavět.

Multi-agentní kolektivní robotická konstrukce je problém, kde se skupina robotů za úkol postavit trojrozměrnou strukturu z bloků. Roboti mohou jednotlivé bloky nosit, sbírat a pokládat. Vzhledem k tomu, že bloky bývají umístěné na mřížce, nabízí se hra Minecraft jako zajímavé simulační prostředí.

Tato práce nejprve projde teoretická východiska robotické kolektivní konstrukce, ze kterých vyjde při návrhu vlastního hierarchického multi-agentního systému. Při návrhu se bere v potaz občasná potřeba přistavit lešení k jinak nedostupným místům a zároveň se dbá na dobrou škálovatelnost na větší struktury o velikosti stovek tisíců bloků.

Výsledný systém se následně implementuje jako rozšíření (mod) do hry Minecraft a na experimentech se ukáže jeho schopnost stavit libovolné struktury spolu s velmi dobrou škálovatelností.

Klíčová slova kolektivní konstrukce, umělá inteligence, kooperativní agenti, multi-agentní systém, robotické stavění, Java, Minecraft, Minecraft Forge API

Abstract

Minecraft is a sandbox computer game, in which the player finds himself in a cubic world and has a free hand over he will do. In particular, what, where and how he pleases to build anything.

Multi-agent collective robotic construction is a problem where a group of robots are tasked to build a three-dimensional structure from blocks. Robots can carry, pick up and lay down individual blocks. Because the blocks are usually placed on a grid, Minecraft offers an interesting simulation environment.

Based on the theoretical background of collective robotic construction, this thesis designs its own hierarchical multi-agent system. The design takes into account the occasional need to construct scaffolding to reach otherwise inaccessible places, while also trying to ensure good scalability to larger structures consisting up to hundreds of thousands blocks.

The system is then implemented in form of an extension (mod) to the game Minecraft and follow-up experiments proves the system is able to build any structure while keeping very good scalability.

Keywords collective construction, artificial intelligence, cooperative agents, multi-agent system, robotic construction, Java, Minecraft, Minecraft Forge API

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Ať chceme, nebo ne, svět se postupně automatizuje. Setkáváme se samoobslužnými pokladnami či dokonce autonomními auty. Tato práce se zaměřuje na oblast výstavby, ve které současně dominují lidské týmy operující s těžkou technikou. Robotický konstrukční tým by mohl výstavbu na mnoha místech usnadnit, například v nepříznivých nebo nebezpečných podmínkách, jako třeba na Měsíci [1].

Práce se tedy zaměří na možnosti robotické kolektivní konstrukce. Náš studovaný případ bude ve hře Minecraft [2], která nabízí skvělé simulační prostředí pro stavění. Samotná hra je primárně dělaná jako sandbox, kde si každý může postavit to, co chce, je intuitivní a je i snadno rozšiřitelná, třeba právě o robotický tým, co staví za nás.

Struktura bude mít dvě části – teoretickou a praktickou. V teoretické nejprve prozkoumáme metody multi-agentní [3] kolektivní konstrukce a následně navrhneme metody vlastní, které v praktické části implementujeme jako rozšíření (mod) hry Minecraft.

Kapitola 1

Cíl práce

Cílem práce je vymyslet metody pro řízení multi-agentní kolektivní konstrukce a následně je implementovat a otestovat v rámci rozšíření (modu) hry Minecraft. Na splnění je potřeba vykonat několik dílčích cílů.

První dílčí cíl je provedení rešerše na multi-agentní kolektivní konstrukci. Je potřeba získat přehled o možných algoritmech a zhodnotit, které by byly použitelné ve hře Minecraft.

Druhý dílčí cíl je návrh vlastního multi-agentního systému na základě provedené rešerše. Vzhledem ke struktuře Minecraftu, kde jednotlivé entity nejsou provázány, bude ideální hierarchický mutli-agentní systém, ve kterém jednotliví roboti (agenti) budou dostávat úkoly od nadřazené řídící entity, která takto bude koordinovat stavbu.

Třetí dílčí cíl je vytvoření modu do hry Minecraft, který bude implementovat náš systém. Tedy bude do hry potřeba přidat novou entitu (robot), její umělou inteligenci a rozhraní, přes které budeme náš systém ve hře ovládat.

Poslední dílčí cíl je samotné otestování systému na relevantních konstrukčních scénářích ve hře.

Kapitola 2

Analýza současného stavu problému

Kapitola projde teoretická východiska práce, počínaje možnostmi kolektivní konstrukce a následně prodiskutuje Minecraft a jeho modování.

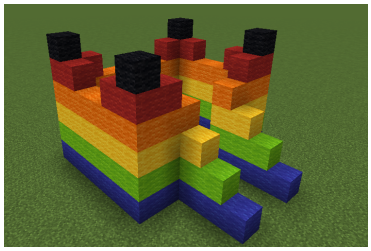
2.1 Kolektivní konstrukce

Obecně je kolektivní konstrukce je problém, kde máme plán 3D struktury (například obrázek 2.1) a množinu robotů, kteří mají za úkol tuto strukturu postavit. Struktura typicky sestává z bloků, které mohou roboti zvedat, nosit a pokládat. Všechny bloky jsou ve vrstvách na čtvercové mřížce a tudíž se jednotlivé struktury často zobrazují jen jako projekce na 2D mřížku, kde jednotlivá čísla v mřížce udávají výšku na daném místě (například obr. 2.3, nevyplněné číslo značí výšku 0). Struktury jdou také brát jako graf [4], kde vrcholy jsou jednotlivé bloky ohodnocené výškou a hrana mezi dvěma bloky vede tehdy, když manhattanská vzdálenost bloků je jedna a jejich výškový rozdíl je menší nebo rovno jedné.

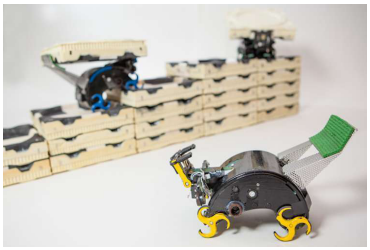
Hlavní dva směry řešení problému jsou plánovací a heuristický. Plánovací najde optimální řešení, ale stráví mnoho času počítáním, oproti tomu heuristicky najde neoptimální řešení, za to v krátkém čase. Dále je zde i možnost zpětnovazebného učení, ovšem tato metoda nepřináší uspokojivé výsledky – roboti mají často problém správně dokončit strukturu a také mají problémy při odklízení lešení [5].

2.1.1 Roboti

Harvardská práce TERMES [6] se zabývá roboty pro kolektivní konstrukci (obrázek 2.2). Každý robot je schopný vzít, nést a položit stavební blok, otočit se o 90°, dále se umí pohybovat a může vylézt na stavební blok. Roboti také



Obrázek 2.1: Možný plán struktury ve hře Minecraft (vlastní)



Obrázek 2.2: TERMES robot (autor: Eliza Grinnell, Harvard SEAS)

poznají, zdali je jiný robot před nimi. Tyto schopnosti postačují k tomu, aby roboti byly schopni postavit libovolnou zadanou strukturu [7]. Zároveň tito roboti mají velmi vysokou spolehlivost (během testování nedošlo k chybě) [6].

TERMES zároveň poskytuje jednoduchý pseudokód na ovládání robotů (alg. 1). Robot dostane přidělenou cestu, kterou následuje a hledá vhodné místo na položení bloku. Zároveň se cestou k zásobovací oblasti libovolně, ale vhodně vyhýbá ostatním robotům [6].

Roboti současných prací vycházejí právě z těchto robotů a jejich základ využívají v simulacích. Stejně tak tomu bude u této práce.

Algoritmus 1: Pseudokód řídícího programu robotů TERMES [6]

```
1  loop
2      go get block from supply
3      go to structure
4      climb onto structure
5      while on structure do
6          while robot detected ahead do
7              wait
8          follow given structpath
9          if holding block then
10             A ← structpath specifies block at current site
11             B ← previous site was at higher level
12             C ← previous site was at same level and meant to be empty
13             D ← next site along structpath is at same level
14             if A and (B or C) and D then
15                 move to the next site
16                 place block at site just vacated
```

6	5	4	4	4	4	5	6
5							5
4							4
4							4
5							5
6	5	4			4	5	6
		3			3		
		2			2		
		1			1		

(a) Struktura 1, hrad z obr. 2.1

		3			
		3			
		3			
3	3	3	3	3	3
		3			
		3			

(b) Struktura 2 – kříž

3	1	1	1	1	3
1					1
1					1
1					1
1					1
3	1	1	1	1	3

(c) Struktura 3 – hradba

1	1	1	1	1	1
1	2	2	2	2	1
1	2	3	3	2	1
1	2	3	3	2	1
1	2	2	2	2	1
1	1	1	1	1	1

(d) Struktura 4 – pyramida

Obrázek 2.3: 2D projekce struktur

2.1.2 Plánovací přístupy

Plánovacími přístupy se zabývá zejména práce [8]. Pro řešení problému testují dva modely – a to celočíselné programování (MILP) [9] a programování s omezujícími podmínkami (CP) [10].

Oba tyto modely hledají optimální řešení. V první řadě se snaží minimalizovat čas stavby a v druhé řadě počet potřebných akcí na dokončení. Tedy ve chvíli, kdy plánovač zná minimální možný čas, najde pro něj minimální počet akcí. Zároveň spočítají potřebné množství robotů pro daný plán.

Oproti heuristickým přístupům může být čas na dokončení stavby snížen až 100x, ovšem optimální řešení s sebou nese zásadní problém – čas potřebný na vytvoření plánu. Práce se testovala na strukturách stejných jako na 2-4 na obr. 2.3 a dalších velikostně podobných. Z experimentů vyšlo najevo, že MILP si vede mnohem lépe, než CP. Pro nejmenší struktury zvládl najít řešení v řádu

vteřin, pro nejsložitější strukturu (st. 2, obr. 2.3) v řádu dnů. Oproti tomu CP zvládl nejmenší strukturu za 1.2 hodiny a výpočet všech ostatních trval přes týden.

struktura z obr. 2.3	čas MILP	čas CP
2	5.7 d	>7 d
3	29 s	>7 d
4	1.2 h	>7 d

Tabulka 2.1: Některé experimentální výsledky práce [8]

Vytvoření plánu pro takto malé struktury může zabrat až týdny, kdybychom uvažovali struktury mnohonásobně větší, výpočty by byly časově neúnosné. Proto je tento přístup použitelný jen pro malé scénáře.

2.1.3 Heuristické přístupy

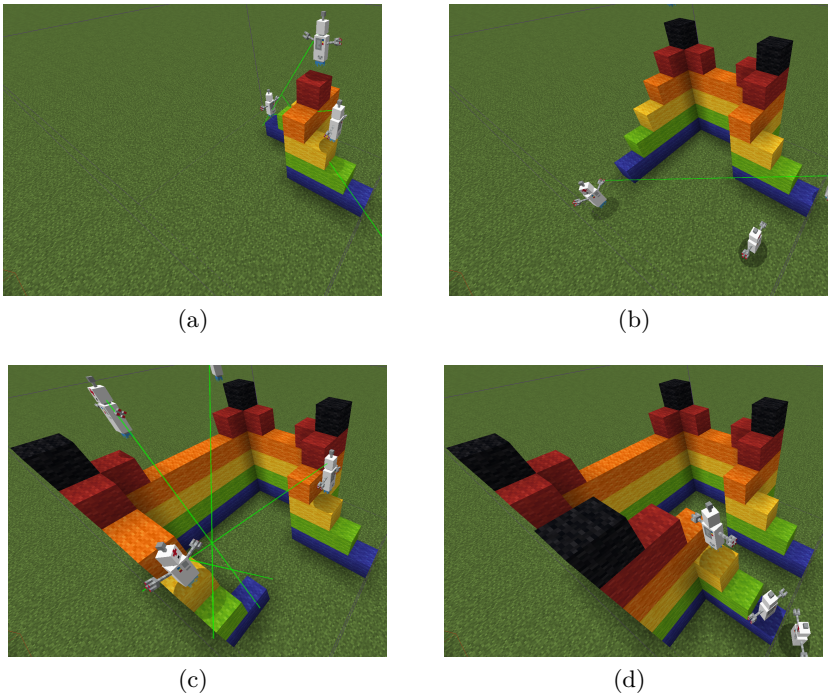
Heuristické přístupy typicky nejprve najdou možnou cestu strukturou a následně pomocí algoritmu podobnému algoritmu 1 vysílají po jednom roboty umístit blok na nějaké místo. Část, kdy se hledá vhodná cesta je nazývána jako kompilace, zároveň je to výpočetně nejnáročnější část.

Již práce TERMES [6], která přišla s funkčními roboty, poskytuje jednoduchý heuristický přístup – vybere nějaké dostupné místo jako start a pomocí vyhledávání do hloubky [4] se snaží najít takovou cestu, která pokryje celou strukturu s tím, že z cílového místa lze strukturu opustit. Po zkompilování dostanou roboti přidělenou výslednou cestu a vykonávají algoritmus 1. Tento přístup ovšem nezvládne postavit všechny možné struktury – ze struktur na obrázku 2.3 zvládne pouze struktury 1 a 4. Ostatní potřebují přistavit lešení pro dosažení vyšších míst. Struktura 2 má kromě potřeby lešení ještě problém s tím, že není možnost zvolit dva body tak, aby mezi ně šla dát cesta, která by pokryla všechna potřebná místa.

S odlišným heuristickým přístupem přišla práce [7]. Tento přístup vytváří minimální kostru grafu [4] struktury kde hrany jsou ohodnoceny rozdílem výšek vrcholů. Zároveň je stavební plocha rozšířena do stran, aby bylo dost místa pro stavění lešení (Samotný sloupec velikosti n potřebuje nájezdovou rampu (lešení) o délce $n - 1$). Roboti se následně po struktuře pohybují pouze po hranách této kostry a bloky umísťují podle vytvořeného plánu, který se snaží minimalizovat počet cest do zásobárny bloků. Roboti se tedy snaží v částech stavby dočasně používat stavební bloky jako lešení, které časem umístí na správné místo.

Ačkoliv tato metoda zvládá postavit všechny zadané struktury, nese s sebou jisté nevýhody. Vzhledem k tomu, že cestování je povoleno jen po hranách kostry, na stavbu je naráz vpuštěno jen málo robotů, kteří nechodí po optimálních cestách, tudíž stavba může trvat delší dobu.

S podobným přístupem jako práce TERMES [6] přišla práce [11], která vytvořila rychlý, snadno modifikovatelný kompilátor BFD.



(a) (b) (c) (d)

Obrázek 2.4: Ilustrace stavby hradu z obr. 2.1 pomocí TERMES algoritmu [6] (vlastní)

2.1.3.1 BFD kompilátor

Tento kompilátor zvládne zkompilovat struktury velké přes milion bloků v řádu sekund (tvůrci uvádějí, že jeho časová složitost je skoro lineární [11]). BFD vybere 2 pozice jako start a cíl a následně vytvoří orientovaný graf od startu k cíli, po kterém se roboti mohou pohybovat.

Roztržení struktury znamená, že ze startu nadále nejsou dosažitelné všechny bloky z $L \backslash Q_{visited}$. Toto ověření se dělá pomocí kostry grafu [4] $L \backslash Q_{visited}$. Tato kostra se vytvoří pomocí hledání do šířky spuštěného ze startu. Pokud je vrchol k odstranění listem v kostře, může být odstraněn hned, pokud listem není, vytvoří se nová kostra stejným způsobem a zkontroluje se, zdali obsahuje všechny nenavštívené vrcholy.

Jelikož vytvořený graf může mít rozcestí, roboti se řídí modifikovaným algoritmem 1. Na rozcestích si mohou náhodně vybrat kudy půjdou, nebo může mít každé rozcestí upravené šance na zvolení každé cesty pro zrychlení konstrukce.

Algoritmus 2: BFD kompilátor [11]

```
1  initialize Q_frontier and Q_visited to be empty
2  initialize map to be an empty graph over the vertex set L
3  add L_EXIT to Q_frontier
4  while Q_frontier is not empty do
5      remove l_0 from Q_frontier
6      if l_0 is not in between two other unvisited sites and removing l_0
         does not disconnect the structure then
7          Add l_0 to Q_visited
8          for each unvisited neighboring site l_i of l_0
9              add edge(l_i, l_0) to map
10             if path l_i to l_0 is traversable then
11                 add l_i to Q_frontier
12 if |Q_visited| = |L| then
13     return map
14 else
15     return False
```

Navzdory rychlosti kompilace algoritmus neumí přidávat lešení a robotům se může stát, že projdou celým grafem bez nalezení vhodného místa pro umístění bloku (v nejhorších případech mohou roboti projít strukturu o 406 blocích skoro 200 000krát bez položení bloku [11]). Ačkoliv oproti naivnímu kompilátoru z práce [6] zvládne postavit více struktur, stále nedokáže postavit všechny, mimo absence lešení kvůli podmínce, že je zakázáno dávat blok mezi dva jiné bloky. Tato podmínka může být nahrazena následující podmínkou: *blok nesmí být položen na takové místo, které je jediným odchozím místem jiného místa, na kterém by měl být položen blok, ale není.*

Z algoritmu 1 víme, že na položení bloku je potřeba jedno volné místo za místem, kam se blok chytáme položit. Uvažujme tři místa A, B, C, vedle sebe v tomto pořadí. Pokud je na A a C blok, tak buď byl nějaký blok položen v rozporu s náhradním pravidlem (z B vede jen jedna cesta do vedlejšího bloku), nebo z B vede alternativní cesta kolmo na naše tři místa do místa D, ze kterého můžeme položit blok na místo B.

Vzhledem k jednoduchosti a rychlosti algoritmu v dalších kapitolách upravíme tento algoritmus tak, aby byl schopen postavit libovolnou zadanou strukturu a zároveň si zachoval dobré škálování na velké struktury.

2.2 Minecraft

Minecraft je sandboxová hra z roku 2011 [2] a momentálně nejprodávanější hra vůbec [12]. Hráč se zde ocitne ve 3D světě z kostiček, kde každá kostička (dále blok) může být zničena a obdobně je možné libovolné bloky kamkoliv umístit (jsou zde zanedbatelné výjimky). Ve světě se vyskytují i různé příšery, či zvířata (dále jen entity), které je možno využít pro získání surovin.

Obrázek 2.5: Minas Tirith v Minecraftu (převzato z Minecraft Middle Earth `https://www.mcmiddleearth.com/`)

Až na pár výjimek umístěné bloky nepodléhají gravitaci, mohou se tedy zůstat vznášet ve vzduchu, oproti tomu entity gravitaci podléhají všechny.

Jelikož je možné hru rozšiřovat (dále modovat), nabízí zajímavé simulační prostředí pro robotickou konstrukci.

Od vydání Minecraftu vzniklo několik verzí na různé platformy [2]. V kontextu modování se ovšem o jiné než Java verzi moc nemluví.

2.2.1 Modování Minecraftu

Modování Minecraftu je proces, při kterém hru rozšíříme o nějaké vlastní prvky, například nové entity, nebo upravíme nějaká stávající pravidla hry. V našem případě chceme přidat novou entitu (robot), umělou inteligenci a rozhraní, přes které budeme tyto roboty ovládat.

Existuje několik možností, jak vytvářet mody. Například Bukkit [13] nabízí možnosti pro servery. Bukkit je speciální Minecraft server, na který je možné nahrávat pluginy. Tyto pluginy jsou tedy mody, ovšem nabízejí jen možnosti úpravy serveru, mezi které nepatří vytváření vlastních entit.

Pro vytvoření „plnohodnotného“ modu, který může pozměnit vše, je potřeba možnost úpravy i klientské části hry. Tomuto momentálně vévodí dvě platformy a to Minecraft Forge [14] (dále jen Forge) a Fabric [15].

Oba tyto nástroje fungují jako tzv. Mod Loader a i poskytují vlastní API na vytváření modů. Mod Loader je technologie, která umožňuje snadné přidávání modů do hry jako takové. Při startu hry do ní načte všechny mody v *.jar* podobně, vytvořené s příslušným API z dané složky.

Hlavní rozdíl mezi nimi je, že Fabric je relativně nový a experimentální [15], zatímco Forge je momentálně populárnější s větší modderskou základnou. Proto považuji Forge jako ideální volbu pro tuto práci.

2.2.2 Minecraft jako simulační prostředí

Roboti jsou základním pilířem stavby, do Minecraftu by byly přidáni jako entita. Typická entita v Minecraftu zvládne vyskočit do výšky jednoho bloku (tudíž naráz překoná stejnou vertikální vzdálenost jako TERMES robot [6]), zná svojí pozici ve světě a má své cíle. Každá entita se své cíle snaží plnit, například *Zombie* má jako cíl pronásledovat *Vesničany (Villager)* [2].

Cíl se v první řadě snaží aktivovat, na to je metoda *shouldExecute*, která se ve smyčce snaží spustit daný cíl. Pokud uspěje, další na řadě je metoda *startExecuting*, která cíl inicializuje a následně se ve smyčce volají metody *tick* a *shouldContinueExecuting* (v tomto pořadí). Metoda *tick* má za úkol provést část cíle (pohni se daným směrem) a druhá metoda ukončuje cíl, buď je hotov, nebo již nelze splnit [2].

Jednotlivé entity jsou schopny odhalit přítomnost jiných v okolí, ovšem na zjišťování robotů blokujících cestu to nestačí, proto by bylo ideální mít nadřazenou entitu hlídající kolize. Dalším důvodem pro volbu hierarchického systému je fakt, že se musí vytvořit plán stavby, který by tato nadřazená entita měla za úkol.

Pokládání bloků se může simulovat manuálně v kódu – pokud robot dorazí na místo položení bloku, přidá se na dané místo blok.

Struktury k postavení a kompilace struktur mohou zajišťovat vlastní příkazy.

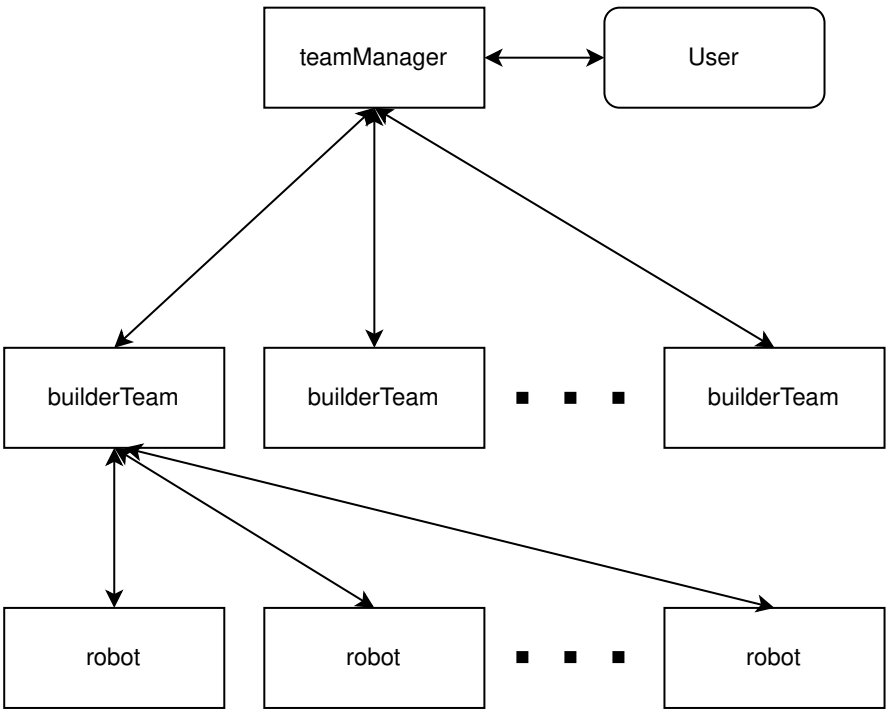
Kapitola 3

Návrh vlastního systému

Kapitola probere návrh nového systému na kolektivní konstrukci. Počínaje samotným systémem, přes roboty, a nakonec samotný kompilátor a jeho dílčí součásti.

3.1 Hierarchie

Systém pro stavění bude mít hierarchickou podobu jako na obrázku 3.1.



Obrázek 3.1: Hierarchický systém

Uživatel skrz příkazy komunikuje s manažerem. Manažer je nejvyšší entita v hierarchii, obsahuje stavitelské týmy, obsluhuje přidělování stavebních oblastí a robotů. Každý tým obsahuje a obsluhuje přidělené roboty, je zodpovědný za rozdávání úkolů pro roboty a vytvoření stavebního plánu. Jednotliví roboti pouze dostávají úkoly od nadřazeného týmu, které vykonávají.

3.2 Roboti

Roboti budou následovat vlastnosti robotů TERMES [6], budou moci vylézt o blok výše (a níže), pohybovat se a pokládat bloky, pokud je za nimi volné místo. Vzhledem k tomu, že položení bloku nelze simulovat jinak než manuálním přidáním bloku do hry, budou roboti pro usnadnění implementace pokládat blok pod sebe (při položení vyskočí a tím se pod nimi udělá místo, eliminuje se tím potřeba pamatovat si předchozí lokaci a zároveň ulehčí řešení kolizí).

Jako cíl bude mít entita robota za úkol získat od svého týmu místo kam blok položit, cestu strukturou na toto místo a následnou cestu pryč ze struktury zpět k zásobárně bloků. Oproti TERMES robotům budou roboti svou cestu strukturou měnit při každém navštívení zásobárny bloků, to eliminuje problém BFD algoritmu, kdy roboti mnohokrát prošli strukturou bez umístění bloku [11]. Robot tedy dostane přidělený cíl od nadřazené entity, který následně začne vykonávat.

Vyhýbání kolizím bude řešit nadřazená entita.

Algoritmus 3: Cíl pro řízení robotů v Minecraftu

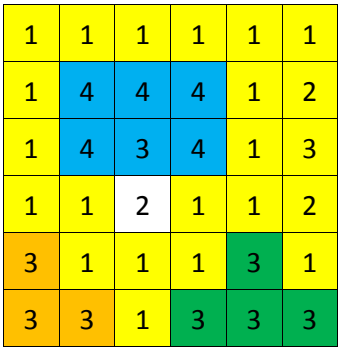
```
1 let Q be assigned path
2 let T be assigned location to place a block
3 while not at the end of Q do
4     while robot detected ahead do
5         wait
6     follow Q
7     if holding block and at T then
8         jump
9         place block at T
```

3.3 Možné problémy kompilace

Kromě potřeby lešení pro nedostupná místa skýtá stavba i další problémy, kterým je potřeba věnovat pozornost. Struktrua 5 (obr. 3.2) ilustruje dvě hlavní kategorie problémů – komponenty souvislosti [4] a křižovatky. Oba tyto problémy není algoritmus BFD (2) schopen vyřešit.

1	1	1	1	1	1
1	4	4	4	1	2
1	4	3	4	1	3
1	1	2	1	1	2
3	1	1	1	3	1
3	3	1	3	3	3

(a) Struktura 5 – komponenty



(b) Struktura 5 barevně

Obrázek 3.2: Struktura s několika komponentami

3.3.1 Komponenty souvislosti

Graf struktury 5 má celkem tři komponenty souvislosti. Zelená a oranžová jdou triviálně odhalit a mohou být stavěny nezávisle na zbytku struktury (nejdříve postavit zelenou, poté oranžovou, a nakonec zbytek). Tento přístup, který staví komponenty zvlášť je potřebný například při stavění struktury, která vypadá jako šachovnice, kde bílá políčka mají výšku 0 a černá výšku 3. Pokud by se vše stavělo najednou, tak by nakonec nešlo odebrat přidané lešení. Stavět komponenty zvlášť je tedy nutnost (pokud nenajdeme 2 komponenty, které mohou být stavěny bez překážení si).

Ačkoliv modrá část struktury patří k žluté komponentě, nejde stavit naráz, jelikož se do modré části lze dostat pouze přes bílé políčko a následně by roboti neměli jak tuto část opouštět. Buď je potřeba přidat lešení, nebo brát modrou část jako samostatnou komponentu. Ovšem oproti samotným komponentám tato část nelze odhalit triviálně.

3.3.2 Křižovatky

Další z problémů se týká zelené části, která se větví podobně jako struktura 2 (obr. 2.3). U tohoto tvaru je potřeba více cílových bloků, aby mohl být plynule zkompilován, nebo alternativně se může kříž rozdělit na více komponent sestávající z jednotlivých „částí“ kříže.

Více cílových bloků nemusí pomoct vždy. Pokud budeme uvažovat strukturu o výšce dva, vypadající jako písmeno Q (řekněme, že písmeno Q je definováno jako kolečko a ocásek), tak jeden z těchto „slepých konců“ bude na ocásku uvnitř kolečka a nepůjde postavit spolu se strukturou. Je potřeba brát jako samostatná komponenta, která se musí postavit dříve.

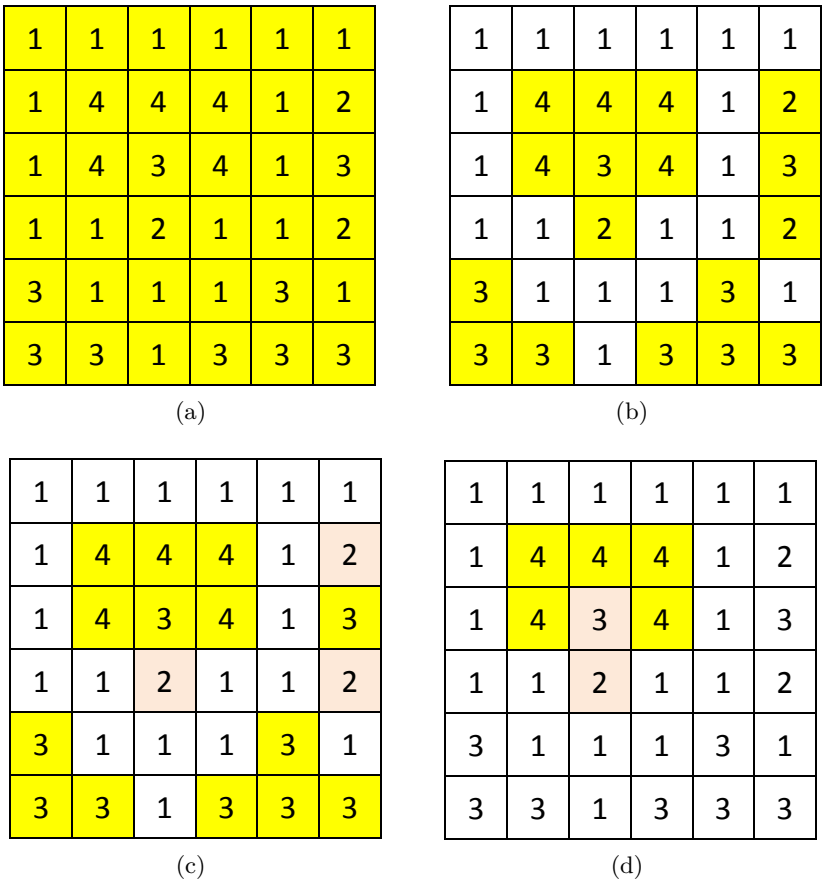
Také pokud bychom propojili tři struktury cestou o šířce 1, dostaneme podobnou situaci jako s modrou částí struktury 5. Je tedy potřeba hlídat vrcholy, ze kterých vedou jen dvě hrany.

3.4 Bottom-up kompilátor

Je vidět, že celý proces by mohl být optimalizovaný na postavení základny o výšce 1 a až poté stavit jednotlivé komponenty (a přidávat lešení). Při uvažování samotné oranžové části (obr. 3.2), a pokud by byla stavěna první, bylo by potřeba $2*(2+1)$ bloků lešení. Pokud by již existovala základna struktury, bloky lešení by byly potřeba jen dva. Toto lze zobecnit na to, že je výhodnější (z hlediska lešení) stavit dvě dotýkající se komponenty (mající stejnou základnu) naráz.

Toto tvrzení dále podporuje fakt, že u plánů vytvořených v pracích [6] a [11] a následném použijí algoritmu 1 se bloky staví nejprve do výšky a až poté do šířky (roboti pak musí urazit delší vzdálenosti (nejprve musí přelézt kopec), ilustrováno na obr. 2.4).

Tento postup nejenže sníží počet potřebných lešení a urychlí pohyb robotů, zároveň také usnadní hledání komponent (snadno odhalí modrou komponentu z obr. 3.2).



Obrázek 3.3: Postup stavby struktury 5 zdola nahoru

Při každém navýšení základny se zkontrolují jednotlivé komponenty a následně se každá komponenta zvlášť postaví. Zároveň je potřeba nalezené komponenty řadit tak, aby se nestalo, že se k nějaké následně nepůjde dostat.

Nevýhody tohoto přístupu jsou, že se musí počítat nový stavební plán pro každou komponentu a každou výšku, což může značně zvýšit čas potřebný ke kompilaci. Ovšem pokud se spočítá první část plánu, roboti ji mohou ihned začít stavět, zatímco se bude dále provádět výpočet. Než roboti postaví danou část dá se očekávat, že již bude proveden výpočet pro část další (vycházíme z toho, že BFD kompilátor (alg. 2) je schopen vytvořit plán v rámci sekund [11]).

Algoritmus 4: Bottom-up kompilátor

```
1  j ← structure height to start at, defaultly 0
2  h ← max structure height
3  X ← structure component used, defaultly the whole structure
4  for i from j to h do
5      exclude sites from X with height less than i
6      C ← compute components of X
7      sort C by availability
8      for each component c in C
9          create plan p for c                  // robots can start working on c
10         run compiler recursively with X ← c and j ← i + 1
11         remove any scaffolding constructed by p
```

Algoritmus Bottom-up kompilátoru řeší hlavně správné dělení komponent souvislosti, kromě toho je i potřeba komponenty seřadit a vytvořit plán pro stavbu.

3.4.1 Dělení komponent

Prvotní dělení ignoruje bloky do dané výšky a zbytek bloků rozdělí na komponenty souvislosti pomocí jednoduchého hledání do šířky. Následně, aby se předešlo křižovatkám se každá komponenta rozdělí (pokud možno) na další komponenty. Pro každou komponentu se nejprve naleznou mosty jejího grafu [4]. Poté se prochází přes jednotlivé mosty, a pokud mají oba vrcholy, které daný most propojuje stupeň nejvýše dva, most se zachová, jinak se odstraní. Díky to tomu se odstraní slepé uličky a problémové křižovatky.

Hlavní dva druhy vznikajících komponent jsou úzké cesty o šířce 1, které mají označená místa pro start a cíl (konce této cesty), a komponenty, u kterých na místu výběru startu, či cíle nezáleží (z hlediska postavitelnosti).

3.4.2 Řazení komponent

Nesmí se stát, že by se nejprve postavily takové komponenty, jejichž stavba by znemožnila stavbu jiných. Každá komponenta si bude uchovávat číslo udávající nejvzdálenější bod od středu. Podle tohoto čísla se následně budou stavět, kdy komponenty nejblíže ke středu budou stavěny první. Toto zároveň řeší případ, kdy je jedna komponenta uvnitř jiné. Tato „jiná“ komponenta bude mít vždy jednu část dále od středu než komponenta vnitřní.

Ačkoliv toto řazení udá pořadí zajišťující dokončení stavby, mohou nastat případy, kdy jiné řazení by vedlo k optimálnějším cestám a rychlejšímu průběhu stavby.

Kromě optimalizace řazení je zde i dobrá možnost paralelizace, pokud by se našly dvě komponenty, které spolu nijak neinteragují. Například struktura 1000x1000, která má v každém rohu věž – věže by šli stavět naráz.

3.5 Plán stavby

První krok k vytvoření plánu stavby pro libovolnou komponentu je nalezení vhodných bloků pro start a cíl. Robot při sebrání bloku ze zásobárny nejprve dojde na start, poté prochází strukturou, kde se snaží umístit blok a nakonec strukturu skrze cíl opustí.

Pro zjednodušení problému budou roboti vždy začínat na souřadnicích $(0, 0)$ a končit v protějším rohu. Musí tedy mít z prvního rohu cestu na start a následně cestu skrze cíl do druhého rohu. Na toto hledání se dá využít Dijkstrův algoritmus [4], najde nejkratší cestu v ohodnoceném grafu. Možností jak ohodnotit hrany je více, problémová jsou zejména místa, kde je potřeba přistavit lešení. Pokud lešení potřeba není, váha hrany může být 1. Oproti tomu, když potřebujeme přistavit lešení mezi dvě místa A, B, bude potřeba nejméně $\frac{|B-A|*(|B-A|-1)}{2}$ bloků lešení. K tomu se dále může připočítat cesta robota s lešením, odstranění lešení, a naopak se může brát v potaz o kolik by blok lešení mohl zkrátit cestu, kterou se následně může dále chodit. Zde byl zvolen dvojnásobek minimálního počtu bloků lešení – každý blok lešení se musí přinést a zároveň odnést. Zároveň má hledací algoritmus dovoleno chodit pouze nahoru, aby se nedostal do místa, kde je potřeba lešení, které by na daném místě nešlo postavit.

Pro komponenty, které nemají předem daný start a cíl se hledací algoritmus pustí obdobně s tím, že jako cíl má hledací algoritmus zadán libovolný blok komponenty, který se následně příslušně označí jako start, či cíl.

Poté, co jsou vybrány pozice pro start a cíl, spustí se algoritmus BFD (alg. 2) (s upravenou podmínkou z minulé kapitoly). Výsledný graf, který BFD vytvoří se propojí s cestami k zásobárně bloků.

3.5.1 Lešení

Pro zjištění, kam má být postaveno lešení stačí projít vytvořené cesty od struktury ke kraji a přidávat lešení tak, aby byl rozdíl výšky každých dvou bloků menší nebo roven jedné. Vzhledem k tomu, že hledací algoritmus může pouze na místa vyšší, nebo stejně velká lešení se budou stavět vždy mimo aktuálně stavenou část struktury.

Lešení se následně postaví odzdola nahoru před bloky struktury. Vzhledem k tomu, že komponenta, ke které je lešení přistavěno může být jedině vyšší, lešení se odebere až po dostavění dané komponenty do maximální výšky. Tím se zabrání opakovanému pokládání lešení na stejné místo.

Aby na lešení bylo vždy dost místa, tak před spuštěním celého kompilátoru se rozšíří stavební plocha po vzoru [7] tak, aby od každého bloku bylo možné natáhnout dostatečně dlouhou rampu (lešení) k nejbližšímu hraji.

Nakonec stačí robotům jen lešení umístit ve správném pořadí. To jest vždy umístit lešení na první blok, který za sebou nemá volné místo. Toto funguje jen pro lešení směrem ke struktuře, směrem od struktury lze použít pro umístění algoritmus TERMES robotů 1.

3.5.2 Seřazení bloků

Po konstrukci lešení mohou roboti začít stavět strukturu. Aby se předešlo tomu, že roboti projdou cestu bez položení bloku (jeden z hlavních problémů práce [11]). Roboti budou při sebrání bloku instruování místem, na které blok mají doručit. Pro snadné určení dalšího místa na položení bloku se může graf z výstupu BFD topologicky seřadit [4]. Tím bude zajištěno, že každý blok bude mít místo, ze kterého půjde bezpečně položit. Roboti také nemusí zbytečně procházet celou strukturu, stačí najít nejkratší cestu k místu položení bloku a následně pryč.

Kapitola 4

Implementace modu

Kapitola probírá implementaci modu a jeho částí. Mod je implementován v Minecraft Forge API [14], verze 1.16.3-34.1.0.

4.1 Týmový Manažer

Manažer, jakožto nejvyšší entita v hierarchii je implementován jako statická třída s rozhraním na komunikaci a ovládání jednotlivých týmů. Všechny týmy si ukládá do statického pole, zároveň obsahuje vzor struktury, která má být postavena. Naráz tedy může být stavěna několikrát pouze jedna struktura.

Po startu hry (resp. načtení světa) vytvoří výchozí tým, který slouží jako skladiště pro nepřiřazené roboty a zároveň všechny roboty v daném světě do tohoto týmu přidá. Po odchodu ze světa všechny týmy rozpustí, aby se záznamy o robotech nepřenášely mezi různými světy.

4.2 Stavební týmy

Jednotlivé týmy (resp. entita týmu) mají dva hlavní úkoly – vytvořit plán stavby a přidělovat úkoly robotům.

Po startu stavby, který je zahlášen z *manageru* (je vytvořeno nové vlákno, aby nebyly blokované příkazy během stavby), entita nejprve zkompiluje stavbu, následně dá znamení robotům, že začala stavba a mohou si říkat o úkoly, a uspí hlavní vlákno stavby, dokud stavba není hotová.

Jednotliví roboti si poté žádají o cíl a entita jim vybírá blok, který umístit a zároveň vypočítá cestu strukturou. Po umístění posledního bloku (nebo sebrání posledního lešení) a vyklizení všech robotů z oblasti stavby probere hlavní vlákno, které zahlásí dobu trvání stavby a ukončí se.

Ačkoliv se dělají jednoduchá ověření validity pro oblast stavby, je zejména na uživateli, aby zajistil její dostupnost a dostatečný prostor pro manévrování.

K dispozici jsou následující stavební týmy:

ID	popis
dummy	4.2.1
naive_termes	4.2.2
bottom_up_compiler	4.2.3
bottom_up_fast	4.2.4

Tabulka 4.1: Seznam stavebních týmů

4.2.1 Tým *dummy*

Výchozí tým pro *teamManager* (4.1), jen svolává roboty na jedno místo. Nic nestaví.

4.2.2 Tým *naive_termes*

Implementuje naivní TERMES kompilátor [6] a zároveň demonstruje funkčnost robotů. Při velkém počtu robotů dojde k zahlcení u vstupu a nejspíše se nepodaří úspěšně dokončit stavbu. Řešení zahlcení se nestihlo zapojit.

4.2.3 Tým *bottom_up_compiler*

Implementuje samotný Bottom-up kompilátor. Tým jako takový nic nestaví, nýbrž pouze zkompiluje zadanou strukturu a vytvoří kompletní plán stavby. Následně vypíše čas kompilace, čas kompilace 1. plánu, počet potřebných bloků, počet potřebných lešení a odhad práce.

4.2.4 Tým *bottom_up_fast*

Tým zkompiluje zadanou strukturu pomocí Bottom-up kompilátoru a následně nasimuluje průběh stavby. Výsledkem tedy je postavená struktura a je i vidět její průběh. Tento tým lze snadno modifikovat pro (méně) detailnější simulaci, či rychlost simulace.

4.2.5 Tým *bottom_up*

Samotný tým, který by roboty přímo používal místo simuloval se nestihl. Průběh stavby demonstruje tým 4.2.4 a použití robotů demonstruje tým 4.2.2. Na případné dokončení tohoto týmu je potřeba implementovat metodu předávající cíle robotům (rozšíření simulační verze).

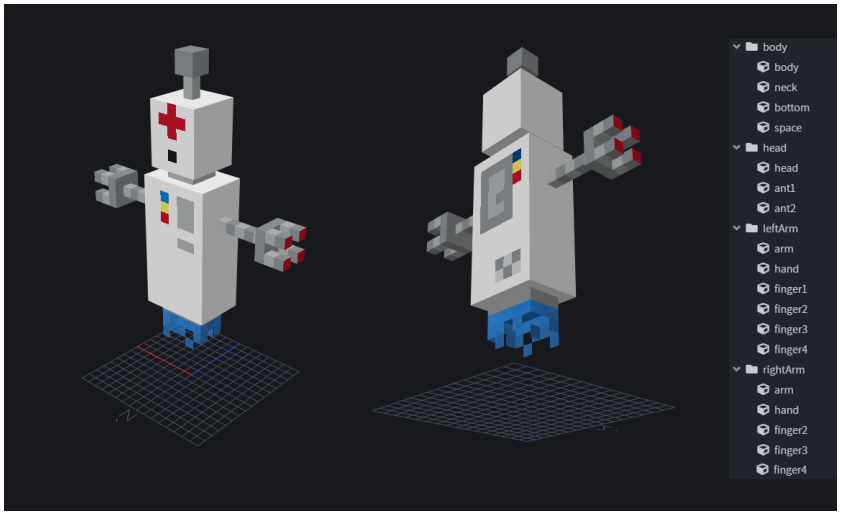
4.3 Roboti

Pro úspěšné přidání nové entity (robota) do hry je potřeba vytvořit tři třídy – model, render a třídu pro entitu jako takovou [16, 2, 14].

4.3.1 Model

Model obstarává, jak bude entita vypadat ve hře. To obnáší texturu, rotace paží a hlavy. V kódu je potřeba jen vyplnit zmíněné rotace, o zbytek se postará nástroj Blockbench [17]. To je editor, ve kterém se postupně entita namodeluje a následovně sám vygeneruje použitelný kód.

Rotace jsou nastavené tak, aby robot koukal na cíl a rukama mával a točil při pohybu. Nohy nemá, místo nich má trysku.



Obrázek 4.1: Robot v nástroji Blockbench [17] (vlastní)

4.3.2 Render

Renderovací třída se stará o zobrazení modelu ve hře. Prakticky tato třída pouze dědí ze třídy *MobRenderer* [2] a volá nadřazenou metodu *render*. Navíc se zde přidal paprsek, kterým robot ukazuje místo, na které plánuje umístit blok.

4.3.3 Entita

Třída pro entitu jako takovou obsahuje údaje jako počet životů, rychlost atp. Zároveň také cíle entity. Hlavní cíl robota pro komunikaci s nadřazenou entitou se aktivuje po zahájení stavby.

Během doby, kdy robot nestaví pouze čeká, ovšem je možnost dovolit mu volně se pohybovat po okolí pomocí příkazu *allowFreeWill <boolean>*. Toto má jen čistě estetický význam, aby roboti vypadali živěji, ovšem může to způsobit, že se roboti připletou na nechtěné místo.

Pokud je entita příliš daleko od hráčů, tak pozastavuje svoji činnost. Z toho plyne, že je potřeba být nablízku robotům, kteří mají stavět.

4.4 Příkazy

Příkazy se zadávají do chatu ve hře. Seznam dostupných příkazů je v následující tabulce.

název	popis
showTeams	4.4.1
addRobots	4.4.2
moveRobots	4.4.4
addTeam	4.4.3
activeTeam	4.4.5
pos1	4.4.6
pos2	4.4.6
setArea	4.4.7
startConstruction	4.4.8
showTarget	4.4.9

Tabulka 4.2: Seznam příkazů

4.4.1 showTeams

Zobrazí všechny konstrukční týmu spolu s počtem robotů a aktuálním stavem ((ne)může stavět / staví).

4.4.2 addRobots <int>

Přidá do hry počet robotů z parametru. Všichni roboti jsou automaticky přidáni do nultého týmu.

4.4.3 addTeam <string>

Vytvoří nový konstrukční tým daného ID. Jednotlivé týmy popsány v tabulce 4.1.

4.4.4 moveRobots <from> <to> <how_many>

Přesune požadovaný počet (*how_many*) robotů z týmu *from* do týmu *to*. Všechny parametry jsou čísla.

4.4.5 activeTeam <int>

Příkaz ovládá proměnnou *activeTeam*. Tato proměnná je používána v ostatních příkazech, udává aktivní tým.

Bez parametru vypíše aktivní tým, s parametrem změní aktivní tým na požadovanou hodnotu.

4.4.6 pos1, pos2

Příkazy nastavují rohy oblasti, kterou chceme použít jako vzor stavby.

4.4.7 setArea

Nastaví stavební oblast pro aktivní tým. Do této oblasti tým postaví vzor stavby. Vzor stavby musí být nastaven.

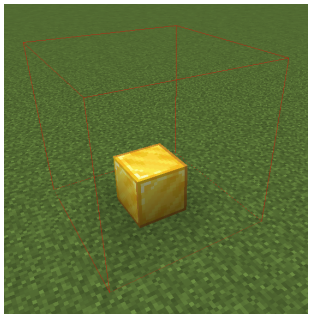
4.4.8 startConstruction

Zahájí stavbu pro aktivní tým (pokud možno).

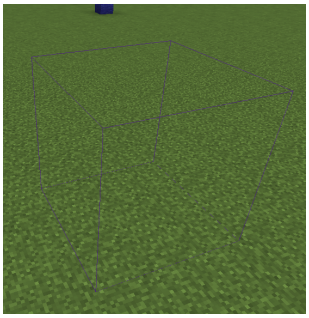
4.4.9 showTarget <bool>

Vypne/zapne paprsek, kterým jednotliví roboti zobrazují svůj aktuální cíl při stavění.

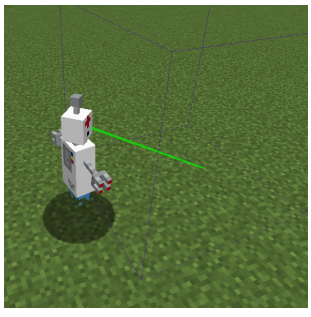
Bez parametru přepne z *true* na *false* a naopak.



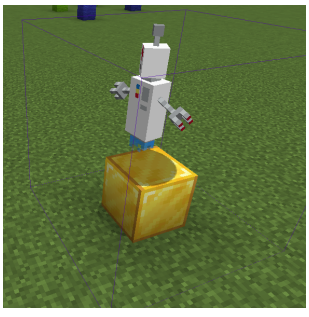
(a) Oblast označená jako vzor



(b) Oblast pro stavbu



(c) Robot směřující k cíli



(d) Robot položil blok

Obrázek 4.2: Ukázky implementace ze hry

Kapitola 5

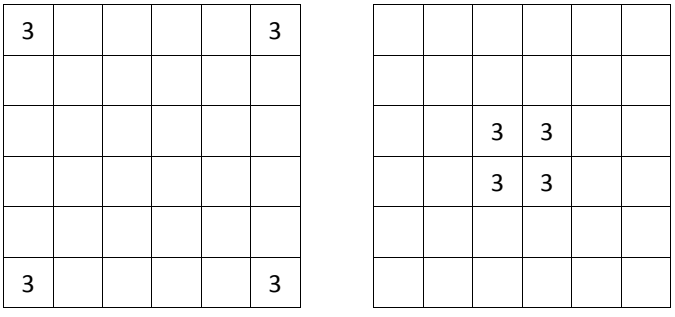
Experimentální sekce

V experimentální sekci se zaměříme na výkon Bottom-up kompilátoru v kontrastu s plánovacími a následně i ostatními heuristickými přístupy. Lze se zaměřit na dobu kompilace, škálovatelnost, či akce robotů (jako akce je brán pohyb, či pokládání bloku). Pro získání dat byl použit samotný kompilátor bez stavění (viz. tabulka 4.1), výsledný čas byl vždy měřen jako průměr z pěti, počet akcí je odhadem délkou cesty, počtu potřebných bloků a lešení; je to skutečně odhad, reálná hodnota bude nejspíš vyšší.

Všechny experimenty jsou prováděny na stolním počítači s procesorem Intel Core i5-7500 @ 3.40 GHz, 16 GB RAM.

5.1 Ověření funkčnosti

Pro ověření funkčnosti, že kompilátor opravdu produkuje validní mapy je použit *bottom_up_fast* tým, který nasimuluje jednotlivé kroky stavby, jako kdyby ji stavěli roboti. Kompilátor vždy vytvořil validní plán, ilustrace plánů krok po kroku jsou k nalezení v příloze.



(a) Struktura 6 – věže (b) Struktura 7 – pařez

Obrázek 5.1: Struktury 6 a 7

5.2 Porovnání s plánovacími přístupy

Pro porovnání s plánovacími přístupy je kompilátor puštěn na stejné struktury, které jsou použity v práci [8], to jsou struktury 2,3 a 4 z obrázku 2.3 a dále struktury 6 a 7 z obrázku 5.1.

Jelikož se v práci [8] ukázalo, že řešení pomocí MILP je o mnoho rychlejší než CP, není CP brán v potaz. V následující tabulce jsou zachyceny výsledky kompilace pro zmíněné struktury a počty akcí robotů. Sloupeček *akce MILP* udává celkový počet akcí robotů spolu s celkovou dobou trvání.

struktura	bloků	čas MILP	čas BU	akce MILP	akce BU
kříž (2)	33	5.7 d	0.011 s	368 / 17	1128
hradba (3)	28	29 s	0.025 s	176 / 11	953
pyramida (4)	56	1.2 h	0.003 s	344 / 13	98
věže (6)	12	3 s	0.009 s	128 / 11	1512
pařez (7)	12	183 s	0.001 s	234 / 15	267

Tabulka 5.1: Porovnání Bottom-up kompilátoru s MILP modelem [8]

Podle očekávání heuristický přístup zvládne kompilaci problému mnohem rychleji, oproti tomu plánovací nachází optimální řešení. Struktury 3 a 6 zvládne MILP postavit rychleji, vzhledem ke krátké době kompilace. Plánovací přístupy ovšem špatně škálují na větší problémy, už struktura 2 zabere 5 dní kompilace. Mimo to je zde vynechána struktura, jenž je 3x3x3 krychle. Doba kompilace této krychle pro MILP trvá 5 hodin, jak se později ukáže, BU kompilátor zvládne zkompilovat větší krychle za menší čas.

Plánovací přístupy zároveň řeší optimální počet robotů na stavbu. Pro strukturu 4 je podle MILP ideální počet 44, což je počet, který si heuristické modely na takto malé struktuře nemohou dovolit, poněvadž jsou vypouštěni za sebou a takovéto množství by zahltilo celou stavbu.

Dále si lze zde všimnout nepřesnosti odhadu akcí BU kompilátoru, pro pyramidu udává celkový počet akcí menší, než optimální MILP).

5.3 Porovnání s heuristickými přístupy

Hlavním konkurentem BU kompilátoru je BFD [11], ze kterého vychází. Jelikož BFD nezvládá stavění lešení, zbývá porovnání ve škálovatelnosti. Test škálovatelnosti je proveden na krychlích. Počáteční velikost krychle je 10x10x10 a následně se zvětšuje až do rozměrů 80x80x80. V tabulce 5.2 jsou zaneseny počty bloků v krychli, počet potřebných lešení, čas kompilace a čas vytvoření prvního plánu.

Čas vytvoření prvního plánu je silná stránka BU kompilátoru. Ve chvíli jeho vytvoření je možné ihned začít stavět, zatím co výpočet probíhá na pozadí

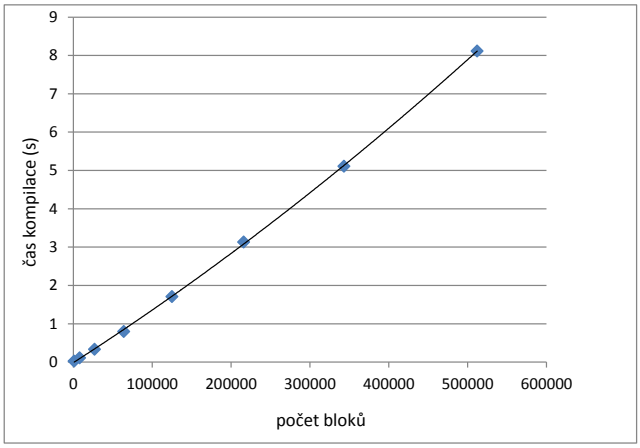
dále. Už u krychle o straně 80 by stavba mohla začít o 6 vteřin před úplným dokončení kompilace.

Na druhou stranu tento test zároveň odhaluje některé implementační nedostatky. Krychle o rozměrech 90x90x90 nebyla úspěšně zkompilovaná kvůli své velikosti – hledání mostů v grafu využívá rekurzi, při čemž došla paměť na zásobníku. Kompilátor je celkově náročný na paměť, což však lze částečně vyřešit tím, že se omezí počet naráz vytvořených plánů.

krychle	bloků	lešení	čas	čas_0
10	1000	90	0.025 s	–
20	8000	380	0.110 s	0.035 s
30	27000	870	0.336 s	0.048 s
40	64000	1560	0.797 s	0.08 s
50	125000	2450	1.708 s	0.08 s
60	216000	3540	3.132 s	0.08 s
70	343000	4830	5.109 s	0.1 s
80	512000	6320	8.116 s	1.4 s

Tabulka 5.2: Demonstrace škálovatelnosti BU kompilátoru

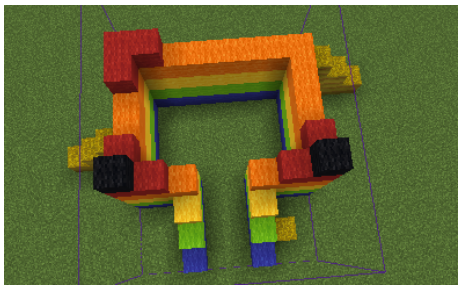
Podle dat z tabulky 5.2 můžeme sestrojit graf škálovatelnosti (obr. 5.2). Z grafu jde vidět, že škálovatelnost je téměř lineární, tudíž si BU kompilátor zachovat velmi dobrou škálovatelnost i přes složitější algoritmus zvládající stavbu lešení.



Obrázek 5.2: Škálovatelnost Bottom-up kompilátoru

Dále lze porovnávat heuristické přístupy v počtu potřebných kroků pro jednotlivé struktury. Hned jako první stojí za zmínku struktura hrad (obr. 2.3). Na její konstrukci není třeba lešení a lze postavit ji postavit i s naivním TERMES kompilátorem (ilustrováno na obr.2.4). Na horní vrstvě hradu

je cimbuří. Cimbuří jako takové nepředstavuje překážku, jelikož jde přelézt, ovšem BU kompilátor zde narazí na problém, jelikož při hledání cesty směrem ke komponentě má dovoleny přechody pouze na bloky stejné, nebo vyšší úrovně. Tudíž bude potřebovat lešení a kvůli tomu na této struktuře skončí nejhůře ze všech heuristických kompilátorů. Toto je ukázáno na obrázku 5.3. (Na tomto obrázku si také lze všimnout bloku lešení u pravého nájezdu na hrad. Tento blok zde byl přidán zjevně proto, aby zkracoval cestu (start či cíl je „nad“ hradem) a bude odstraněn jako poslední.)



Obrázek 5.3: BU kompilátor potřebuje na stavbu hradu lešení

Na takovýchto menších strukturách tedy BU kompilátor může být relativně snadno zahanben, ale s přibývající velikostí struktur se začnou projevovat nedostatky ostatních kompilátorů – je větší šance, že nebudou schopni strukturu zkompilovat, nebo se v případě BFD začne více stávat, že roboti projdou strukturou bez umístění bloku.

5.4 Shrnutí

Nejsilnější stránkou BU kompilátoru je jeho škálovatelnost (struktura o velikosti půl milionu bloků za několik vteřin) a postupné tvoření plánů, díky kterému mohou roboti začít stavět dříve, než se dokončí celá kompilace. Dále je algoritmus relativně jednoduchý a tudíž jde snadno modifikovat a poskytuje mnoho možností pro optimalizace. Zejména heuristika a možnost přejít na nižší blok při hledání cesty strukturou a dále například nestacionární start a cíl. A také nabízí dobré možnosti paralelizace, jak při výpočtech, tak při stavění (práce na dvou komponentách zároveň).

Oproti tomu slabé stránky jsou: malé úzké struktury, na kterých vytváří zbytečně složitý plán (jako třeba zmiňovaný hrad); dekonstrukce lešení, která je prováděna prakticky jedním robotem naráz, jelikož odebráním lešení ničí plynulou cestu strukturou; a relativní závislost celého systému na spolehlivost robotů – určování místa na umístění bloku předem se dostane do potíží ve chvíli, kdy se nějaký robot porouchá cestou k umístění bloku.

Závěr

Cílem práce bylo vymyslet metody pro řízení multi-agentní kolektivní konstrukce na základě rešerše, a následně je implementovat a otestovat v rámci rozšíření (modu) hry Minecraft.

Na základě provedené rešerše na kolektivní konstrukci, byl zvolen heuristický přístup řešení kolektivní konstrukce oproti plánovacímu. Heuristické přístupy zvládnou vytvořit plán pro stavbu mnohem rychleji, avšak nenaleznou optimální řešení a často si neporadí s konstrukcí lešení.

Vzhledem ke struktuře Minecraftu se vytvořil hierarchický multi-agentní systém pro robotickou kolektivní konstrukci. Jednotliví roboti jsou podřízeni nadřazené entitě, která jim udává místo, na které je třeba umístit blok a cestu, po které se na dané místo dostanou. Stěžejní část hierarchického systému je jeho kompilátor, který vytváří plán stavby zadané struktury.

Nový kompilátor vychází z kompilátoru BFD, který má dobré škálování, ale nezvládá postavit všechny možné struktury, hlavně nedovede postavit lešení na nedostupná místa. Tento nový kompilátor si ponechává dobré škálování a zároveň si poradí s lešením. Díky tomu je schopný vytvořit plán pro stavbu libovolné struktury. Kompilátor byl pojmenován Bottom-up, jelikož postupně vytváří plány na stavbu struktury po patrech od základny k vrchu. Kromě rozdělení struktury na patra se dále dělí i samotná patra odstraňováním mostů z grafu patra tak, aby bylo vždy možné projít grafem komponenty plynule bez vracení (například oddělí slepé uličky do vlastní komponenty).

Bottom-up kompilátor se implementoval v Javě pomocí Minecraft Forge API jako mod do Minecraftu. Díky tomu je možno přímo v Minecraftu vybrat a označit nějakou strukturu, vytvořit plán stavby a následně i odsimulovat průběh stavby.

Nakonec se v experimentální sekci ukazuje funkčnost a výkonnost kompilátoru na strukturách v Minecraftu. Kompilátor zachovává skoro lineární škálování BFD (strutury o velikosti přes 500 000 bloků zkompiluje za několik sekund), a navíc díky postupnému tvoření plánu umožňuje začít se stavbou ještě dříve, než je zhotoven kompletní plán pro celou strukturu.

Všechny cíle práce se tedy podařilo naplnit. Do budoucna nabízí Bottom-up kompilátor různé možnosti optimalizací – od vhodné paralelizace stavění jednotlivých částí struktur, po optimalizace pokládání lešení a hledání cest.

Bibliografie

1. BROOKS, R. A.; MAES, P.; MATARIC, M. J.; MORE, G. Lunar base construction robots. In: *EEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*. 1990, 389–392 vol.1. Dostupné z DOI: 10.1109/IROS.1990.262415.
2. *Minecraft* [video game]. Mojang Studios, 2011. Dostupné také z: https://www.minecraft.net.
3. WOOLDRIDGE, Michael J. *An introduction to multiagent systems*. 2nd edition. John Wiley, 2009. ISBN 978-0470519462.
4. MEHLHORN, Kurt; SANDERS, Peter. *Algorithms and data structures: the basic toolbox*. Springer, 2008. ISBN 978-3540779773.
5. SARTORETTI, Guillaume; WU, Yue; PAIVINE, William; KUMAR, T. K. Satish; KOENIG, Sven; CHOSET, Howie. Distributed Reinforcement Learning for Multi-robot Decentralized Collective Construction. In: CORRELL, Nikolaus; SCHWAGER, Mac; OTTE, Michael (ed.). *Distributed Autonomous Robotic Systems*. Cham: Springer International Publishing, 2019, s. 35–49. ISBN 978-3-030-05816-6.
6. PETERSEN, Kirstin; NAGPAL, Radhika; WERFEL, Justin. TERMES: An Autonomous Robotic System for Three-Dimensional Collective Construction. *Robotics: Science and Systems Conference VII*. 2011. Dostupné z DOI: 10.7551/mitpress/9481.003.0038.
7. KUMAR, T. K. Satish; JUNG, Sangmook; KOENIG, Sven. A Tree-Based Algorithm for Construction Robots. In: *Proceedings of the Twenty-Fourth International Conferenc on International Conference on Automated Planning and Scheduling*. Portsmouth, New Hampshire, USA: AAAI Press, 2014, s. 481–489. ICAPS'14. ISBN 9781577356608.

8. LAM, Edward; STUCKEY, Peter J.; KOENIG, Sven; KUMAR, T. K. Satish. Exact Approaches to the Multi-agent Collective Construction Problem. In: SIMONIS, Helmut (ed.). *Principles and Practice of Constraint Programming*. Cham: Springer International Publishing, 2020, s. 743–758. ISBN 978-3-030-58475-7.
9. SIERKSMA, Gerard; ZWOLS, Yori. *Linear and integer optimization: theory and practice*. 3rd edition. CRC Press, 2015. ISBN 978-1498710169.
10. DECHTER, Rina. *Constraint Processing*. Morgan Kaufmann Publishers, 2003. ISBN 978-1558608900.
11. DENG, Yawen; HUA, Yiwen; NAPP, Nils; PETERSEN, Kirstin. A Compiler for Scalable Construction by the TERMES Robot Collective. *Robotics and Autonomous Systems*. 2019, roč. 121, s. 103240. ISSN 0921-8890. Dostupné z DOI: https://doi.org/10.1016/j.robot.2019.07.010.
12. *List of best-selling video games - Wikipedia* [online] [cit. 2021-03-20]. Dostupné z: https://en.wikipedia.org/wiki/List_of_best-selling_video_games.
13. *Bukkit* [online] [cit. 2020-05-12]. Dostupné z: https://dev.bukkit.org.
14. *Minecraft Forge* [online] [cit. 2020-05-12]. Dostupné z: https://minecraftforge.net.
15. *Fabric Minecraft mod development* [online] [cit. 2020-05-12]. Dostupné z: https://fabricmc.net.
16. TECHNOVISION. *Minecraft 1.16: Forge Modding Tutorial - Custom Entities (#18)* [online]. 2020 [cit. 2020-05-12]. Dostupné z: https://www.youtube.com/watch?v=zJkQc2A1PIk.
17. *Blockbench* [online] [cit. 2020-05-12]. Dostupné z: https://blockbench.net.

Příloha A

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `mod` ..... adresář se zkompilovaným modem a instalační příručkou
- `runs`
  - `ilustrace` ..... adresář s ilustracemi průběhu staveb
  - `results.xls` ..... tabulka se všemi výsledky experimentů
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF