České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Šedivý**

Studijní program: Otevřená informatika
Obor: Počítačová grafika a interakce

Název tématu: **Život mravenců - počítačová hra**

Pokyny pro vypracování:

Vyjděte z design dokumentu [1] a navrhněte a implementujte následující součásti hry ANTS-Mravenci:
1) datovou strukturu reprezentující herní svět,
2) pohyb a ovládání hráčova mravence,
3) umělou inteligenci autonomních mravenců,
4) přímou komunikaci mezi hráčovým mravencem a autonomními mravenci.

Volbu datových struktur a algoritmů řádně zdůvodněte. Diskutujte i případné jiné varianty. Navrhněte testovací scénáře a implementaci na nich ověřte.

Seznam odborné literatury:

[1] Šedivý, Martin. ANTS. Design dokument hry. Semestrální projekt. FEL ČVUT, 2015
[2] WERBER, Bernard. Mravenci. Překlad Richard Podaný. 2. vyd. Praha: Knižní klub, 2005, 317 s. ISBN 80-242-1378-8.
[3] MILES, Petr. Mravenci a jejich podivuhodný svět. In: ekolist [online]. 2011-03-03 [cit. 2015-04-13].
http://ekolist.cz/cz/publicistika/priroda/mravenci-a-jejich-podivuhodny-svet
[4] POKORNÝ, Zbyňek. Sociální chování mravenců. In: Chov Zvířat [online]. 2015-01-19 [cit. 2015-04-16].
http://www.chovzvirat.cz/clanek/669-socialni-chovani-mravencu/

Vedoucí: Ing. Petr Felkel, Ph.D.

Platnost zadání: do konce zimního semestru 2016/2017

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6. 10. 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Život mravenců - počítačová hra

Bc. Martin Šedivý

Vedoucí práce: Ing. Petr Felkel, Ph.D.

7. ledna 2016

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Petru Felkelovi, Ph.D. za konzultace a cenné připomínky. Dále děkuji své přítelkyni za podporu při psaní této práce a také svým rodičům za podporu během studia.

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.

V Praze dne 7. ledna 2016 .....

České vysoké učení technické v Praze
Fakulta elektrotechnická

Odkaz na tuto práci

ŠEDIVÝ, Martin. *Život mravenců - počítačová hra: Diplomová práce*. Praha: ČVUT v Praze, Fakulta elektrotechnická, 2016.

Abstract

The aim of this diploma thesis is to design own game engine for realization of video game from the world of ants. The emphasis is on design and implementation of data structure for representation of the game world and on implementation of ant movement which is controlled by the player. Next the attention is paid to artificial intelligence of autonomous ants including their indirect control by the player via his ant.

Keywords video game, game engine, game world, ants, 3D aplication, data structures, OpenGL

Abstrakt

Cílem této diplomové práce je navrhnout vlastní herní engine pro realizaci počítačové hry ze světa mravenců. Důraz je kladen na návrh a implementaci datové struktury pro reprezentaci herního světa a na implementaci pohybu mravence ovládaného hráčem. Dále je věnována pozornost umělé inteligenci autonomních mravenců včetně jejich nepřímého ovládání hráčem prostřednictvím jeho mravence.

Klíčová slova počítačová hra, herní engine, herní svět, mravenci, 3D aplikace, datové struktury, OpenGL

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Předmětem této práce je tvorba počítačové hry zasazené do mikrosvěta hmyzí říše, kde hlavním hrdinou je mravenec. Cílem hry jako takové je představit široké veřejnosti život sociálního hmyzu, tedy mravenců, a to nenásilnou a zábavnou formou [1]. Hra byla inspirována knihou *Mravenci* od francouzského spisovatele Bernarda Werbera (viz [2]).

Her s „mravenčí“ nebo obecně „hmyzí“ tématikou existuje poměrně mnoho. Najít tak lze hry různých žánrů, od jednoduchých arkádových her, kde je úkolem rozmáčknout mravence prstem na displeji tabletu, přes složitější 3D strategie, kde již mravenci představují jednotnou kolonii, až po realistické simulátory, kde hráč doslova proniká do mravenčího života. Setkáváme se tak s různými přístupy, jak je možné mravenčí život představit a nalézt nové řešení se stává obtížnou, ne-li nemožnou, disciplínou.

Úkolem této práce ovšem není vymyslet originální a novou hru, která by ještě nikde neexistovala. Cílem je vyzkoušet si samotný proces realizace komplexní počítačové hry, a to od prvotního jednoduchého náčrtu přes sofistikovaný návrh až po samotnou implementaci. Z tohoto důvodu také nebyl pro vývoj aplikace použit žádný, již hotový, *herní engine*[1] třetích stran, ale hra byla vyvíjena úplně od základu.

Tato práce si také neklade za cíl vytvořit kompletní hratelnou počítačovou hru. V tomto případě se jedná o komplexní projekt, který vyžaduje velké množství navrhování a ještě větší množství programování. Důraz byl tedy kladen na návrh a implementaci několika samostatných částí. Jedná se ovšem o klíčové části hry, na kterých bude dále možné stavět.

První částí je návrh a implementace datové struktury reprezentující herní svět. Druhým klíčovým bodem je pohyb a ovládání mravence v 3D prostoru. S tímto také souvisí interakce mravence s herním světem. Třetí část se zabývá umělou inteligencí autonomních mravenců. A konečně čtvrtá část, která

[1] Základní jádro hry. Více informací viz [20].

zahrnuje proces přímé komunikace mezi mravencem ovládaným hráčem a autonomními mravenci. Více o tom, co přímá komunikace znamená, je vysvětleno v 1. kapitole: *Popis hry.*

Samotné práci předcházelo vytvoření konceptu hry a tvorba *design dokumentu*[2] (viz [1]), ze kterého se při návrhu hry vycházelo. To znamenalo vymyslet, o čem vůbec hra bude, jakého žánru, co bude jejím cílem, a také navrhnout vhodné herní mechanismy, které by život mravenců do určité míry věrohodně napodobovaly, ale zároveň aby hráče dostatečně bavily a neodrazovaly od dalšího hraní.

Práce je rozdělena do pěti kapitol. V první kapitole je popsána samotná hra. Stručně jsou představeny hlavní mechanismy hry. Zmíněny jsou také některé již existující hry s podobnou tématikou.

Druhá kapitola obsahuje popis implementačního prostředí. Jaký byl zvolen jazyk pro implementaci a proč. Nebo pro jakou platformu je výsledná aplikace určena. Dále tato kapitola obsahuje výčet použitých knihoven.

Třetí kapitola se zabývá návrhem aplikace, tedy vlastního herního enginu. Popsáno je rozdělení aplikace na vhodné subsystémy a potažmo jejich další dělení na samostatné balíčky.

Ve čtvrté kapitole se nachází detailní popis implementace zaměřený na čtyři klíčové části hry zmíněné výše. Tato kapitola již nepopisuje ostatní části enginu (na rozdíl od třetí kapitoly zabývající se návrhem), které sice byly v rámci práce implementovány, ale není na ně brán zřetel. Nesouvisí přímo se čtyřmi body zadání.

Pátá a poslední kapitola seznamuje čtenáře s testovacími scénáři a se samotnými výsledky testování.

[2]Dokument obsahující velmi detailní popis počítačové hry. Je psán a udržován samotnými vývojáři [21].

KAPITOLA 1

Popis hry

V této kapitole je popsána počítačová hra, pro kterou byl vlastní herní engine vyvíjen. První část kapitoly vysvětluje, o čem by hra měla být, jakého je žánru a v jakém prostředí se odehrává. Následuje část, kde jsou uvedeny příklady již existujících her, které se také zabývají „mravenčí“ tématikou. Poslední část kapitoly je věnována herním mechanismům. Zde jsou popsány pouze ty mechanismy, které nějak souvisí s předmětem této práce. Kompletní popis fungování hry je detailně popsán v design dokumentu hry [1].

1.1 Žánr

Život mravenců je 3D počítačová hra, ve které hráč přímo ovládá jednoho mravence z pohledu 3. osoby v otevřeném světě a nepřímo ovládá, prostřednictvím příkazů, ostatní autonomní mravence. Náplní hry je obstarávání potravy jak pro vlastního mravence, tak pro královnu a celou mravenčí kolonii, a dále obrana mraveniště před jinými nebezpečnými živočichy [1].

Hra je tak kombinací více žánrů, zejména akce a strategie. Akční prvek hry je patrný ze způsobu ovládání vlastního mravence (pohled 3. osoby) a z faktu, že hráč bude muset bojovat s jiným hmyzem. Naopak řízení autonomních mravenců a rozšiřování kolonie je znakem strategické hry. Hráči je také umožněno měnit strukturu herního svět a to tak, že bude moci hloubit tunely a podzemní chodby [1].

Cílem hry je vybudovat mravenčí kolonii čítající určité množství mravenců nebo neprohrát po určitý časový úsek. Hráč prohrává, pokud přijde o svou královnu. Královna může zahynout buď hladem, nebo na následky zranění nepřátelským hmyzem [1].

Hra se snaží napodobit slavnější počítačové hry, kde je zpravidla lidská postava ovládána z pohledu 3. osoby a hráč s její pomocí objevuje rozsáhlý otevřený svět. V této hře je ovšem hlavním hrdinou mravenec, což má na první pohled působit úsměvně, ale zároveň v potenciálním hráči vzbudit zvědavost a zájem.

Není třeba zastírat, že možnost měnit herní svět, byla inspirována opět slavnou hrou, a tou je *Minecraft*[3]. Nicméně snaha byla o to, posunout hranice o něco dále a neomezovat se pouze na „krychličky“, ze kterých je svět v Minecraftu tvořen.

Tato kombinace více žánrů, která ze zprvu zdála být velmi originální, zas až tak jedinečná není, jak ilustruje následující sekce věnována rešerši již existujících her ze světa mravenců.

1.2 Existující hry

Počítačových her ze světa hmyzu obecně existuje velké množství. Velkou část z toho zaujímají právě hry s „mravenčí“ tématikou. Tyto hry se dají rozdělit do dvou skupin. První skupinou jsou hry určené pro tablety a mobilní zařízení. Jedná se především o jednoduché arkádové hry ovládané prstem na displeji zařízení. Druhou skupinu tvoří hry cílené na osobní počítače, kde je možné nalézt už složitější hry, které více odpovídají cílům této práce.

Tablety a mobilní zařízení

Do této kategorie spadají především arkádové hry a strategie typu *tower defense (TD)*[4].

Jedná se například o hru **Anthill**, která údajně vychází ze skutečného chování mravenců. Hráč vytváří pachové stopy, kterými směruje celé skupiny mravenců do různých míst, kde útočí na nepřátelské brouky. Cílem hry je ubránit mraveniště. Jedná se o TD. Pro více informací viz [13].

Dalším zástupcem tabletových her je **Ant Raid**. Tato hra sice patří mezi strategie, ale o TD se v pravém slova smyslu nejedná. Úkolem je opět ochránit mraveniště a jeho královnu. Tentokrát ovšem před zmutovaným hmyzem, na který hráč přímo posílá své mravenčí jednotky. Více detailů viz [14].

[3] Více o hře Minecraft viz [12].

[4] Jedná se o podžánr realtimových strategií, kde je cílem zastavit nepřátelské útoky stavěním různých druhů věží, které na pochodující nepřátele střílejí.

Posledním zde uváděným zástupcem her určených pro tablety je **Ant Run**. Jedná se arkádovou hru, kde hráč ovládá jednoho mravence a snaží se dosáhnout co největšího počtu bodů sbíráním ovoce, přičemž se vyhýbá překážkám a ostatním zástupcům hmyzu. Zajímavá je tato hra tím, že je ve 3D. Více informací viz [15].

Osobní počítače

Mezi hrami určenými pro osobní počítače je možné nalézt již větší a zajímavější projekty. Historie těchto her sahá až do devadesátých let minulého století, kdy se již hry s tématem mravenců objevovaly. Z žánrů zaujímá největší zastoupení strategie. Je to celkem pochopitelné, jelikož mravenci svou povahou sociálního hmyzu, kde jedinec nic neznamená - důležitý je prospěch celé kolonie, k tomu vybízejí. Nicméně i tak lze nalézt hry akční zaměřující se na ovládání jednoho mravence.

Jednou z prvních strategií, která se týká mravenců je hra **SimAnt** slavného vývojářského studia Maxis Software. Vydána byla již v roce 1991 pro platformu DOS. Úkolem je řídit mravenčí kolonii, rozšiřovat vlastní území, hledat potravu a chránit královnu. Hráči je umožněno hloubit tunely. Kolonii ohrožují útoky jakýchsi „zlých červených mravenců“. Více informací zde [16].

Podobnou hrou, se stejným cílem, je **Empire of the Ants**, která je ale již ve 3D. Hra byla inspirována, podobně jako tato práce, románem Bernarda Werbera *Mravenci* [2]. Vydána byla v roce 2001. Pro detailnější popis a obrázky ze hry viz [17].

Další zajímavou hrou je opět 3D strategie s prostým názvem **Ant**. Mezi důležité rysy této hry patří možnost volby jednoho ze tří druhů mravenců, a také náhodné generování herních map [18].

Nejnovější a nejlépe vypadající hrou ze světa mravenců je hra **Ant Simulator**, viz [19]. Hra je v současné době stále vyvíjena, což znamená, že vznikala současně s touto prací. Jedná se o simulátor kolonie mravenců, kde hráč ovládá pouze jednoho mravence z pohledu 1. osoby v otevřeném 3D světě. Tento svět je náhodně generován a je možné ho ve hře měnit. Hráč tedy může hloubit tunely a budovat tak podzemní mraveniště. Setkává se tu také s jinými živočichy, kteří hráče ohrožují. Není zcela zřejmé, jaký je cíl hry.

Svět hry je zpracovaný velmi detailně, stejně jako samotní mravenci. Tuto skutečnost ilustrují obrázky 1.1 a 1.2. Autoři zmiňují, že při modelování dbali na vědeckou přesnost veškerého hmyzu ve hře se vyskytujícího [19].

Obrázek 1.1: Detail mravence. Obrázek ze hry Ant Simulator. Zdroj [19].

Obrázek 1.2: Pohled mravence. Obrázek ze hry Ant Simulator. Zdroj [19].

30. listopadu 2015 byla uvedena do prodeje omezená beta verze této hry. Podle dostupných informací a zveřejněných videí (k přečtení a zhlédnutí zde [19]) je zřejmé, že Ant Simulátor je velmi podobný hře vyvíjené v rámci této práce, ačkoli oba projekty vznikaly nezávisle na sobě.

Hry se samozřejmě v některých detailech liší. Např. Ant simulator je ovládán pouze z pohledu 1. osoby, zatímco hra Život mravenců počítá s možností přizpůsobení pohledu 3. osoby dle potřeb hráče. Dále se hra realizovaná v rámci této práce zaměřuje na větší variabilitu mravenčích kast (více o tomto v následující sekci věnované herním mechanismům). Simulátor představil pouze dva druhy mravenců, dělníka a vojáka. Rozdílný je také způsob rozhodovaní o tom, do jaké kasty má nový mravenec spadat (kdy se o tom rozhoduje).

1.3 Herní mechanismy

Tato část se zabývá představením herních mechanismů tak, jak byly navrženy a popsány v design dokumentu. Detailnější popis toho, jak má celá hra fungovat, naleznete přímo ve zmiňovaném dokumentu [1], kde je také uvedeno, jak tyto mechanismy souvisí s fungováním mravenčí kolonie v reálném světě. Inspirace pro hru pocházejí z opravdového světa mravenců. Samozřejmě některé mechanismy musely být upraveny, aby lépe odpovídaly požadavkům kladeným na počítačovou hru.

Princip hry

Hráč přímo ovládá jednoho z mravenců z pohledu 3. osoby. Pomocí tohoto mravence objevuje otevřený herní svět, sbírá potravu, dává příkazy ostatním mravencům (ovládaní umělou inteligencí) včetně královny a bojuje s nepřátelskými živočichy, kteří mravence ohrožují.

Důležitou postavou je mravenčí královna, kterou hráč přímo neovládá. Jejím údělem je klást vajíčka, ze kterých se rodí noví mravenci. Jací mravenci se mají narodit (do jaké spadají kasty), o tom už rozhoduje sám hráč. Z jakých kast hráč vybírá je popsáno dále v této kapitole.

Vlastnosti mravence

Všichni mravenci mají 4 základní vlastnosti, které říkají například to, jak je každý mravenec silný při útoku nebo kolik unese nákladu. Konkrétně se jedná o následující vlastnosti:

- počet životů (životy, které mravenec ztrácí při boji s nepřátelskými brouky nebo při hladovění),
- zásoba potravy (množství živin, které si mravenec nese s sebou a ze kterých čerpá potravu před tím, než začne hladovět, a tedy ztrácet výše zmíněné životy),
- kapacita nákladu (jaké množství nákladu resp. potravy je mravenec schopný najednou nést),
- síla útoku (jak velké poškození provádí během útoku - počet životů, které ubere nepříteli).

Potrava se ve hře vyskytuje ve dvou stádiích. Surová potrava, kterou mravenci získávají z mrtvých brouků nebo z rostlinných plodů, a předzpracovaná potrava, kterou přímo konzumují. Proto existuje jak vlastnost zásoba potravy (pro předzpracovanou potravu), tak i kapacita nákladu (pro potravu v surovém stádiu).

Mravenčí kasty

To, do jaké míry je u mravence určitá vlastnost vyvinuta, určuje kasta, do které daný mravenec spadá. Kasty mají smysl pouze u autonomních mravenců. Mravenec ovládaný hráčem je univerzální, a tak se zde o příslušnosti k jakékoli kastě hovořit příliš nedá. Je sice všestranný, ale v žádné disciplíně nevyniká. Kromě královny, která je pouze jedna a nemůže být nahrazena nově vylíhnutým mravencem, existují následující kasty:

- pečovatelka (větší zásoby potravy, ze kterých krmí larvy, popřípadě samotnou královnu; je ale velmi slabá, a neměla by tedy opouštět mraveniště),
- zásobovač (větší zásoby potravy; slouží tedy jako pohyblivý zdroj živin pro ostatní mravence),
- nosič (větší kapacita nákladu; vhodný pro přenos potravy od zdroje ke královně v mraveništi),
- lovec (větší síla útoku; nezbytný mravenec pro boj s nepřátelským hmyzem při lovu nebo obraně mraveniště).

Autonomní mravenec jakékoli výše zmíněné kasty se může během hry objevit několikrát. O tom, kdy a jaký mravenec se má vylíhnout, rozhoduje hráč ještě před tím, než dojde k nakladení vajíčka královnou.

Přímá komunikace

Hráč prostřednictvím svého mravence předává instrukce ostatním mravencům, včetně královny, pomocí tzv. *přímé komunikace*[5]. Přímou komunikaci iniciuje vždy hráč, a to v bezprostřední blízkosti jiného mravence, se kterým chce komunikaci zahájit.

Po zahájení komunikace může hráč autonomnímu mravenci předat příkaz. Může jít například o povel ke sběru dané potravy, povel k lovu brouka nebo třeba žádost o potravu. V případě královny se může jednat o žádost, aby nakladla další vajíčko požadovaného typu, ze kterého se vylíhne mravenec dané kasty.

[5] V reálném světě se jedná o proces, kdy se dva mravenci potkají a dotykem tykadel (taktilní soustavou neboli tykadlovou řečí) si předávají informace [3]. Mohou si také navzájem předávat potravu ze svých zásob [2].

Herní svět

Hra se odehrává jak na povrchu, tak i v podzemí. Na povrchu se mohou nacházet spadané větvičky stromů, stébla trávy či ze země vyčnívající kameny. Podzemí je tvořeno tunely a prostory různých velikostí, které mají představovat mraveniště. Mravenec se může pohybovat nejen po zemi, ale také po stěnách a stropech těchto tunelů resp. prostorů.

Jednotlivé tunely představující chodby mraveniště si hloubí sám hráč prostřednictvím svého mravence. Herní svět tedy není statický, ale mění se za běhu hry.

Dále je svět tvořen z několika materiálů, které jsou pro hráče při kopání různě prostupné. Např. skálou prohrabat tunel možné není, zatímco dřevem ano, ale bude to trvat podstatně déle než při hrabání tunelu pískem. Díky tomu je herní svět určitým způsobem vymezován a není tak umožněno hráči prokopat celý prostor.

Na všechny výše zmíněné skutečnosti týkající se jak komunikace mravenců, tak specifikace herního světa, je třeba dbát při návrhu herního enginu. Následující kapitola se zabývá volbou implementačního prostředí pro realizaci tohoto enginu.

KAPITOLA 2

Implementační prostředí

Tato část práce popisuje implementační prostředí, ve kterém byla aplikace realizována. Kapitola je rozdělena do dvou sekcí. V první je zmínka o programovacím jazyku implementace, grafickém API a také platformě, pro kterou je výsledná hra určena. Druhá část obsahuje výčet všech použitých knihoven třetích stran.

2.1 Jazyk implementace

Jelikož hra vyžaduje složitější ovládání při pohybu 3D světem pomocí klávesnice a myši, není určena pro tablety či mobilní zařízení, ale pro osobní počítače.

Snaha byla také o to, aby hru bylo možné hrát na všech třech hlavních operačních systémech OS X, Linux a Windows. Proto jako grafické API bylo zvoleno OpenGL a jazykem implementace se stalo C++. Dalším důvodem pro použití této grafické knihovny je fakt, že autor veškeré své dosavadní školní práce psal v tomto prostředí, a tedy OpenGL a C++ ovládá.

Díky tomu se také mohla aplikace vyvíjet primárně na operačním systému Linux, který autorovi pro programování více vyhovuje. Programovací práce probíhaly ve vývojovém prostředí CLion od společnosti JetBrains. Model mravence vznikl v modelovacím nástroji 3ds Max od společnosti Autodesk.

2.2 Použité knihovny

I když součástí této práce byl vývoj vlastního herního enginu, a to úplně od píky, nebylo možné ani žádoucí vyhýbat se použití několika knihoven třetích stran.

Základem pro ovládání grafického hardwaru se stala již zmíněná knihovna OpenGL za použití jazyka GLSL pro programování shaderů[6]. Jelikož hlavním tématem této práce není grafická stránka hry, nebylo zapotřebí psát v GLSL složité programy pro grafickou kartu. Plně dostačující byly základní shadery pro vykreslení polygonové sítě. Pro správu dostupných rozšíření OpenGL byla použita knihovna GLEW.

Pro správu oken operačních systému, do kterých je renderován výsledný obraz, byla vybrána knihovna SFML, která je multiplatformní a nabízí jednoduché rozhraní pro různé součásti počítače k usnadnění vývoje her a jiných multimediálních aplikací [9].

Pro výpočet transformací a obecně pro práci s maticemi a vektory byla použita knihovna GLM. Jedná se knihovnu specializovanou na matematiku pro grafický software a je založena na specifikaci GLSL [10].

Engine vyžaduje také práci s JSON soubory. Pro tyto účely využívá knihovnu RapidJSON. Knihovna poskytuje rozhraní jak pro parsování, tak pro generování JSON souborů [11]. Více o této problematice se nachází ve 4. kapitole: *Popis implementace.*

Poslední knihovnou využívanou herním enginem je knihovna FreeImage, díky které vznikla dokumentační videa.

[6]Více o shaderech a programování na straně grafické karty pomocí GLSL viz [7].

KAPITOLA 3

Návrh aplikace

Tato kapitola se zabývá návrhem samotné hry Život mravenců. Nejdříve jsou popsány cíle aplikace představující herní engine a požadavky na tuto aplikaci kladené. Poté následuje sekce věnována návrhu subsystémů, kde je nastíněno dělení aplikace na menší samostatné celky, které jsou pak detailněji představeny.

Následující kapitola pak vyzdvihuje a podrobněji popisuje implementaci vybraných částí herního enginu, které jsou pro tuto práci stěžejní. Jedná se přesně o ty části, které byly představeny již v úvodu práce.

3.1 Herní engine

Herní engine, ačkoli je vyvíjen pro realizaci konkrétní hry, by měl být navržen tak, aby ho bylo možné využít i v dalších projektech. Aby nesloužil jen pro tuto jednu hru. Při návrhu se tedy dbalo na to, aby úlohy spojené např. s vykreslováním grafiky byly řešeny co možná nejobecněji.

Velká pozornost byla věnována tomu, aby bylo možné engine dále rozšiřovat. Aby při potřebě přidání některé funkčnosti nebylo nutné přepisovat velké množství kódu. Tuto skutečnost dobře ilustruje například způsob nahrávání 3D modelů ze souborů. Model mravence je uložen v obj[7] souboru. Stačilo by tedy, aby aplikace obsahovala třídu (nebo třídy) pro nahrávání modelů do scény, která by implementovala parsování obj souboru.

[7]Wavefront OBJ (object) file. Standard pro reprezentaci polygonálních dat [23].

Pokud bychom ovšem chtěli někdy v budoucnu nahrávat modely i z jiných formátů, museli bychom tuto třídu přepsat a zřejmě sáhnout i do jiných částí aplikace, kde by bylo definováno, jaký že to formát používáme.

Proto je v tomto enginu nejprve navrženo obecné rozhraní pro parsování souborů obsahujících data 3D modelů. O parsování obj souborů se stará až konkrétní implementace tohoto rozhraní. Až bude v pozdějších projektech vyžadováno nahrávání souborů v jiném formátu, stačí přidat další třídu implementující toto rozhraní, která bude zpracovávat soubory v novém formátu. Problémem nebude ani situace, kdy bude vyžadována knihovna třetí strany. Konkrétní implementace rozhraní může tuto knihovnu využívat[8].

Podobně je řešeno i nahrávání dat týkajících se herního světa. Opět je nejdříve navrženo rozhraní, které implementuje třída starající se o parsování konkrétního souboru. Pokud by došlo k rozhodnutí změnit formát pro ukládání herního světa, stačí jednoduše přidat další implementaci tohoto rozhraní. Více o tom, jak je herní svět implementován se nachází ve 4. kapitole.

Dalším požadavkem kladeným na herní engine bylo oddělení grafiky od logiky. Tedy aby programování umělé inteligence nebylo nijak vázáno na renderování objektů a její implementace stála v programu odděleně od částí týkajících se grafické reprezentace těchto objektů.

Výhodou takového přístupu je, že výpočet umělé inteligence může probíhat zcela nezávisle na jejich vykreslování. Objekty není potřeba vždy renderovat. Ani to není žádoucí z pohledu výkonu aplikace. Ideální je zobrazovat pouze ty objekty, které může v dané chvíli hráč vidět. V takovém případě je ovšem nutné stále vědět, kde se objekty nacházejí, aby je bylo možné zobrazit, až se znovu dostanou na scénu. Poloha objektu závisí na rozhodnutích, které daný objekt, prostřednictvím umělé inteligence, provede.

Dalším kladem je možnost nahrazení celého grafického enginu za jiný, aniž by došlo k „rozbití“ logického subsystému. Nebo naopak výměna celé logiky za jiný matematický model.

Následující sekce se zabývají již konkrétním návrhem herního enginu. Jeho rozdělením na subsystémy, které jsou dále více rozepsány. Pro lepší ilustraci jsou uvedeny některé UML diagramy vytvořené během návrhu aplikace před samotnou implementací.

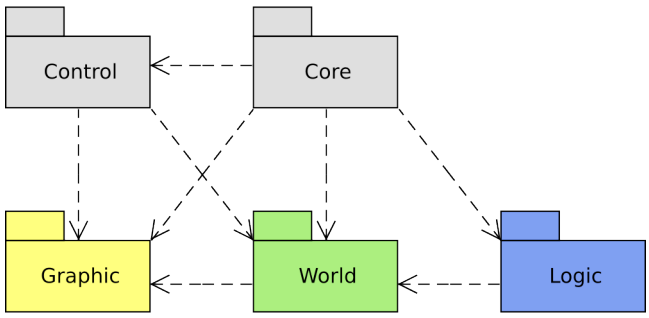
[8] Jelikož tento projekt vyžaduje načítání modelů pouze z obj souborů, které nejsou nijak složité a jejich struktura je autorovi známá, bylo rozhodnuto žádnou knihovnu pro parsování 3D modelů nevyužívat.

3.2 Návrh subsystémů

Herní engine obsahuje tři základní subsystému neboli moduly. Jedná se o modul **Graphic**, který má na starosti správu geometrických objektů a jejich renderování. Dále modul **World**, kde je umístěna mj. implementace grafu scény nebo datová struktura reprezentující herní svět. A konečně subsystém **Logic**, který se stará o ovládání autonomních mravenců.

Mimo tyto základní moduly obsahuje engine ještě menší modul pro realizaci ovládání pomocí klávesnice a myši ze strany hráče nazvaný **Control**. Posledním modulem je **Core** neboli jádro, které se stará o inicializaci všech výše zmíněných modulů. Zde je také umístěna herní smyčka, kde se stále dokola provádí aktualizace scény a rendering světa. Závislosti mezi jednotlivými moduly zobrazuje diagram balíčků na obrázku 3.1.

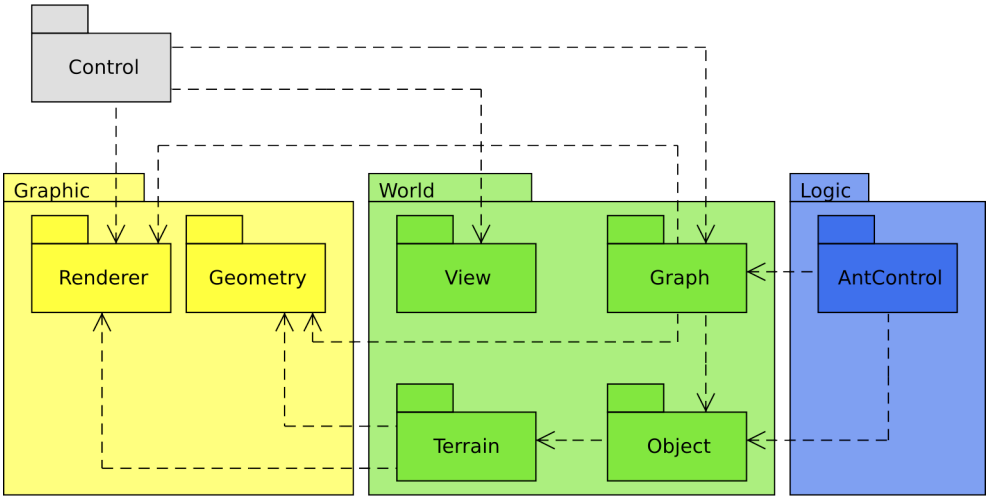
Kromě těchto pěti subsystému obsahuje aplikace ještě správu konfiguračních souborů **Config** a další převážně statické třídy obsahující podpůrné funkce společné pro ostatní moduly. Tyto třídy jsou umístěny v samostatném balíčku pod souhrnným nazvaném **Common**. Nejedná se přímo o moduly, ale pouze o třídy, které jsou využívány napříč celým enginem. Proto ani nejsou na obrázku 3.1 zobrazeny.



Obrázek 3.1: Diagram balíčků: rozdělení na subsystémy.

Engine také disponuje vlastní správou výjimek. Tato část ovšem není tolik důležitá a je již nad rámec této práce.

Moduly Graphic, World a Logic jsou dále děleny na menší balíčky. Toto rozdělení ilustruje podrobnější diagram balíčků na obrázku 3.2. Diagram pro větší přehlednost již neobsahuje modul Core, který závisí na všech ostatních modulech.



Obrázek 3.2: Diagram balíčků: rozdělení subsystémů na balíčky.

V několika následujících sekcích této kapitoly jsou podrobněji jednotlivé moduly popsány. Hlavně to, jakou zastávají funkci. Uvedeny jsou také návrhy tříd realizující tuto funkčnost.

3.3 Graphic

Modul zabývající se grafickou stránkou hry Graphic je rozdělen na dva balíčky. Jedná se o balíček nazvaný **Renderer**, kde, jak už název napovídá, jsou umístěny všechny třídy zabývající se vykreslováním grafiky. Druhý balíček se zabývá správou geometrických objektů, jejich načítáním ze souborů a přenos do paměti grafické karty. Tento balíček byl nazván **Geometry**.

Renderer

Balíček Renderer má na starosti renderovací proces herního enginu. Jádru enginu nabízí rozhraní *Renderer*, podle kterého je také celý balíček nazván. Toto rozhraní obsahuje metody pro správu okna, do kterého se vykresluje. Modulu Control pro ovládání mravence hráčem poskytuje rozhraní *WindowController* obsahující metody pro získávání událostí týkajících se renderovacího okna. Jedná se jak o události zavření či zvětšení okna, tak i třeba o stisknutí klávesy nebo tlačítka myši.

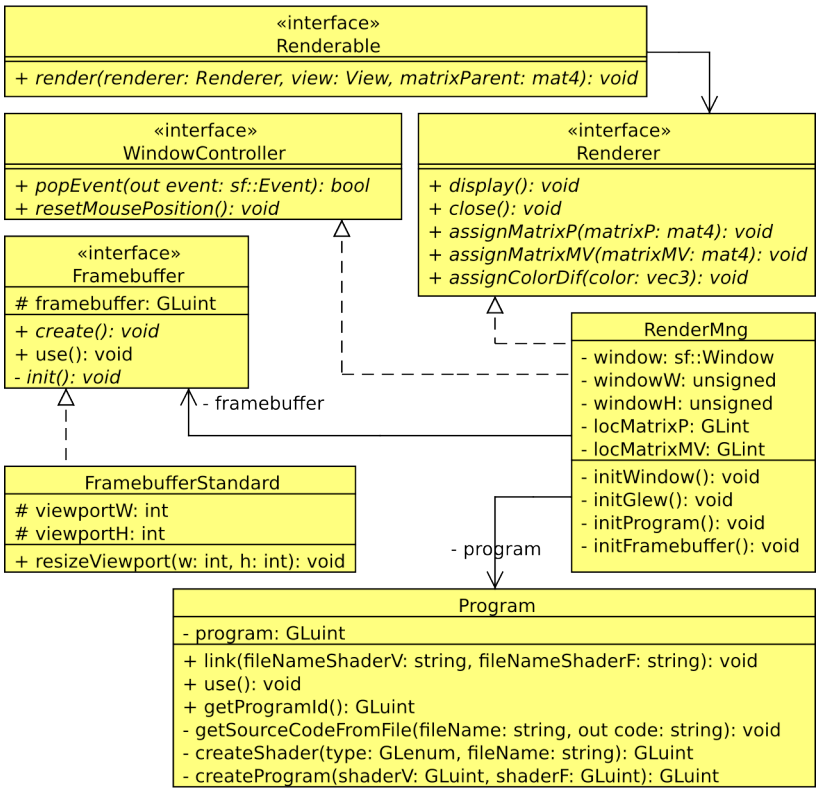
Obě tato rozhraní implementuje jediná třída *RenderMng*. Správu renderovacího okna implementuje za pomocí knihovny SFML. Obaluje tak objekt této knihovny, který realizuje správné fungování okna napříč různými platformami.

SFML se také stará o udržování fronty událostí přicházejících ze strany hráče či operačního systému.

Mimo to se třída stará o inicializaci framebufferu a také správu shaderů. Přestože engine zatím využívá pouze jeden (a to standardní) framebuffer, poskytuje balíček Renderer rozhraní pro práci s ním, ze kterého teprve dědí konkrétní implementace framebufferu. Pokud by bylo potřeba realizovat jiný než standardní framebuffer, stačí přidat třídu implementující toto rozhraní.

Kompilaci shaderů a následné vytvoření a slinkování programu běžícího na grafické kartě má na starosti třída *Program*, která tak obaluje objekt *program* knihovny OpenGL.

Balíček dále nabízí ostatním modulům rozhraní *Renderable*. Toto rozhraní obsahuje jedinou metodu a tou je metoda pro renderování. Objekt třídy implementující toto rozhraní se tedy stává objektem určeným pro vykreslení. Může jím být např. mravenec nebo část herního světa.

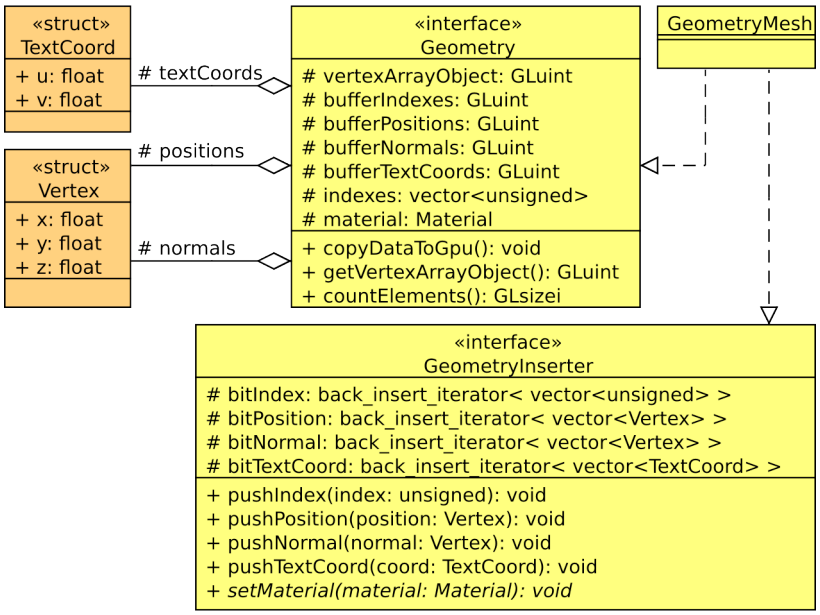


Obrázek 3.3: Diagram tříd: balíček Renderer modulu Graphic.

Výše popsaný návrh balíčku doplňuje diagram tříd na obrázku 3.3. Pro jednoduchost jsou vynechány některé detaily, které pro ilustraci a pochopení struktury balíčku nejsou důležité.

Geometry

Základem tohoto balíčku je (opět obecné) rozhraní reprezentující geometrii 3D objektu *Geometry*. Třída implementující toto rozhraní obaluje OpenGL objekt *vertex array object* spolu se všemi potřebnými buffer objekty (*vertex buffer object*) pro uchování dat vrcholů dané geometrie. Pro potřeby parsování souborů s uloženými modely obsahuje balíček také rozhraní *GeometryInserter*, které umožňuje vkládat do paměti nová data vrcholů.



Obrázek 3.4: Diagram tříd: balíček Geometry modulu Graphic - 1. část.

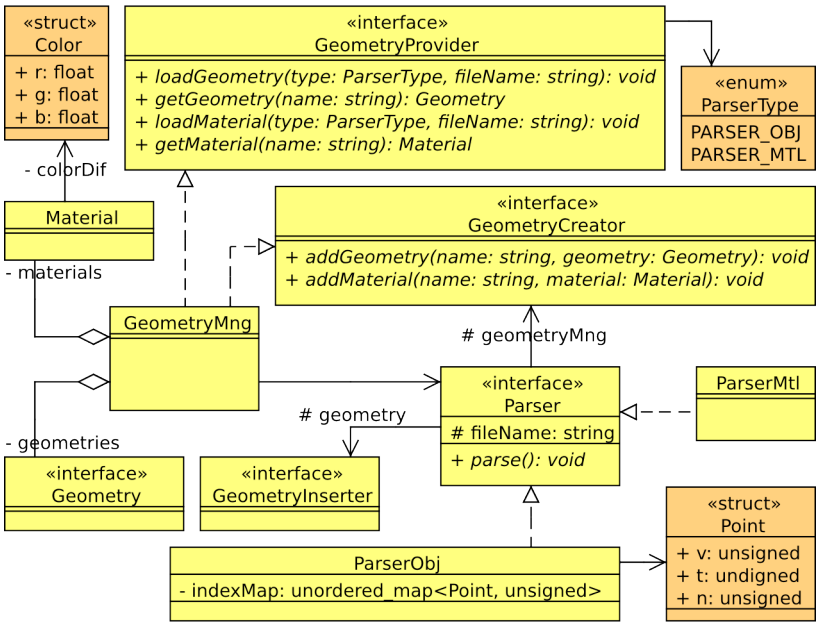
Konkrétní geometrie může implementovat obě zmíněná rozhraní, jestliže vyžaduje načtení dat ze souboru. Pokud se ovšem nejedná o model uložený v souboru, stáčí, když třída reprezentující tuto geometrii implementuje pouze rozhraní Geometry. Potom je ale potřeba naplnit příslušné kontejnery daty vrcholů explicitně v konstruktoru třídy.

V každém případě je úkol těchto tříd stejný. Třídy zajišťují kopírování dat vrcholů do paměti grafické karty a také poskytují informace například o po-

čtech elementů, které je třeba renderovat, nebo o materiálech, které jsou s modelem spojeny.

Diagram tříd ilustrující tuto část balíčku je na obrázku 3.4. Opět zde nejsou pro jednoduchost uvedeny veškeré implementační detaily včetně metod, které nejsou pro pochopení funkčnosti stěžejní.

Konkrétní objekty reprezentující geometrie 3D modelů jsou uchovávány ve správci geometrie (třída *GeometryMng*). Kromě modelů obsahuje tato třída také seznam dostupných materiálů. Třída implementující materiál uchovává data týkající se podoby povrchu modelu. Zatím je počítáno pouze s difúzní barvou povrchu.



Obrázek 3.5: Diagram tříd: balíček Geometry modulu Graphic - 2. část.

Jelikož je se správcem geometrie zacházeno dvěma odlišnými způsoby, třída implementuje opět dvě rozhraní. První umožňuje řídit načítání požadovaných modelů a příslušných materiálů a poté umožňuje získat tyto načtené modely (resp. materiály) dle jejich názvů v podobě geometrie. Jedná se o rozhraní *GeometryProvider*.

Druhé rozhraní je poskytováno parserům pouze v rámci tohoto balíčku a dává jim tak možnost přidávat modely resp. materiály do správce geomet-

rie. Rozhraní *GeometryCreator*. Tuto část balíčku ilustruje diagram tříd na obrázku 3.5.

Samotný parser je realizován jako obecné rozhraní pro načítání dat geometrií, resp. materiálů ze souborů. Parsování konkrétních souborů (jako je například obj) se provede implementací tohoto rozhraní. Díky tomuto postupu je snadné rozšířit podporující formáty modelů pouhým přidáním další implementace tohoto rozhraní.

Pro vkládání dat vrcholů do objektu reprezentujícího geometrii modelu je využíváno rozhraní GeometryInserter popsané dříve.

3.4 World

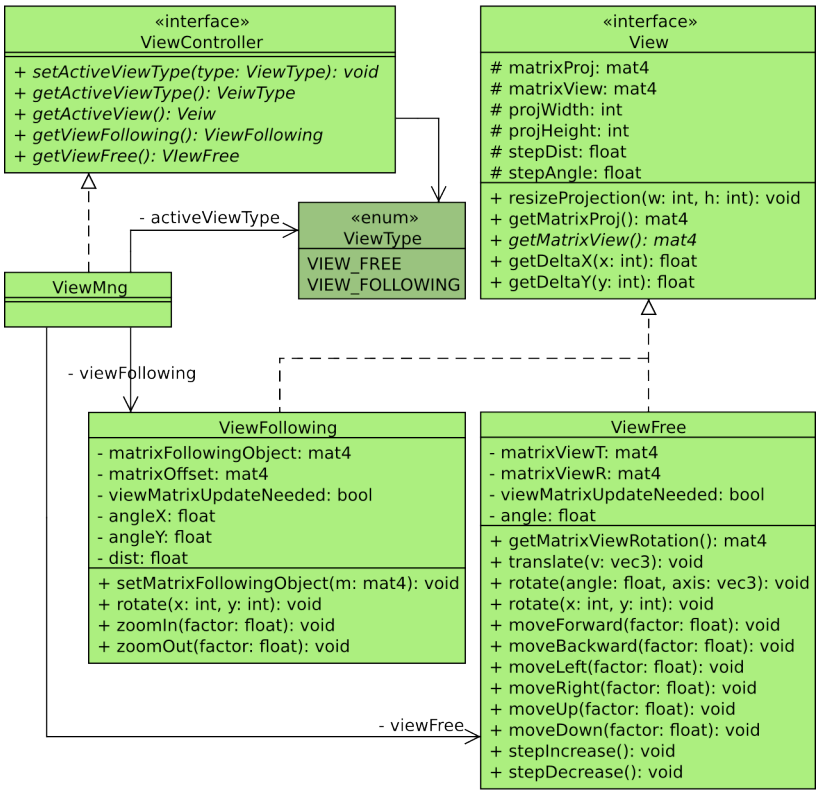
Účelem modulu World je spravovat všechny objekty týkající se herního světa. A to nejen datové struktury reprezentující terén světa, ale i 3D objekty ve světě se nacházející. Tyto objekty jsou uspořádány do grafu scény. Dalším prvkem spadajícím do tohoto modulu je pohled (view), neboli objekt zapouzdřující pohledovou matici.

Modul je rozdělen do čtyř samostatných balíčků. Balíček **Terrain** má na starosti implementaci již zmíněné datové struktury pro reprezentaci herního světa. Dalším je balíček **Object**, který udržuje informace týkající se 3D objektů scény, jako je jeho poloha ve světě. Balíček **View** má na starosti zmíněnou pohledovou matici. A konečně posledním balíčkem je **Graph**, který implementuje samotný graf scény.

Následuje popis návrhu balíčku View a úvod do balíčku Graph, jehož detailnější popis spolu s balíčkem Object se nachází v následující 4. kapitole týkající se samotné implementace. Stejně tak balíčku Terrain je věnována samostatná sekce následující kapitoly.

View

Diagram tříd ilustrující návrh tohoto balíčku je zobrazen na obrázku 3.6. Pohled neobsahuje pouze pohledovou matici, ale zapouzdřuje také matici projekční. Účelem pohledu je kromě obalování těchto matic také poskytování metod pro změnu rozměrů projekce (užitečné např. při změně velikosti okna) nebo pro ovládání kamery, jako je rotace či pohyb, popřípadě i změna rychlosti pohybu kamery.



Obrázek 3.6: Diagram tříd: balíček View modulu World.

Pohledů může existovat více druhů, proto je opět definováno rozhraní obsahující společnou funkčnost pro všechny tyto pohledy. Třídy implementující toto rozhraní tak realizují konkrétní pohledy.

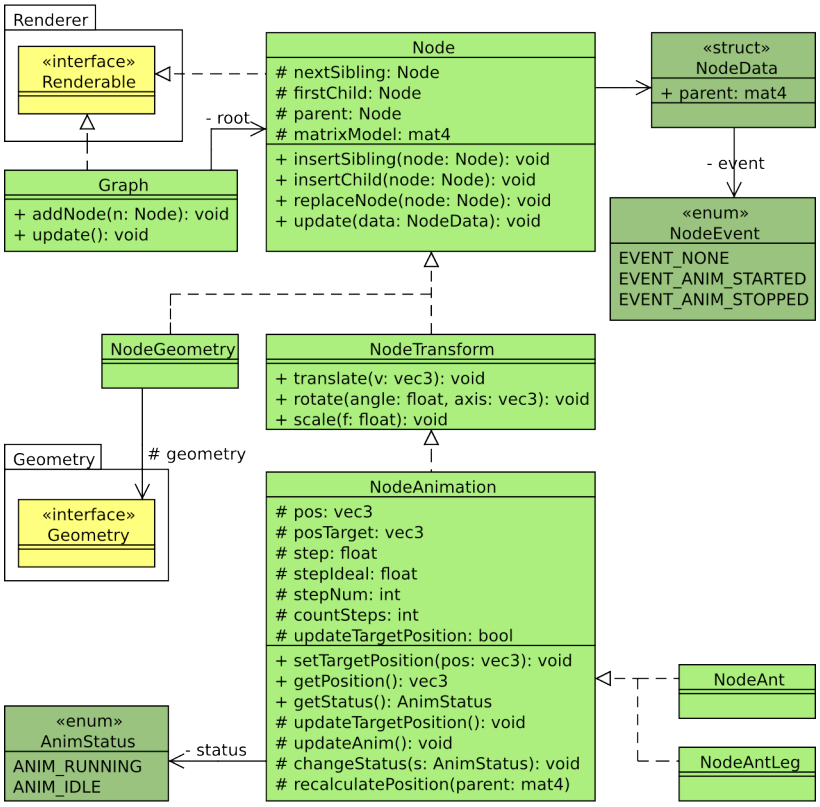
V tuto chvíli je počítáno se dvěma různými pohledy. Jednak volná kamera (třída *ViewFree*), která umožňuje volně se pohybovat v prostoru bez ohledu na objekty či terén a slouží tedy hlavně pro testování, a jednak kamera vázaná na určitý objekt (třída *ViewFollowing*), který následuje (např. mravence). Tato druhá kamera představuje pohled ze 3. osoby a je tedy využívána pro samotné hraní hry.

Balíček dále obsahuje správce těchto pohledů (rozhraní *ViewController* a konkrétní implementace *ViewMng*), který udržuje informaci o aktuálně vybraném pohledu a poskytuje tento pohled modulu Control. Ten využívá pohled pro jeho ovládání, případně přepínání mezi pohledy. Pohled může být samozřejmě aktivní pouze jeden.

Graph

V tomto balíčku je implementován graf scény. Jedná se o stromovou datovou strukturu usnadňující proces renderingu, neboť dovoluje skládat transformace navázaných objektů během průchodu stromem. Obrázek 3.7 obsahuje zjednodušený diagram tříd popisující jeho implementaci. Diagram obsahuje kromě tříd spadajících do tohoto balíčku (zelená barva) také zjednodušené třídy modulu Graphic (žlutá barva).

Základním stavebním kamenem grafu je třída *Node* představující uzel stromu. Třída obsahuje ukazatele na prvního potomka a následujícího sourozence (popřípadě i na rodiče) a slouží jako základ pro dědění ostatními již specifickými uzly. Dále definuje matici modelu, která reprezentuje transformaci uzlu relativně k jeho rodiči.



Obrázek 3.7: Diagram tříd: balíček Graph modulu World - 1. část. Žlutou barvou jsou vyznačeny třídy spadající do modulu Graphic, ostatní zelené třídy patří modulu World.

Třída implementuje rozhraní Renderable modulu Graphic, což znamená, že objekt této třídy je určen k vykreslování. Během renderingu obdrží transformační matici svého rodiče, kterou použije spolu s vlastní maticí modelu pro výpočet celkové transformace. Po vykreslení (pokud má co vykreslovat) předá výslednou matici potomkovi, který proces opakuje.

Podobným způsobem probíhá i aktualizace stavu uzlu. Tentokrát ovšem nedochází k renderingu, ale k úpravě samotné matice modelu. Aby si mohly různé uzly předávat různá data, předávají si ukazatel na obecnou strukturu *NodeData*, ze které opět specifické implementace datových struktur dědí, podobně jako jednotlivé uzly dědí od třídy Node.

Základní datová struktura obsahuje pouze matici rodiče a identifikátor události, která nastala v předkovi. Tímto způsobem je umožněno reagovat na události vzniklé dříve během procesu aktualizace. Událostí může být například zahájení či ukončení animace daného uzlu.

Specifickým uzlem dědícím z třídy Node je například uzel *NodeTransform*, který poskytuje veřejné metody pro rotaci, translaci a změnu měřítka. Dalším uzlem je třída *NodeGeometry* obsahují instanci geometrie a tedy slouží pro renderování 3D objektu ve scéně. Obecné rozhraní pro animaci představuje třída *NodeAnimation*. Třída nabízí metody pro řízení plynulé animace přesunu z jednoho místa na druhé.

Z animačního uzlu také dědí dvě třídy reprezentující samotného mravence. Jedná se o uzly *NodeAnt* a *NodeAntLeg*. Tyto třídy jsou podrobněji popsány až v následující kapitole týkající se implementaci, proto zde nejsou dále popisovány. Proto také diagram tříd na obrázku 3.7 neobsahuje podrobný popis těchto tříd. Zobrazuje pouze jejich zjednodušenou verzi, aby bylo zřejmé, jak do celkového návrhu zapadají.

3.5 Logic

Modul logic se zabývá umělou inteligencí ovládající pohyb a chování autonomních mravenců. Modul obsahuje pouze jediný balíček *AntControl* zajišťující tuto funkčnost.

Jelikož se jedná o jedno ze čtyř stěžejních témat této práce, jsou detaily o jeho implementaci uvedeny ve 4. kapitole: *Popis implementace*. Tato kapitola týkající se návrhu aplikace se tomuto tématu více nevěnuje.

3.6 Control a Core

Modul Control představuje jednu jedinou třídu s názvem *Control*, která zpracovává události ze vstupních zařízení (klávesnice, myš) a také renderovacího okna (zavření, změna velikosti). K tomu využívá rozhraní WindowController poskytované modulem Graphic. Pro ovládání aktuálního pohledu využívá rozhraní ViewController modulu World. A konečně k ovládání samotného mravence používá rozhraní Controlable také z modulu World. Více o tomto rozhraní je uvedeno v následující 4. kapitole, v části zabývající se pohybem mravence.

Jádro aplikace, odkud je řízena inicializace všech potřebných modulů, představuje opět jediná třída nazvaná jednoduše *Core*. V této třídě je implementována herní smyčka. Jedná se o cyklus, který je prováděn stále dokola, dokud není aplikace ukončena. V cyklu se provádí rendering všech potřebných 3D objektů a terénu, aktualizace světa a zpracování vstupů z klávesnice a myši.

Způsob, jak je prováděna aktualizace scény, byl převzat z článku [4], kde je představeno několik možných způsobů implementace herní smyčky. Implementována byla čtvrtá varianta v pořadí. Tedy ta, kdy dochází k akumulaci času renderování jednotlivých snímků (framů), ale stále bez interpolace stavu scény před vykreslením.

Aktualizace scény je prováděna v pevně daném časovém intervalu, což znamená, že nenastává nutně v každém cyklu, kdy je scéna renderována, ale jen tehdy, když dojde k překročení časového limitu. Pokud naopak renderování trvá déle než je stanovený limit, provede se aktualizace vícekrát v jednom cyklu. Ubíhající čas mezi jednotlivými snímky je postupně akumulován, čili rendering a aktualizace probíhají zcela asynchronně. Díky tomu nezáleží na snímkové frekvenci (framerate), ale aktualizace světa bude probíhat vždy v požadovaném časovém intervalu.

Ve zmíněném článku je tento proces akumulace času zajímavě vysvětlen pomocí metafory. Ta říká, že renderování scény produkuje čas, který simulace fyziky (zde aktualizace scény) spotřebovává po dávkách předem zvolené velikosti.

Jelikož jsou stavy objektů reprezentovány transformačními maticemi, nikoli kvaterniony, je scéna vykreslena vždy po provedení celého kroku a nedochází k interpolaci stavu (určení přesného mezistavu mezi aktuálním a následujícím stavem).

3.7 Config a Common

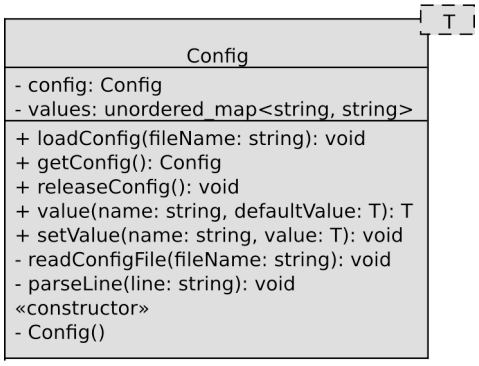
Modul Config má na starosti správu konfiguračních souborů. Je tvořen opět jedinou třídou s názvem odvozeným od názvu modulu. Konfiguraci je možné nahrát z více souborů. Všechny záznamy jsou ukládány do stejného kontejneru, proto musí mít unikátní označení napříč všemi konfiguračními soubory, jinak dojde k jejich přepsání.

Třída umožňuje také jednotlivé záznamy nastavovat za běhu programu (myšleno tak, že nahrané záznamy ze souboru mohou být dále za běhu programu přepsány). To se hodí například v situaci, kdy v souboru je nastaveno větší rozlišení okna, než je velikost monitoru. Knihovna SFML v takovém případě velikost okna přizpůsobí aktuálním možnostem. Skutečné rozlišení je pak v instanci konfigurační třídy přenastaveno.

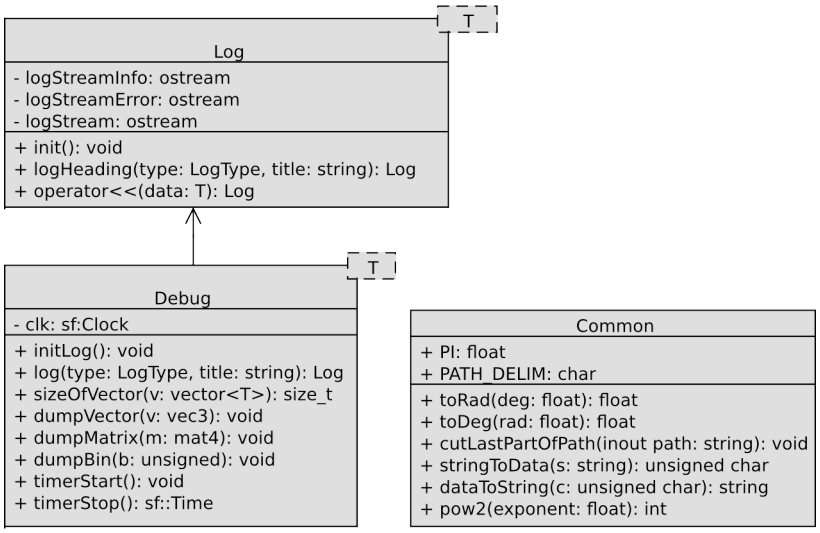
Také z tohoto důvodu třída implementuje návrhový vzor singleton, který zajišťuje, že v celé aplikaci existuje pouze jediná instance této třídy. Díky tomu sdílejí všechny subsystémy a jejich části stejnou konfiguraci a nemůže se stát, že jeden modul přenastaví velikost okna a druhý bude uvažovat původní hodnoty ze souboru s uloženou konfigurací.

Hodnoty jednotlivých záznamů jsou v paměti ukládány jako textové řetězce. Teprve při jejich poptávání ze strany ostatních modulů dochází k jejich převedení na požadovaný datový typ.

Tentokrát kompletní návrh třídy Config zajišťující správu konfiguračních souborů je zobrazen na obrázku 3.8.



Obrázek 3.8: Diagram tříd: modul Config. Privátní konstruktor, statická proměnná *config* udržující instanci třídy a statická metoda *getConfig()* vracející tuto instanci zajišťují, že se třída chová jako singleton.



Obrázek 3.9: Diagram tříd: skupina tříd pod souhrnným názvem Common.

Obrázek 3.9 ilustruje návrh skupiny tříd pod souhrnným názvem Common. Jedná se převážně o statické třídy (neexistují jejich instance, pouze jsou využívány jejich statické metody) a mohou být využívány všemi subsystémy enginu.

Jednou takovou třídou je třída *Common*, která obsahuje užitečné metody například pro převod stupňů na radiány nebo práci s řetězci. Nabízí také užitečné konstanty, jako je například číslo π.

Třída *Debug* slouží, jak už napovídá její název, k testování a debugování aplikace. Definuje několik statických metod pro výpis různých objektů nebo pro měření času na straně CPU i GPU. Kromě toho poskytuje mechanismus pro logování. K tomu využívá třídu *Log*, která přetěžuje operátor << pro výstup zpráv do předem zvoleného streamu. Operátor vrací objekt třídy Log, aby bylo možné zřetězovat jednotlivé části zprávy a proměnné. Deklarace přetížení operátoru je k vidění na obrázku 3.9.

KAPITOLA 4

Popis implementace

Tato část práce se zabývá detailnějším popisem implementace čtyř hlavních témat spojených s touto prací. Jedná se o datovou strukturu reprezentující terén herního světa, pohyb mravence a jeho interakce s terénem, ovládání autonomních mravenců pomocí umělé inteligence a realizace procesu přímé komunikace. Kapitola je rozdělena podle těchto témat do čtyř sekcí.

4.1 Herní svět

První část této podkapitoly se zabývá výběrem vhodné datové struktury. Shrnuty jsou požadavky na herní svět, uvedeny jsou také kritéria výběru. Další části se již zabývají samotnou datovou strukturou a její implementací.

Výběr vhodné datové struktury

V 1. kapitole zabývající se představením herních principů jsou zmíněny požadavky na herní svět, které musí datová struktura pro jeho reprezentaci naplňovat. Pro shrnutí jsou zde tyto požadavky v bodech uvedeny:

- existence tunelů a podzemních chodeb,
- několik odlišných druhů materiálu terénu,
- změna terénu hráčem za běhu hry.

Možností, jak herní svět implementovat existuje více. Každá datová struktura je vhodná pro něco jiného a některé nelze použít vůbec, jelikož neumožňují naplnit všechny zmíněné požadavky. Další kritéria pro výběr vhodné datové struktury mohou být následující:

- časová náročnost konstrukce celé datové struktury (proces předzpracování), popřípadě jen její části (při nahrávání nových oblastí),
- časová náročnost aktualizace struktury (po změně provedené hráčem za běhu programu)
- časová náročnost výpočtu kolizí s ostatními objekty světa (výpočet polohy objektu na povrchu terénu),
- paměťová náročnost.

Doba trvání konstrukce datové struktury není stěžejní, jelikož je možné ji provést v procesu předzpracování. Nicméně nepředpokládá se, že celá struktura bude nahrána v paměti, pouze její část. Proto je nutné uvažovat i situaci, kdy bude potřeba načíst do paměti nové oblasti. To je ovšem možné provádět s dostatečným předstihem, aby hráč nezaregistroval případnou delší dobu trvání tohoto procesu.

Důležitějším faktorem se může zdát doba trvání změny struktury, která je prováděna přímo hráčem. Ovšem ani toto kritérium není stěžejní, protože změna nemusí být okamžitá. Naopak schválně trvá déle, aby simulovala proces hloubení tunelu mravencem, který i v reálném světě nějakou dobu trvá.

Čas potřebný pro výpočet kolizí už ale důležitým hlediskem je. Poloha všech renderovaných objektů na povrchu terénu se musí počítat v reálném čase. Důležitým faktorem je i paměťová náročnost datové struktury, jelikož má být herní svět dostatečně detailní a je nutné udržovat viditelné oblasti v paměti.

Možné datové struktury vhodné pro reprezentaci terénu jsou následující (zdaleka ne jediné):

- polygonální síť (např. okřídlená hrana),
- výšková mapa (height map),
- buněčný model,
- oktalový strom (octree).

Polygonální sítě představují povrchovou reprezentaci terénu a jsou tedy snadno renderovatelné a nejsou paměťově náročné (v porovnání s objemovou reprezentací). Výpočet kolizí je ovšem náročnější operací. Proto je nutné uspořádat povrchová data pomocí vhodné datové struktury, jakou je třeba *okřídlená hrana*. Dále není jednoduché takto definovaný povrch měnit.

Další povrchovou reprezentací je *výšková mapa*, která naopak umožňuje snadnou detekci kolizí. Problém je ovšem s tvorbou tunelů a podzemních prostor, jelikož výšková mapa uvažuje pouze jednu vrstvu povrchu.

Objemová reprezentace pomocí *buněčného modelu* umožňuje snadno vytvářet tyto tunely a hlavně je i snadno modifikovatelná. Vyhodnocování kolizí je také velmi jednoduché. Nevýhodou tohoto modelu je jeho velká paměťová náročnost.

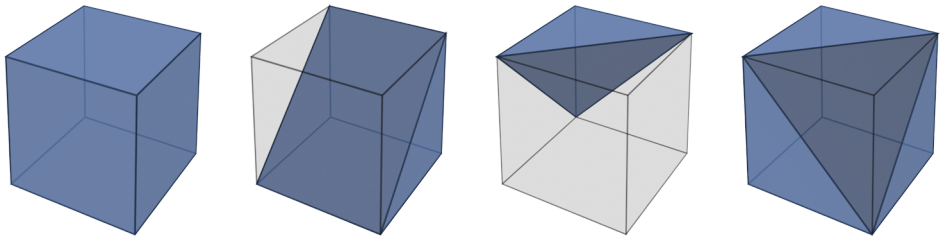
Vhodným kompromisem mezi paměťovými nároky a složitostí operace detekce kolizí představuje další objemová reprezentace: *oktalový strom*, neboli *octree*. Při této reprezentaci dochází k sjednocování malých objemů stejného materiálu do větších celků, čímž nastává úspora paměti. Vyhodnocování kolizí je složitější než v případě buněčného modelu, protože je potřeba procházet stromovou strukturu.

Oktalový strom se jeví jako nejvhodnější kandidát, proto byl také vybrán pro reprezentaci herního světa této hry.

Buňky terénu

Herní svět je tvořen buňkami různých velikostí, které jsou uspořádány do stromové struktury octree. Za účelem zvýšení vizuálního zážitku ze hry bylo rozhodnuto nepoužívat pouze buňky tvaru krychle, ale uvažovány jsou i jiné složitější útvary. Díky tomu nebude svět na první pohled působit hranatě. Možné tvary buněk jsou zobrazeny na obrázku 4.1.

Základní buňkou zůstává stále krychle, pouze rozhraní dvou materiálů (mezi materiály počítán i vzduch) může být tvořeno jinými tvary, které ovšem stále zapadají do buněčné mřížky.

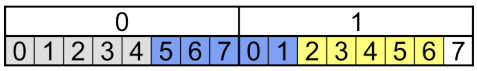


Obrázek 4.1: Ukázka možných tvarů buňky terénu.

Obrázek 4.1 obsahuje 4 základní tvary buněk. Jejich otočením vznikají tvary další. Definováno tak bylo celkem 29 různých tvarů buněk. Jejich kompletní seznam je uveden v příloze B. Prvním tvarem je krychle (na obrázku první zleva), která je pouze jedna (otáčením dle libovolné osy nové tvary nevznikají). Z druhého tvaru je možné otáčením vyrobit hned 12 dalších tvarů. Třetí a čtvrtý tvar dávají každý 8 nových tvarů.

Vlivem zavedení těchto různých tvarů může být buňka terénu tvořena buď jedním, nebo dvěma různými materiály. Materiálů bylo zatím definováno 5. Materiálem, který vyplňuje prázdný prostor je *vzduch*. Základním materiálem tvořícím půdu je *písek* a hůře prostupný *jíl*. Dále existuje *dřevo* a neprostupný materiál představuje *skála*. Předpokládá se, že budou existovat i další materiály například pro stébla trávy či listy stromů.

Pro uložení jedné buňky do paměti je tedy potřeba minimálně 11b (5b pro specifikaci tvaru, 2 x 3b pro každý materiál). Jelikož nestačí 1B, je potřeba pro každou buňku rezervovat 2B. Nepotřebné bity mohou být využity pro materiály, kterých tak může existovat více (až 5b pro specifikaci prvního, resp. druhého, tedy až 32 různých materiálů). 1b zůstává nevyužit. Rozložení bitů je zobrazeno na obrázku 4.2.



Obrázek 4.2: Uspořádání dat buňky terénu. Šedé bity jsou vyhrazeny pro definici tvaru buňky. Modré a žluté představují první, resp. druhý materiál.

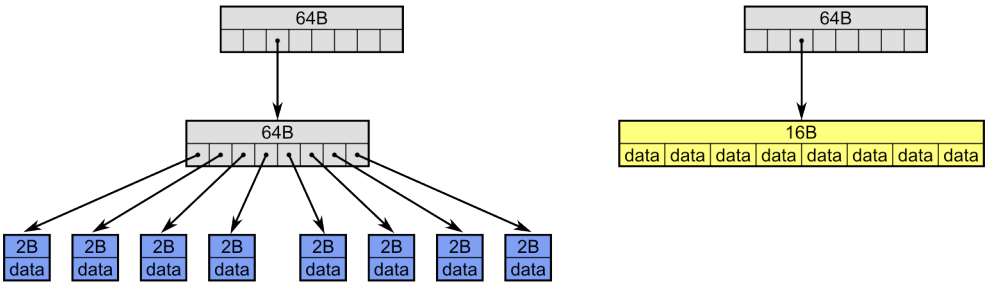
Reprezentace octree

Oktalový strom je možné implementovat buď hierarchicky jako uzly a listy propojené ukazateli, nebo pomocí bezpointerové reprezentace, kdy jsou uzly a listy uspořádány lineárně v paměti. Výhodou lineární reprezentace je, že hledání listu odpovídajícího danému bodu prostoru je rychlejší. Polohu tohoto listu v lineárním poli lze vypočítat, zatímco u pointerové hierarchie je potřeba traverzovat celý strom od kořene [5].

Naopak nevýhodou lineárního pole je, že v paměti musí být vyhrazeno místo pro všechny listy každé větve až do poslední úrovně. Zatímco u hierarchie, pokud to není potřeba, je možné skončit dříve. Tedy dříve se odkazovat na list, který tak zaujímá větší prostor, než list v nižších úrovních. Díky tomu nemusí hierarchie zabírat tolik místa v paměti. Ovšem je zase vyžadována režie pro reprezentaci vnitřních uzlů, které obsahují ukazatele na potomky.

Předpokládá se ale, že herní svět bude z nezanedbatelné části tvořen velkými bloky (např. prázdný prostor nad zemí nebo velké kameny pod zemí), a proto se vyplatí implementovat terén hry pomocí hierarchické struktury využívající ukazatele. Zdali byl předpoklad správný, je uvedeno v 5. kapitole zabývající testováním aplikace.

Aby hierarchická struktura, vlivem přítomnosti ukazatelů, nezabírala tolik místa v paměti, byla ještě vylepšena. Vnitřní uzel obsahující pole ukazatelů na 8 potomků zabírá 8 x 8B, tedy 64B[9], zatímco list využívá pouze 2B (viz výše sekci zabývající se buňkou terénu). Ovšem pokud všechny listy dané větve jsou na stejné úrovni, je možné tyto listy a poslední vnitřní uzel nahradit jedním speciálním uzlem (což je také list), který sdružuje data všech osmi listů dohromady. Takový uzel zabírá v paměti 8 x 2B, tedy 16B, místo původních 64B + 8 x 2B, tedy 80B. V podstatě dochází k vynechání jednoho vnitřního uzlu. Uvedené vylepšení ilustruje obrázek 4.3.



Obrázek 4.3: Porovnání standardní větve octree (levá polovina) a vylepšené větve pomocí speciálního listu (pravá polovina).

Renderování terénu

Nevýhodou objemové reprezentace je její nesnadné zobrazování. Pro renderování herního světa je proto hierarchická struktura octree převáděna do povrchové reprezentace, konkrétně na polygonální síť, která je poté vykreslována místo složitého renderování objemové reprezentace.

Terén není převáděn na polygonální síť najednou, ale po blocích určité velikosti tzv. *chunků*. To je proto, aby se při nahrávání nových oblastí do paměti nebo při změně terénu hráčem nemusela vytvářet znovu polygonová síť celého

[9]64B na 64bitovém operačním systému. Na 32bitovém operačním systému by vnitřní uzel zabíral 8 x 4B, tedy 32B.

herního světa, ale jen daného bloku. Sítě jednotlivých bloků jsou vytvářeny již během nahrávání terénu do paměti. Udržovány jsou tak najednou nejen stromové struktury každého chunku, ale také jejich geometrická reprezentace v podobě polygonální sítě.

Samotnému převodu do povrchové reprezentace nebylo v rámci této práce věnováno příliš pozornosti, proto byla implementována pouze jednoduchá varianta, kdy je pro každou buňku terénu zvlášť vytvořena množina trojúhelníků bez ohledu na sousední buňky. Tyto polygonové sítě jednotlivých buněk jsou poté sjednoceny do jedné množiny trojúhelníků a najednou vykreslovány. Metoda je velmi jednoduchá, ale nevýhodou je, že dochází k redundanci velkého množství polygonů (sousední stěny, které by bylo možné uvažovat pouze jednou) a také existence polygonů, které vůbec není třeba renderovat (stěny buněk uvnitř terénu, které není možné vidět).

Ukázka vyrenderovaného bloku herního světa je obsažena v galerii obrázků ze hry v příloze C.

Ukládání terénu do souborů

Herní svět je nahráván ze souborů uložených na disku počítače. Změněný svět musí být opět do souborů ukládán, aby byl zachován pro příští spuštění hry. Formát ukládaných dat by měl být vhodný jak pro strojové čtení, tak pro čtení lidmi, aby ho bylo možné snadno manuálně upravit (alespoň v této fázi vývoje). Proto není možné ukládat data v binární podobě, ale v textové (myšlena struktura terénu, nikoli samotná data listů definující tvar a materiál buňky, ty v binární podobě být mohou).

Zvolen byl formát JSON, který těmto požadavkům vyhovuje. Další možností by byl například formát XML, který ovšem v porovnání s JSON má větší režijní náklady (myšleno paměťové), jelikož využívá párové tagy [22]. Oba formáty umožňují ukládat stromovou strukturu a jejich výhodou je snadná přenositelnost dat do jiných aplikací např. externího editoru pro tvorbu herního světa (zatím není implementován).

Struktura uloženého bloku herního světa je následující: kořenový objekt je nazván *root* a obsahuje kromě dat popisující potomky také informace o pozici bloku v prostoru (souřadnice bloku). Obecný uzel obsahuje informaci o úrovni uzlu octree (0 odpovídá listům na nejnižší úrovni), specifikaci typu uzlu (zda se jedná o 64B vnitřní uzel či 16B nebo 2B list) a data, kde je buď seznam potomků, nebo samotná data reprezentující tvar resp. materiál buňky. Pro přehlednost může ještě obsahovat popisek specifikující oktant, do kterého uzel patří. Popisek nicméně pro správné fungování nutný není, jelikož oktant je dán pořadím uzlů.

V názvu souboru je obsažena úroveň kořenového objektu (uzlu) a souřadnice bloku, který soubor reprezentuje. Struktura názvu je následující:

$$< rootlevel >< x >< y >< z > .json,$$

kde $< rootlevel >$ zaujímá jednu cifru a souřadnice $< x >$, $< y >$ a $< z >$ zaujímají 3 x 4 cifry. Jelikož je potřeba počítat i se zápornými souřadnicemi, značí první cifra souřadnice znaménko (0 pro kladná čísla, 1 pro záporná čísla). Jednotkou souřadnic je celý blok úrovně $< rootlevel >$. Svět tak může být prakticky neomezený. Přibližně 2000 x 2000 x 2000 bloků (1000 v kladné ose a další 1000 v záporné ose) úrovně maximálně 9, což je blok o šířce 512 buněk. Omezením ovšem může být velikost dostupného místa na disku.

Rozměry herního světa

Nejmenší možná buňka terénu úrovně 0 představuje blok o šířce 1,25 mm. Velikost mravence, ve hře se vyskytujícího, je přibližně 3 mm v případě dělnice a 8 mm u královny. Chunky jsou používány úrovně 4, tedy bloky o šířce 16 buněk, resp. 2 cm. Pro představu je uvedena tabulka šířek úrovní octree (tabulka 4.1).

úroveň	**šířka bloku**	
0	1 buňka	1,25 mm
1	2 buňky	2,50 mm
2	4 buňky	5,00 mm
3	8 buněk	1 cm
4	16 buněk	2 cm
5	32 buněk	4 cm
6	64 buněk	8 cm
7	128 buněk	16 cm
8	256 buněk	32 cm
9	512 buněk	64 cm

Tabulka 4.1: Rozměry bloků herního světa.

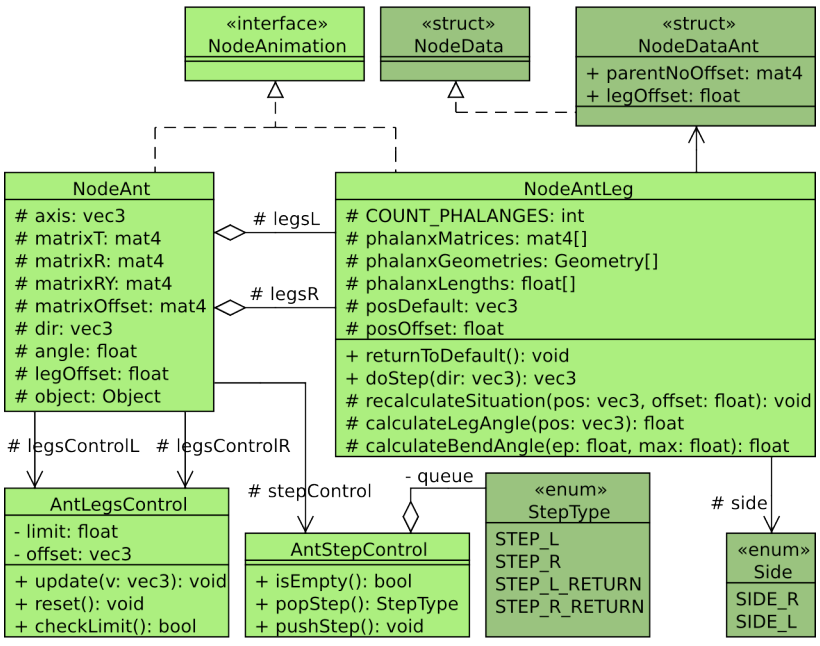
Takto definovaný herní svět může být široký maximálně 2000 x 64 cm, tedy zhruba 1,3 km. Což je pro mravence o velikosti 3 mm více než dostačující.

4.2 Pohyb a ovládání mravence

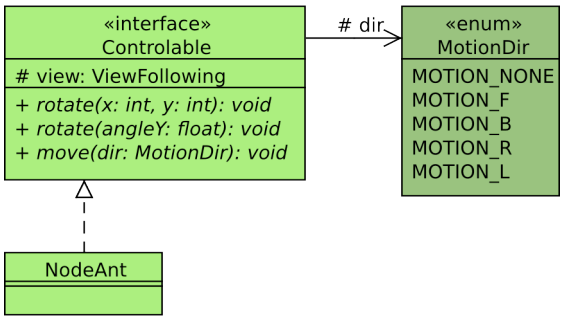
Tato sekce popisuje implementaci pohybu mravence a jeho ovládání. V úvodu sekce je doplněn návrh modulu World o úsek zabývající se právě pohybem mravence. Poté následují části věnované detailům ze samotné implementace. První část se zabývá pohybem a ovládáním celého mravence. Druhá představuje pohyb nohou. Zbývající části jsou věnovány interakci mravence s herním světem.

Model mravence je načítán z obj souboru a je rozdělen na několik samostatných objektů reprezentujících tělo mravence a jednotlivé články mravencovy nohy. Důvodem je jednak skutečnost, že všechny nohy jsou stejné, proto není důvod reprezentovat je v modelu vícekrát, a jednak aby bylo možné články nohou animovat.

Renderovatelný objekt mravence je zastoupen dvěma uzly grafu scény. *NodeAnt*, který reprezentuje tělo, a na něj navázány objekty třídy *NodeAntLeg* reprezentující nohy mravence. Uzel NodeAnt obsahuje logiku pro řízení jednotlivých kroků a realizuje animaci pohybu mravence, zatímco třída NodeAntLeg se stará o správnou polohu článků nohy a jejich animaci. Digram tříd znázorňující tyto dvě třídy a jejich začlenění do grafu scény je na obrázku 4.4.



Obrázek 4.4: Diagram tříd: balíček Graph modulu World - 2. část.



Obrázek 4.5: Diagram tříd: balíček Graph modulu World - 3. část. Rozhraní *Controlable* je poskytováno jak modulu Control pro ovládání hráčova mravence, tak modulu Logic pro řízení autonomních mravenců.

Pohyb mravence

Třída NodeAnt (uzel grafu scény) implementuje rozhraní *Controlable*, které je poskytováno modulu Control a nebo modulu Logic. Toto rozhraní nabízí metody pro ovládání mravence a to jak ze strany hráče, tak pro řízení umělou inteligencí. Konkrétně se jedná o pohyb mravence daným směrem a o rotaci. Rozhraní Controlable je znázorněno v UML diagramu na obrázku 4.5.

Pohyb mravence je animovaný, což znamená, že při každém zavolání funkce *move* se mravenec přímo nepohybuje, ale je pouze vypočítána nová pozice, kam se má během následujících kroků animace mravenec dostat. Jednotlivé animační kroky jsou prováděny až při aktualizaci stavu, tedy při vykonání metody *update* uzlu grafu scény.

Nová pozice je počítána na základě předaného směru, kterým se má mravenec pohnout. Směr je dán výčtovým typem *MotionDir*. Jednotlivé konstanty reprezentující směry pohybu jsou definovány tak, aby je bylo možné kombinovat pomocí logických operátorů. Díky tomu bylo možné snadno vyřešit situace, kdy je stisknuto více kláves najednou (např. pro pohyb dopředu a zároveň doprava - výsledný směr mravence je šikmý).

Pro rotaci mravence poskytuje rozhraní hned dvě metody. Jedna umožňuje otočit mravence o daný úhel a druhá úhel vypočítává na základě souřadnic kurzoru myši na obrazovce. První metoda je vhodná pro modul Logic, kde jsou řízeni autonomní mravenci, zatímco druhá varianta odpovídá ovládání rotace mravence hráčem prostřednictvím myši.

Pohyb nohou

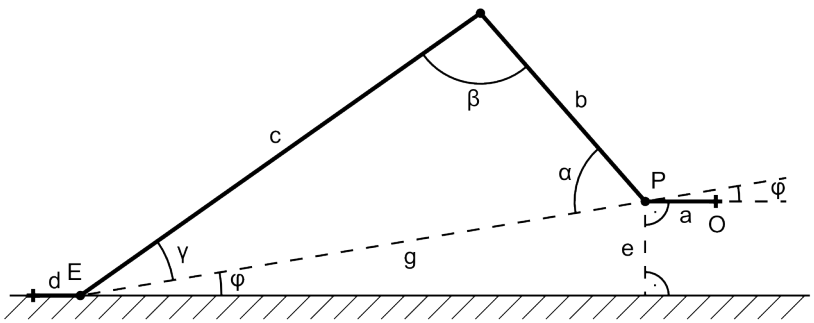
Při implementaci pohybu mravence byla věnována velká pozornost pohybu jeho nohou. Důvodem byla snaha o to, aby pohyb vypadal realisticky (působil přirozeně). Hýbání nohou podle předem specifikované animace, které by mohlo způsobovat viditelné „podkluzování“ nebo jiné nereálné případy, nepřicházelo v úvahu.

Proto je poloha nohy (přesněji místa, kde se noha stýká se zemí - podobně jako u člověka chodidlo) počítána přesně a během chůze je k zemi opravdu „přilepena“. Polohu vůči zemi samozřejmě mění, ale až s novým krokem, kdy je celá noha zvednuta a přenesena na novou pozici.

Noha mravence je složena ze čtyř článků. Úhly jednotlivých kloubů, stejně jako otočení celé nohy, je potřeba v každém aktualizačním kroku scény dopočítávat. Tento výpočet odpovídá procesu inverzní kinematiky. Délky jednotlivých článků jsou známé předem. Známé jsou v každém kroku také polohy těla mravence (místa, kde je noha upevněna k tělu) a koncového efektoru (místa styku posledního článku se zemí).

Výpočet úhlů natočení jednotlivých kloubů je zjednodušen, protože první a poslední články zůstávají stále ve vodorovné poloze a nemění se. Úloha inverzní kinematiky tak spočívá pouze v dopočítání 3 úhlů v trojúhelníku, kde jsou známé délky všech jeho stran.

Výpočet ilustruje obrázek 4.6. Jedná se o levou nohu při pohledu zezadu na pohybujícího se mravence směrem od pozorovatele. Bod O odpovídá místu, kde se noha dotýká těla mravence, zatímco bod E značí koncový efektor (místo styku se zemí) umístěný, díky zjednodušení, již před posledním článkem. Úsečky a, b, c a d reprezentují články nohy.



Obrázek 4.6: Výpočet úhlů kloubů jednotlivých článků nohy mravence.

Za použití kosinové věty je možné vypočítat úhly α a β následujícím způsobem:

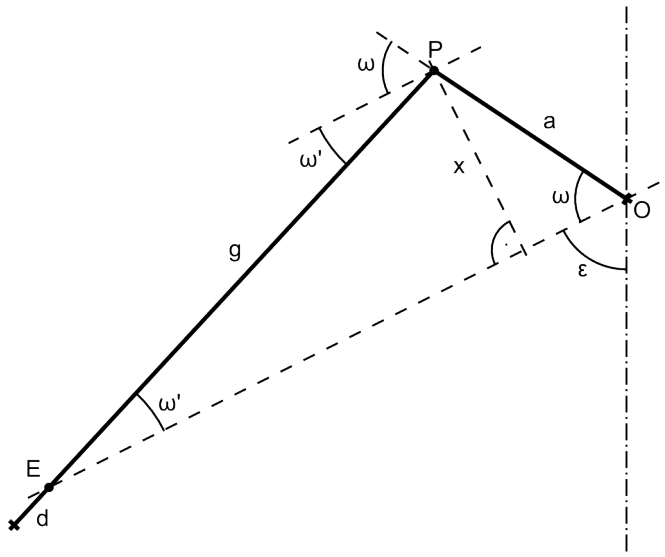
$$\cos\alpha = \frac{b^2 + g^2 - c^2}{2bg} \tag{4.1}$$

$$\cos\beta = \frac{b^2 + c^2 - g^2}{2bc} \tag{4.2}$$

Kde g je vzdálenost koncového bodu prvního článku P a koncového efektoru nohy E (viz obrázek 4.6). Úhel γ lze pak dopočítat na základě součtu vnitřních úhlů trojúhelníku.

Úhel φ se dá vypočítat pomocí vzdálenosti těla mravence od země e (která se vlivem chůze mění) a úsečky g za použití goniometrických funkcí v pravoúhlém trojúhelníku.

Aby noha působila přirozeněji, dochází při pohybu mravence k jejímu prohnutí do strany. Tuto skutečnost ilustruje obrázek 4.7. Tentokrát jde o pohled shora na stejnou nohu. Vertikální čerchovaná čára znázorňuje osu těla mravence. Mravenec se pohybuje směrem nahoru. Ve skutečnosti není prohnutí nohy (ve znázorněné poloze), ani velikost prvního článku, tak výrazné. Zde jsou pro lepší orientaci rozměry zkresleny.



Obrázek 4.7: Výpočet úhlů otočení a ohybu nohy mravence.

Prohnutí je prováděno pouze mezi prvním a druhým článkem. Úhel prohnutí ω je dán velikostí úhlu otočení celé nohy ε (čím menší úhel mezi nohou a osou mravence, tím větší prohnutí nohy) a je potřeba s ním počítat při určování polohy koncového bodu prvního článku P a tedy i při výpočtu vzdálenosti g.

Aby bylo možné prohnutí realizovat, je potřeba také vypočítat úhel ω' (viz obrázek 4.7). Ten lze dopočítat na základě dvou následujících rovností:

$$x = a \sin \omega \tag{4.3}$$
$$x = g \sin \omega' \tag{4.4}$$

Úhel otočení celé nohy ε lze vypočítat snadno, jelikož je známá poloha bodu O a koncového efektoru E. Jedná se o úhel mezi úsečkou danou těmito dvěma body a osou mravence (jeho směrovým vektorem).

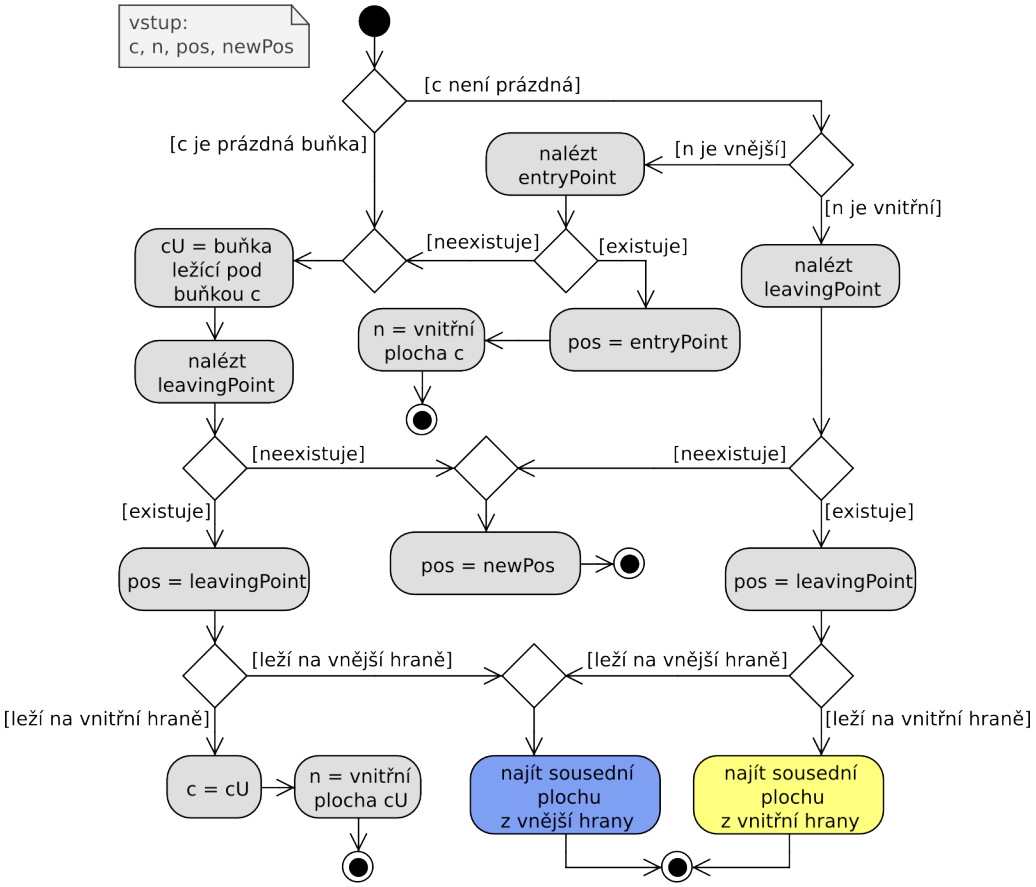
Tím je situace (úhly jednotlivých kloubů včetně prohnutí i otočení celé nohy) kompletně určena a noha mravence může být vykreslena. Stejná procedura probíhá i u ostatních pěti nohou.

Interakce s terénem

Interakce mravence s herním světem spočívá v určení polohy mravence v prostoru tak, aby kopíroval terén a nevznášel se nad ním nebo naopak jím neprocházel.

Terén je reprezentován datovou strukturou octree (viz první sekce této kapitoly), čili určení polohy znamená, najít správnou buňku (list octree), ve které se mravenec nachází, a přesnou polohu uvnitř této buňky. Jelikož buňky terénu nejsou jen krychle, ale jedná se obecně o mnohostěny, je potřeba také určit správnou stěnu neboli plochu, na které mravenec stojí. Mravenec se může pohybovat jak po zemi, tak po bočních stěnách nebo stropě. Proto se může vyskytovat na jakékoli stěně buňky (nezáleží na tom, zda je vodorovná, svislá či šikmá).

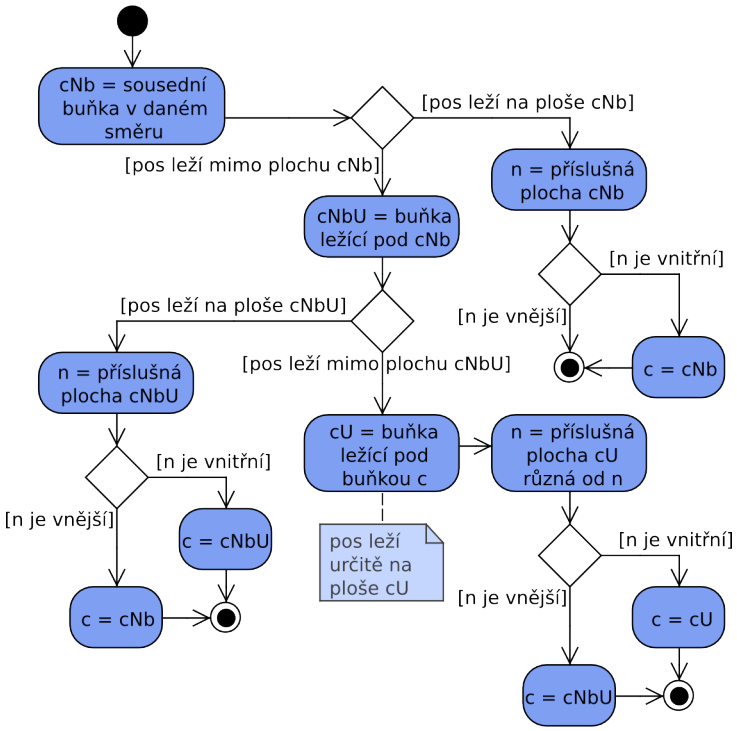
Správná poloha mravence je počítána v každém kroku aktualizace scény. Hledána je vždy nová poloha, kam se může mravenec přesunout. Tento proces je vzhledem k velkému počtu možností, které mohou nastat, poměrně složitý. UML diagram aktivit na obrázku 4.8, 4.9 a 4.10 se snaží proces hledání nové polohy nastínit.



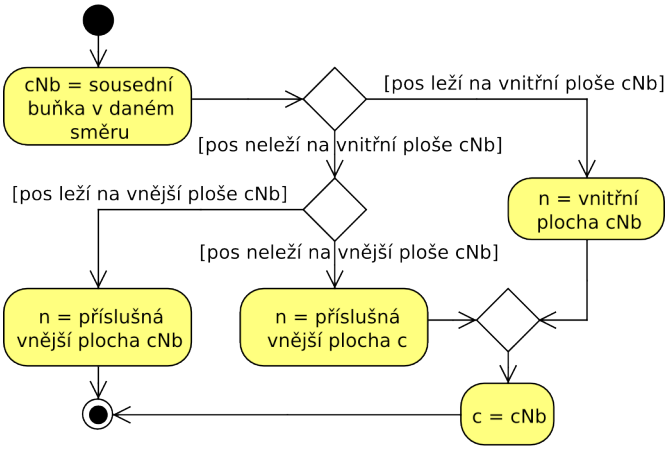
Obrázek 4.8: Diagram aktivit: interakce mravence s terénem - 1. část. Hledání nové polohy mravence. Vstupem je aktuální buňka c, normála reprezentující aktuální plochu n, vektor udávající přesnou polohu pos a vektor předpokládané polohy po vykonání kroku $newPos$. Výstupem jsou aktualizované hodnoty c, n a pos.

Procedura předpokládá správnou předchozí polohu mravence, do které se dostal v minulém kroku. Proto je vstupem buňka terénu, ve které se mravenec nyní nachází, v diagramu označována jako c. Dále vektor n představující normálu aktuální plochy, na které mravenec stojí (nemusí se nutně jednat o stěnu aktuální buňky), a relativní poloha v rámci této plochy reprezentovaná vektorem pos.

Před vstupem do samotné procedury dochází ještě k odhadu výsledné pozice. Jedná se o bod v prostoru, do kterého by se mravenec dostal bez ohledu na terén. Tento bod vzniká tak, že se k aktuální poloze připočte směrový vek-



Obrázek 4.9: Diagram aktivit: interakce mravence s terénem - 2. část. Nalezení sousední plochy přes vnější hranu.



Obrázek 4.10: Diagram aktivit: interakce mravence s terénem - 3. část. Nalezení sousední plochy přes vnitřní hranu.

tor o délce jednoho kroku. Odhadovaná pozice je čtvrtým vstupem procedury pod názvem *newPos*.

O odhad se jedná proto, že není jisté, zda mravenec v této pozici opravdu skončí. Může se stát, že mezi aktuální polohou a nově odhadovanou nastane přechod do jiné roviny. V takovém případě algoritmus vrátí tuto novou rovinu (popřípadě i novou buňku terénu) a najde novou polohu na hraně oddělující obě roviny.

Výstupem algoritmu jsou aktualizované hodnoty *c*, *n* a *pos*, které po skončení procedury odpovídají nové poloze mravence v terénu.

Stěny buněk mohou být buď *vnější*, nebo *vnitřní*. Vnější plocha odpovídá stěně krychle obalující buňku, zatímco vnitřní plocha je taková stěna buňky (mnohostěnu definujícího tvar buňky), která je uvnitř této krychle (nepřekrývá se s žádnou stěnou obalující krychle). Jedná se o šikmé plochy buněk.

Podobným způsobem jsou definovány také *vnější* a *vnitřní* hrany buňky. A to navzdory tomu, že všechny hrany mnohostěnů leží ve stěnách obalující krychle. Vnější hrany korespondují s hranami této krychle, vnitřní odpovídají uhlopříčkám stěn krychle.

Z definic tvarů možných buněk (kompletní seznam viz příloha B) vyplývá například, že buňka má buď jednu vnitřní stěnu, nebo žádnou. Z toho také vyplývá, že buňka, ve které se mravenec nachází, je buď prázdná, nebo obsahuje právě jednu vnitřní stěnu. Další skutečností je, že pokud mravenec leží na vnější ploše, tato plocha patří sousední buňce, nikoli buňce, ve které se mravenec nalézá (pokud leží na horní stěně, tato stěna patří buňce, která je pod aktuální buňkou). Naopak pokud leží na vnitřní ploše, tato plocha patří aktuální buňce.

V diagramu aktivit jsou také definovány výrazy *entryPoint* a *leavingPoint*. Jedná se o body ležící na hranách a určují přesně místa, kde mravenec vstupuje na vnitřní plochu, resp. kde opouští jakoukoli plochu.

V první fázi algoritmu dochází k určení nové polohy (aktualizace parametru *pos*). Pokud mravenec neopustí aktuální plochu (neopustí aktuální buňku, ani nepřejde z vnější plochy na vnitřní či naopak v rámci této buňky), novou polohou se stává předem odhadovaná pozice a algoritmus končí (buňka i plocha zůstávají stejné). V opačném případě se novou polohou stává vstupní bod vnitřní plochy (*entryPoint*) nebo bod, kde mravenec opouští aktuální plochu (*leavingPoint*) a procedura pokračuje hledáním nové buňky, resp. stěny buňky.

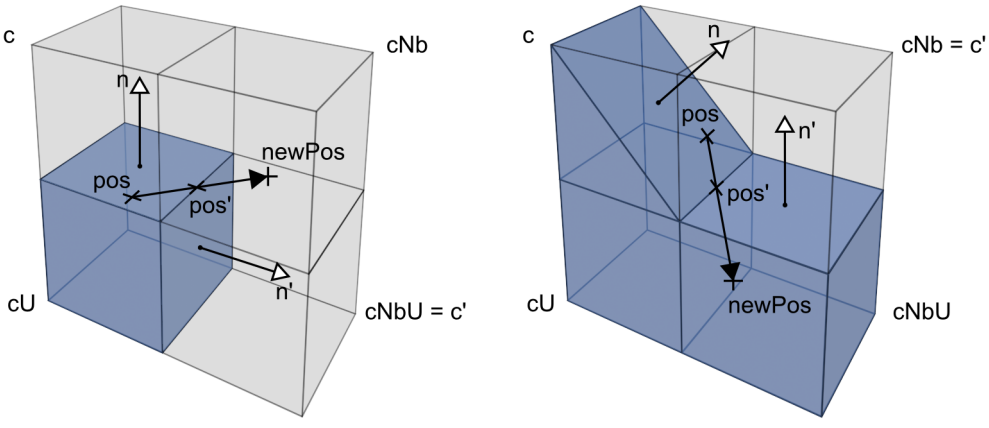
Pokud novou polohou je vstupní bod vnitřní plochy, je situace jednoduchá, neboť buňka zůstává stejná a novou plochou se stává tato vnitřní plocha. Pokud je nová poloha v místě opuštění dané plochy je situace složitější, protože je potřeba novou plochu hledat v rámci sousedních buněk. Pro usnadnění tohoto cíle jsou odlišeny dva případy. Prvním je situace, kdy mravenec aktuální plochu opouští přes vnější hranu (obrázek 4.9), a ve druhém případě opouští plochu přes vnitřní hranu (obrázek 4.10).

V případě vnější hrany je potřeba prozkoumat až 3 sousední buňky. Kromě buňky ležící pod aktuální buňkou (v diagramu označenou cU) ještě sousední ve směru pohybu (cNb) a i buňku ležící pod ní ($cNbU$). Směr *pod* či *nad* je brán z pohledu mravence a nezáleží na jeho otočení.

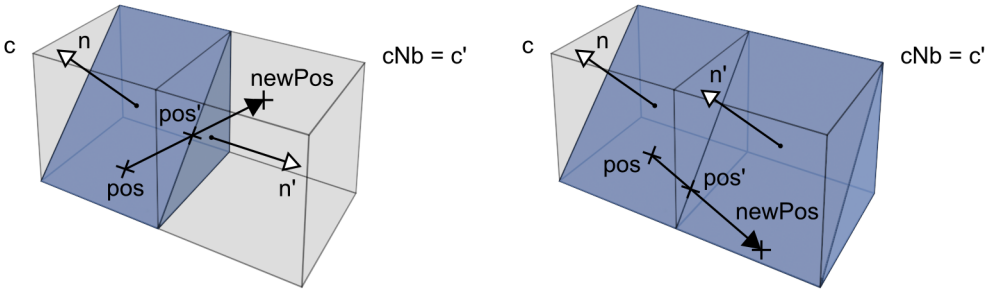
Jednodušší je případ vnitřní hrany, kdy je potřeba přistoupit pouze k jedné sousední buňce. A to buňce ležící před aktuální buňkou ve směru pohybu mravence.

Příklady hledání nové pozice přes vnější hranu jsou uvedeny na obrázku 4.11. V obou případech se mravenec před zahájením nového kroku nachází v levé horní buňce c na pozici pos na ploše definované normálou n. Nová pozice mravence po dokončení algoritmu je označena c', pos' a n'.

V první ukázce (obrázek vlevo) mravenec přeleze přes hranu krychle a ocitá se v pravé dolní buňce, která je opět prázdná. Ve druhé ukázce je původně mravenec na vnitřní ploše, ze které přejde do sousední buňky (pravá horní). Ačkoli putuje z vnitřní plochy, pořád se jedná o přechod přes vnější hranu.



Obrázek 4.11: Příklady hledání nové polohy: sousední plocha přes vnější hranu.



Obrázek 4.12: Příklady hledání nové polohy: sousední plocha přes vnitřní hranu.

Na obrázku 4.12 jsou ukázány příklady hledání nové pozice přes vnitřní hranu. Výchozí poloha mravence je v obou příkladech levá buňka a konečná poloha je uvnitř druhé buňky. V prvním případě mravenec skončí v prázdné buňce na vnější ploše, zatímco v druhém případě skončí opět na vnitřní ploše.

Získání sousední buňky

V procesu hledání nové polohy mravence v herním světě je několikrát požadována sousední buňka terénu. Získání sousední buňky odpovídá hledání sousedního listu v octree a je implementováno podle článku [5], který představuje jednoduchou a efektivní metodu pro realizaci základních operací ve stromových strukturách octree a quadtree (kvadrantový strom) jako je lokalizace bodu, hledání souseda či hledání oblastí. Pro účely této práce je potřeba operace hledání souseda.

K tomu je využíván tzv. *lokační kód* (locational code [5]), který je definován pro každý uzel stromu. Kódem je binární číslo unikátní pro každý list (některé vnitřní uzly sdílejí stejný kód). V případě octree se jedná o vektor 3 čísel, každé pro jednu dimenzi. Definován je následujícím způsobem (příklad pro kód v ose x, stejným způsobem definováno i pro ostatní osy):

$$c_x = (x_{min} \cdot 2^{rootLevel})_b \tag{4.5}$$

Kde x_{min} je minimální x-ová souřadnice buňky odpovídající danému listu a $rootLevel$ zastupuje úroveň kořenového uzlu, což odpovídá počtu úrovní zmenšeného o 1. Nejspodnější vrstva listů je úrovně 0.

Výhodou takto definovaného lokačního kódu je, že umožňuje nad ním vykonávat jednoduché aritmetické operace (např. sčítání kódů mezi sebou nebo přičítání čísla), které zjednodušují zmíněné úlohy.

Pro získání pravého souseda (pravý soused má větší souřadnici) stačí přičíst k lokačnímu kódu aktuální buňky velikost této buňky [5]:

$$\vec{c}_{RN} = \vec{c} + s_b \tag{4.6}$$

Zde se jedná o pravého souseda ve všech 3 osách. Pokud je vyžadována sousední buňka např. pouze ve směru osy z, stačí daný výpočet provést pro z-ovou složku vektoru lokačních kódů. Lokační kódy pro souřadnice ve zbylých osách zůstanou nezměněné. Podobným způsobem lze získat pravého souseda ve směru osy x nebo y, popřípadě kombinace směrů xy, yz apod.

Velikost buňky je možné snadno vypočítat pomocí její úrovně *cellLevel* (velikost buňky úrovně 0 je rovna 1) [5]:

$$s = 2^{cellLevel} \tag{4.7}$$

V případě levého souseda je situace o něco složitější. Jeho velikost nelze snadno určit, jelikož není zřejmé, na které úrovni se nachází. Když se ale odečte od lokačního kódu aktuální buňky velikost nejmenší možné buňky, což je 1, dostane se lokační kód nejmenšího možného souseda:

$$\vec{c}_{LN'} = \vec{c} - 1_b \tag{4.8}$$

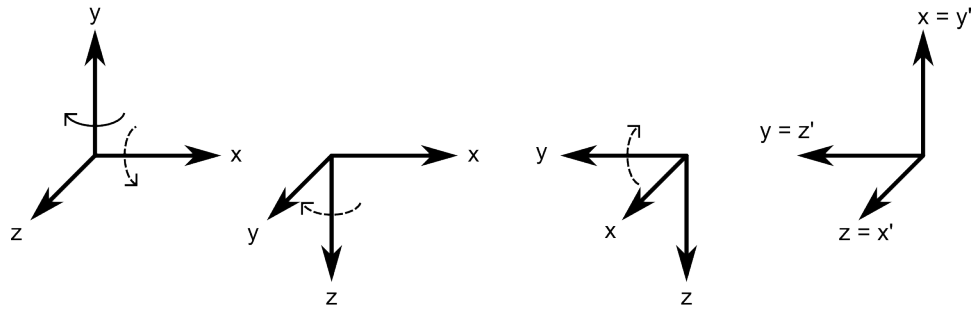
Poté stačí projít strom od kořene a porovnávat příslušné bity lokačních kódů uzlů stromu s tímto kódem nejmenšího možného souseda. Po dosažení listu je získán hledaný levý soused.

Problémy při pohybu mravence v terénu

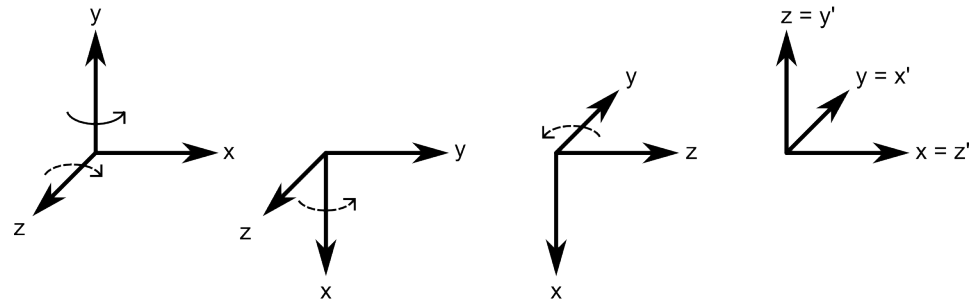
Během implementace interakce mravence s herním světem vzniklo několik problémů. První problém je spojený s otáčením mravence v prostoru. Hráč pohybem myši mění směr mravencova pohybu, a tedy otáčí mravencem kolem osy y (osa y je definována ve směru vzhůru). Pro tuto rotaci je definována samostatná transformační matice. Druhá matice je definována pro rotaci mravence při přechodu do jiné roviny. Výsledná transformace (reprezentovaná maticí modelu) vzniká skládáním těchto transformací (a ještě několika dalších).

Pokud mravenec přechází do jiné roviny přes hranu souběžnou s osu x nebo z, vše probíhá normálně. Problém nastává při rotaci kolem hrany souběžné s osou y. V takovém případě dochází ke ztrátě jednoho stupně volnosti a mravenec se neotáčí tak, jak bylo zamýšleno. Neotočí se kolem hrany, ale rotuje jako při otáčení pomocí myši.

Tuto situaci lze vyřešit tím, že jedno otočení kolem osy y se nahradí třemi jinými rotacemi (nejprve kolem jiné osy, pak požadovaná rotace kolem osy y a následně rotace zpět kolem jiné osy). Příklad je uveden na obrázku 4.13, kdy dochází k nahrazení rotace o $+90°$, a obrázku 4.14, kde se nahrazuje rotace o $-90°$. Po tomto procesu ovšem dochází ke změně souřadného systému. Z obrázků je patrné, že po dokončení operace došlo k požadovanému otočení, ale jednotlivé osy se musí přejmenovat. Při dalším otáčení mravence kolem hrany souběžné s osou y je nutné uvažovat již změněné osy.



Obrázek 4.13: Rotace mravence kolem osy y o $+90°$. Požadovaná rotace je naznačena nepřerušovanou šipkou v prvním obrázku. Čárkované šipky značí nahrazující rotace (první rotace je o $-90°$ kolem osy x, následuje požadované otočení o $+90°$ kolem osy y a konečně rotace zpět o $+90°$ kolem osy x). Změna souřadného systému je naznačena v posledním obrázku.



Obrázek 4.14: Rotace mravence kolem osy y o $-90°$. Požadovaná rotace je naznačena nepřerušovanou šipkou v prvním obrázku. Čárkované šipky značí nahrazující rotace (první rotace je o $+90°$ kolem osy z, následuje požadované otočení o $-90°$ kolem osy y a konečně rotace zpět o $-90°$ kolem osy z). Změna souřadného systému je naznačena v posledním obrázku.

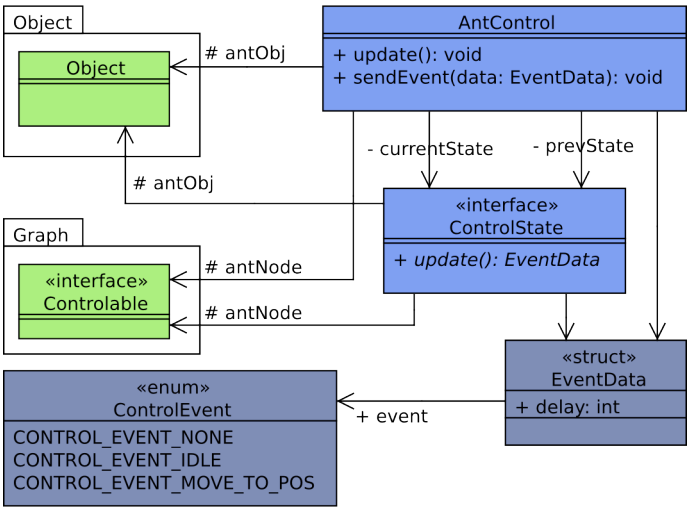
Stejný problém nastává při otáčení kolem šikmých hran, kde y-ová složka je různá od 0. Tyto případy nejsou zatím vyřešeny a je potřeba se jim v budoucnu věnovat. Jejich řešení je již nad rámec této práce.

Další problém spojený s interakcí mravence a herního světa nastává při hledání nové plochy terénu v sousedních buňkách. Výše popsaný algoritmus počítá pouze s buňkami stejné velikosti. Pro kompletní řešení je potřeba algoritmus rozšířit o podporu různě velkých sousedních buněk. Proces získávání sousedního listu octree pomocí lokačních kódů je na tuto situaci již připraven.

4.3 Umělá inteligence autonomních mravenců

Tato sekce kapitoly zabývající se popisem implementace představuje řízení autonomních mravenců. Řízení jejich pohybu tedy umělou inteligenci, nikoli ovládání hráčem, které je popsáno až v další sekci.

Umělá inteligence je realizována pomocí konečného automatu. Každý mravenec se tak nachází v určitém stavu. Podle toho, v jakém stavu se nachází, je prováděna příslušná akce. Během akce může dojít ke změně stavu. Diagram tříd ilustrující realizaci konečného automatu se nachází na obrázku 4.15.



Obrázek 4.15: Diagram tříd: modul Logic - 1. část. Zelenou barvou jsou vyznačeny třídy spadající do modulu World, ostatní modré třídy patří již modulu Logic.

Konečný automat reprezentuje třída *AntControl*. Konkrétní objekt této třídy odpovídá konečnému automatu nad jedním mravencem. Konečný automat obsahuje instanci třídy Object modulu World, která umožňuje získat aktuální pozici mravence v prostoru. Dále obsahuje instanci třídy Controlable, která je implementována uzlem grafu scény modulu World. Toto rozhraní poskytuje metody pro ovládání pohybu mravence (viz předchozí sekce této kapitoly, část *Pohyb mravence*).

Změna stavu konečného automatu je prováděna zasláním události, což je realizováno voláním metody *sendEvent*. Jediným parametrem metody je ukazatel na strukturu *EventData*, která obaluje data předávaná spolu s událostí. Součástí těchto dat je i typ události. Na základě typu je rozhodováno, do kterého stavu automat přechází. Zbývající data jsou potřebná pro realizaci akce spojené s novým stavem.

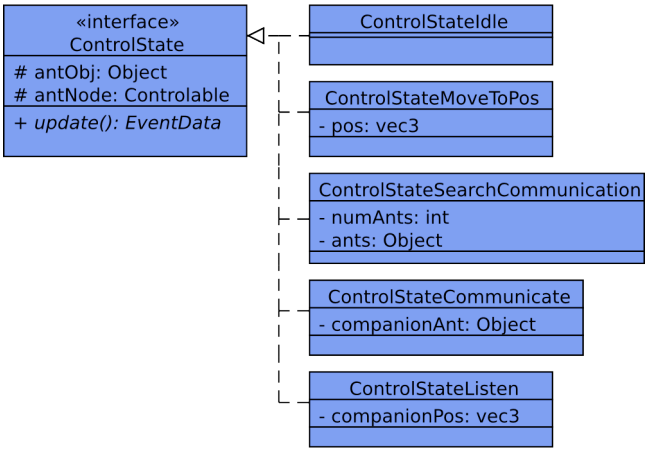
Struktura EventData je základem pro jednotlivé implementace, které od této struktury dědí. Díky tomuto mechanismu je snadné přidávat další stavy do konečného automatu a rozšiřovat tím jeho funkčnost.

Jednotlivé stavy implementují rozhraní *ControlState*. Ukázka realizovaných stavů je uvedena na obrázku 4.16 a příslušná data událostí na obrázku 4.17. Metoda *update* je volána při aktualizaci scény a obsahuje veškerou logiku potřebnou pro provedení akce spojené s daným stavem. K tomuto účelu přebírá třída od konečného automatu (AntControl) instance tříd Object a Controlable modulu World.

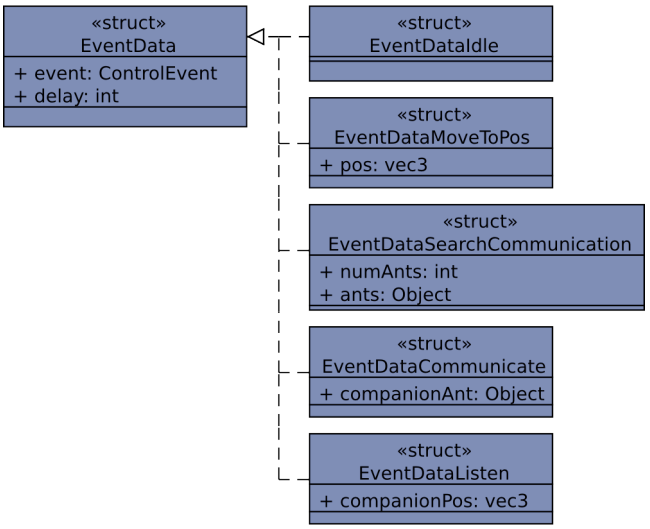
Metoda update vrací data události a říká tak konečnému automatu, jaká událost má být odeslána. Tedy do jakého stavu má přejít, pokud nezůstává v aktuálním stavu.

V této fázi vývoje je pro ukázku funkčnosti umělé inteligence implementován pouze jeden aktivní stav (kromě stavů, které jsou spojené s přímou komunikací, viz další sekce této kapitoly). Tento stav realizuje pohyb mravence na určité místo v prostoru. Jedná se o třídu *ControlStateMoveToPos*. Stavu jsou předávány souřadnice cílového místa reprezentovány polohovým vektorem, struktura *EventDataMoveToPos*.

Pokud mravenec zrovna neprovádí nějakou akci, nachází se ve stavu *ControlStateIdle*. Nejedná se o aktivní stav, neboť s tímto stavem není spojena žádná akce.



Obrázek 4.16: Diagram tříd: modul Logic - 2. část. Hierarchie jednotlivých stavů konečného automatu.



Obrázek 4.17: Diagram tříd: modul Logic - 3. část. Hierarchie struktur reprezentujících data předávaná spolu s událostí konečného automatu.

4.4 Přímá komunikace

Autonomní mravence ve hře se vyskytující nemůže hráč přímo ovládat. Tím se rozumí pomocí klávesnice a myši rovnou říkat, kam se má daný mravenec pohnout. Hráč ovšem může tyto mravence řídit pomocí příkazů, které jim předává prostřednictvím svého mravence.

Jedna z metod předávání příkazů se nazývá *přímá komunikace*. Jedná se o předávání informací mezi dvěma mravenci, kteří jsou v bezprostřední blízkosti. Přímou komunikaci může zahájit hráč, pokud se nachází v blízkosti jiného mravence (do určité minimální vzdálenosti). To provede stisknutím příslušné klávesy. Pokud je poblíž ostatních mravenců více, komunikace je zahájena s nejbližším z nich. Naopak když žádný mravenec v limitní vzdálenosti není, ke komunikaci nedochází.

Stejně jako všechny ostatní akce autonomních mravenců je i komunikace řízena konečným automatem. Mravenec, který vstoupí do komunikačního procesu, přeruší dosavadní akci (např. přesun na určité místo, viz předchozí sekce této kapitoly), zastaví se a zaměří svůj pohled na mravence, který si komunikaci vyžádal. Což je realizováno otočením celého mravence směrem k tazateli.

Nejen řízení autonomních mravenců je realizováno konečným automatem, ale i mravenec ovládaný hráčem se nachází v určitém stavu automatu. Před zahájením komunikace (po stisku příslušné klávesy pro komunikaci) přechází do stavu *SearchCommunication* (viz diagram tříd na obrázku 4.16). V tomto stavu probíhá vyhledávání vhodného mravence, se kterým je možné komunikaci zahájit.

Jelikož se bude autonomních mravenců ve hře vyskytovat jen pár desítek, maximálně několik stovek, vyhledávání probíhá lineárně (sekvenčně) a nebyla implementována žádná speciální datová struktura, která by proces urychlovala. Procházeny jsou postupně všichni mravenci a pro každého je počítána vzdálenost od hráčova mravence. Nejbližší z mravenců, kteří splňují limitní vzdálenost, je vybrán pro komunikaci.

Mravenec ovládaný hráčem po zahájení komunikace přechází do nového stavu *Communicate*, zatímco autonomní mravenec do stavu *Listen*. V tuto chvíli autonomní mravenec očekává nový příkaz. Pokud ho dostane, dojde ke změně jeho stavu na stav, který příkaz vykonává. Jestliže žádný příkaz nedostane (hráč může komunikaci ukončit bez vybraného příkazu stisknutím příslušné klávesy), vrací se do původního stavu a pokračuje ve výkonu předchozí akce.

V tuto chvíli není implementován žádný příkaz, tedy stav konečného automatu provádějící příslušnou akci, který by hráč mohl mravenci předat. Komunikaci lze zatím pouze zahajovat a poté zase ukončovat. Implementace jednotlivých příkazů je nad rámec diplomové práce a není proto její součástí.

Tím byla vyčerpána všechna 4 hlavní témata popisu implementace. Další kapitola se zabývá testováním.

KAPITOLA 5

Testování

V předchozích kapitolách byl představen návrh aplikace realizující počítačovou hru ze světa mravenců a byla popsána implementace 4 základních částí herního enginu. Následuje popis otestování této implementace.

Předmětem testování bylo ověření správného výběru datové struktury pro reprezentaci herního světa. Dále ověření, že pohyb mravence, včetně interakce s herním světem, probíhá dostatečně rychle (tzn. nepřesáhne limit vymezený pro aktualizaci scény) s ohledem na předpokládaný počet mravenců ve hře se vyskytujících. A konečně ověření funkčnosti umělé inteligence, především procesu přímé komunikace.

5.1 Datová struktura reprezentující herní svět

U datové struktury reprezentující herní svět byla zkoumána paměťová náročnost. Vzhledem k tomu, že nebyla věnována pozornost převodu objemové reprezentace na povrchovou, ale byla implementována co nejjednodušší varianta, nehraje v tuto chvíli velkou roli doba trvání renderingu.

Výsledky měření paměťové náročnosti jsou uvedeny v tabulce 5.1. Testování proběhlo na dvou blocích terénu o velikosti 16x16x16 buněk. Podzemní blok a nadzemní. Oba bloky jsou dostatečně složité, tzn. obsahují kromě větších listů octree i nemalé množství listů maximální hloubky. Reprezentují situaci, kdy došlo k úpravě terénu hráčem, který vyhloubil mraveniště. Jedná se o reálná data, která se mohou ve hře vyskytovat. Obrázky těchto bloků jsou obsaženy v galerii v příloze C, včetně ukázky podzemí.

	BP	**P**	$\mathbf{P_{max}}$	**VP**	$\mathbf{VP_{max}}$
podzemní blok	8,2 kB	6,9 kB	45,6 kB	**2,3 kB**	12,9 kB
nadzemní blok	8,2 kB	8,0 kB	45,6 kB	**3,1 kB**	12,9 kB

Tabulka 5.1: Paměťová náročnost jednotlivých implementací octree reprezentujících herní svět. *BP* - bezpointerová implementace, *P* - standardní pointerová implementace, *VP* - vylepšená poinetrová implementace (vybraná pro tuto hru). Sloupce označené *max* představují případy úplně zaplněného stromu do maximální hloubky.

Pro srovnání obsahuje tabulka paměťové nároky i jiných implementací octree, které nebyly realizovány. První sloupec představuje bezpointerovou implementaci, kdy jsou pouze listy ukládány do lineárního pole. Sloupce označené písmenem *P* reprezentují standardní pointerovou implementaci (hierarchické uspořádání), tedy takovou, kde jsou dva druhy uzlů. Vnitřní uzel (sestávající z 8 ukazatelů na potomky) a list (obsahující data buňky terénu). Poslední dva sloupce představují vylepšenou pointerovou implementaci doplněnou o speciální typ listu (viz kapitola 4, sekce *Herní svět*), která byla použita pro tuto hru.

U implementací používajících ukazatele jsou také uvedeny velikosti potřebné paměti při úplném zaplnění stromu v každé větvi do maximální hloubky. Jedná se o případy, kdy je celý blok sestaven pouze z buněk nejmenší velikosti. Bezpointerová implementace ve své podstatě povoluje pouze takové případy[10], proto je uvedena jen jedna varianta.

Z naměřených hodnot vyplývá, že ačkoli hierarchická reprezentace (sloupec *P*) vyžaduje pro ukázkové bloky méně paměti než lineární (*BP*), v nejhorším případě může být výrazně náročnější (P_{max}), zatímco vylepšená implementace při maximálním zaplnění (VP_{max}) není o moc horší než bezpointerová. Předpokládá se, že herní svět bude z velké části tvořen spíše méně složitými bloky, nicméně některé bloky se po zásahu hráčem mohou maximálnímu zaplnění blížit, proto je potřeba i s těmito případy počítat.

Z výsledku testu vyplývá, že vybraná varianta (*VP*) je pro tuto hru vhodná.

[10]Platí pro případ, kdy jsou ukládány pouze listy. Bezpointerovou implementaci je možné realizovat i tak, že budou ukládány také vnitřní uzly, které mohou reprezentovat listy vyšších úrovní. I tak je ale potřeba rezervovat v paměti místo pro celý strom do maximální hloubky. Je tedy více náročná na paměť, a proto není pro hru vhodná a ani zde není uváděna.

5.2 Pohyb mravence a interakce s herním světem

Aktualizace scény probíhá 60krát za vteřinu, čili pro výpočet fyziky všech objektů je rezervováno přibližně 16,7 ms. Za tuto dobu je potřeba mimo jiné vypočítat polohu všech mravenců během jejich pohybu. Pro ověření, že pohyb mravence byl navržen a implementován dostatečně efektivně, bylo provedeno měření časové náročnosti aktualizace scény pro různé počty mravenců.

Výsledky testování jsou uvedeny v tabulce 5.2. Srovnávány jsou následující 4 případy:

- základní pohyb mravence, kdy nedochází k výpočtu úhlů jednotlivých kloubů nohy ani k interakci mravence s herním světem (1. řádek),
- pohyb mravence spolu s výpočtem kloubů (2. řádek),
- pohyb mravence spolu s interakcí s terénem (3. řádek),
- kompletní pohyb mravence včetně výpočtu kloubů a interakce s terénem (4. řádek).

Časy zmíněné v tabulce jsou vždy průměrem 1000 naměřených hodnot aktualizace scény.

	předmět měření	**počet mravenců**				
		10	**50**	**100**	**500**	**1000**
čas [ms]	**pohyb**	0,009	0,025	0,052	0,245	0,542
	pohyb + výp. kloubů	0,051	0,208	0,398	2,012	4,080
	pohyb + interakce	0,015	0,049	0,089	0,415	0,866
	pohyb + výp. kloubů + interakce	0,060	0,240	0,454	2,251	4,483

Tabulka 5.2: Časová náročnost výpočtu polohy mravenců. Předmět měření *pohyb* - základní pohyb mravence bez pohybu nohou a bez interakce s terénem, *výp. kloubů* - výpočet kloubů jednotlivých článků nohou, *interakce* - interakce mravence s herním světem (výpočet polohy v terénu). Uvedené časy jsou v ms.

Implementovaný algoritmus pro výpočet interakce mravence s terénem vyžaduje stejnou hloubku všech listů octree (viz kapitola 4, sekce *Pohyb a ovládání mravence*). Testovací scéna obsahuje listy 1. úrovně (maximální hloubka je na úrovni 0).

Naměřené hodnoty ukazují, že klíčovým aspektem pohybu mravence je výpočet úhlů jednotlivých kloubů nohy mravence, nikoli interakce s herním světem, která probíhá podstatně rychleji. Doba celkového pohybu v případě 1000 mravenců představuje pouze přibližně čtvrtinu vyměřeného limitu, což znamená, že pro očekávaný počet mravenců (maximálně několik set) je efektivita výpočtu polohy mravence dostačující.

5.3 Umělá inteligence a přímá komunikace

Důkazem funkčnosti konečného automatu řídícího pohyb autonomních mravenců je samotná aplikace, která se nachází na přiloženém CD spolu s několika videi. Tato videa ilustrují pohyb mravenců mezi předem danými body v prostoru i proces přímé komunikace, který je také ovládán konečným automatem.

Měřena byla pouze doba procesu vyhledávání nejbližšího mravence pro potřeby přímé komunikace, která nesmí přesáhnout limit aktualizace scény. Měření bylo provedeno pro různý počet mravenců a jeho výsledky jsou uvedeny v tabulce 5.3.

	počet mravenců				
	1000	**2000**	**3000**	**4000**	**5000**
čas [ms]	0,020	0,042	0,067	0,098	0,128

Tabulka 5.3: Časová náročnost vyhledávání nejbližšího mravence během procesu přímé komunikace. Uvedené časy jsou v ms.

Vyhledávání probíhá lineárně, což dokazují i uvedené hodnoty. Naměřená doba výpočtu je zanedbatelná ve srovnání s výpočtem polohy mravence uvedeném v předchozí sekci. Lineární vyhledávání postačuje pro potřeby této hry.

Veškeré testy uvedené v této kapitole probíhaly na počítači s procesorem Intel® Core™ i7-860 8×2,80 GHz, operační pamětí 4 GB a 64bitovým operačním systémem Linux.

Závěr

Náplní této práce bylo navrhnout a implementovat počítačovou hru týkající se života mravenců. Nejprve byla hra představena, jakého je žánru, co je jejím cílem a z jakých herních mechanismů se skládá. Poté následovalo nastínění návrhu herního enginu jako celku. Největší pozornost byla věnována detailnímu popisu implementace. V závěru práce byla implementace otestována.

Implementace byla zaměřena na 4 samostatné části herního enginu. Jednalo se o datovou strukturu reprezentující herní svět, která byla implementována pomocí octree. Dále pohyb mravence a jeho ovládání prostřednictvím klávesnice a myši. Součástí pohybu mravence byla také interakce s herním světem. Třetí část zahrnovala implementaci konečného automatu pro řízení autonomních mravenců. Poslední část představovala realizaci přímé komunikace mezi mravencem ovládaným hráčem a mravencem řízeným počítačem. Důležitou složkou přímé komunikace bylo hledání nejbližšího mravence, které bylo implementováno pomocí lineárního vyhledávání.

Prostřednictvím několika testů bylo ověřeno, že datová struktura pro reprezentaci terénu vyhovuje požadavkům, pohyb mravence spolu s interakcí s herním světem byl implementován dostatečně efektivně, stejně jako proces přímé komunikace, aby během aktualizace scény mohlo být obslouženo požadované množství mravenců.

V budoucnu je potřeba se zaměřit na vyřešení problémů vzniklých během implementace 4 hlavních částí herního enginu, mezi které patří např. rozšíření podpory různé hloubky listů octree během interakce mravence s terénem nebo otáčení mravence při přechodu do jiné roviny. Dále je potřeba vylepšit transformaci objemové reprezentace terénu na povrchovou pro účely renderingu. Poté bude možné se věnovat implementaci zbývajících herních mechanismů.

Literatura

[1] ŠEDIVÝ, Martin. *ANTS: Design dokument hry.* Semestrální projekt. ČVUT v Praze, Fakulta elektrotechnická, 2015.

[2] WERBER, Bernard. *Mravenci.* Překlad Richard Podaný. 2. vyd. Praha: Knižní klub, 2005, 317 s. ISBN 80-242-1378-8.

[3] POKORNÝ, Zbyněk. Komunikace mravenců. In: *ChovZvířat* [online]. 2015-01-21 [cit. 2015-12-14]. Dostupné z: `http://www.chovzvirat.cz/clanek/665-komunikace-mravencu/`

[4] FIEDLER, Glenn. Fix Your Timestep. In: *Gaffer on Games* [online]. [cit. 2015-12-21]. Dostupné z: `http://gafferongames.com/game-physics/fix-your-timestep/`

[5] FRISKEN, Sarah F.; PERRY, Ronald N. Simple and Efficient Traversal Methods for Quadtrees and Octrees. *Journal of Graphics Tools* [online]. 2003, roč. 7, č. 3 [cit. 2015-12-25]. Dostupné z: `http://www.merl.com/publications/docs/TR2002-41.pdf`

[6] OpenGL API Documentation. In *OpenGL: The Industry Standard for High Performance Graphics* [online]. [cit. 2015-12-15]. Dostupné z: `http://www.opengl.org/documentation/`

[7] OpenGL Shading Language. In *OpenGL: The Industry Standard for High Performance Graphics* [online]. [cit. 2015-12-15]. Dostupné z: `http://www.opengl.org/documentation/glsl/`.

[8] *GLEW: OpenGL Extension Wrangler Library* [online]. [cit. 2015-12-15]. Dostupné z: `http://glew.sourceforge.net/`.

[9] *SFML: Simple and Fast Multimedia Library* [online]. [cit. 2015-12-15]. Dostupné z: `http://www.sfml-dev.org/`.

[10] *OpenGL Mathematics* [online]. [cit. 2015-12-15]. Dostupné z: `http://glm.g-truc.net/0.9.7/index.html/`.

[11] *RapidJSON* [online]. [cit. 2015-12-15]. Dostupné z: `http://rapidjson.org/`.

[12] *Minecraft* [online]. [cit. 2015-12-13]. Dostupné z: http://minecraft.net

[13] Anthill. In *Thumbstar* [online]. [cit. 2015-12-14]. Dostupné z: http://www.thumbstar.com/games/anthill

[14] Ant Raid. In *Prank* [online]. [cit. 2015-12-14]. Dostupné z: http://prankentertainment.com/?page=games&gameid=1

[15] Ant Run. In *Make Fire Interactive* [online]. [cit. 2015-12-14]. Dostupné z: http://make-fire.com/Make_Fire_Interactive/Make_Fire_Interactive/Entries/2012/1/26_Ant_Run_(game).html

[16] SimAnt: The Electronic Ant Colony. In *My Abandonwar* [online]. [cit. 2015-12-14]. Dostupné z: http://www.myabandonware.com/game/simant-the-electronic-ant-colony-197

[17] Empire of the Ants. In *Moby Games* [online]. [cit. 2015-12-14]. Dostupné z: http://www.mobygames.com/game/empire-of-the-ants

[18] Ant. In *YoYo Games* [online]. [cit. 2015-12-14]. Dostupné z: http://sandbox.yoyogames.com/games/49421

[19] *ETeeski: Indie Games and How to Make Video Games* [online]. [cit. 2015-12-14]. Dostupné z: http://www.eteeski.com/

[20] Game engine. In *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, edit. 2015-02-25 [cit. 2015-12-12]. Dostupné z: http://en.wikipedia.org/wiki/Game_engine

[21] Game design document. In *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, edit. 2015-08-11 [cit. 2015-12-13]. Dostupné z: http://en.wikipedia.org/wiki/Game_design_document

[22] JSON. In *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, edit. 2015-12-14 [cit. 2015-12-15]. Dostupné z: http://en.wikipedia.org/wiki/JSON

[23] Wavefront OBJ File Format Summary. In *FileFormat.Info: The Digital Rosetta Stone* [online]. [cit. 2015-12-16]. Dostupné z: http://www.fileformat.info/format/wavefrontobj/egff.htm

[24] *Unified Modeling Language (UML)* [online]. [cit. 2015-12-16]. Dostupné z: http://www.uml.org/

PŘÍLOHA A

Seznam použitých zkratek

3D Trojrozměrný (z angl. three-dimensional)

API Rozhraní pro programování aplikací (z angl. application programming interface)

CPU Hlavní procesor (z angl. central processing unit)

GLEW Rozhraní pro správu rozšíření OpenGL (z angl. OpenGL Extension Wrangler Library) [8]

GLM Knihovna specializovaná na matematiku pro grafický software (z angl. OpenGL Mathematics) [10]

GLSL Jazyk pro OpenGL shadery (z angl. OpenGL Shading Language) [7]

GPU Grafický procesor (z angl. graphics processing unit)

JSON Datový formát pro přenos objektů sestávajících z párů klíč-hodnota (z angl. JavaScript Object Notation) [22]

OpenGL Sofwarové rozhraní pro grafický hardware (z angl. Open Graphics Library) [6]

SFML Knihovna pro správu multimediálních komponent počítače (z angl. Simple and Fast Multimedia Library) [9]

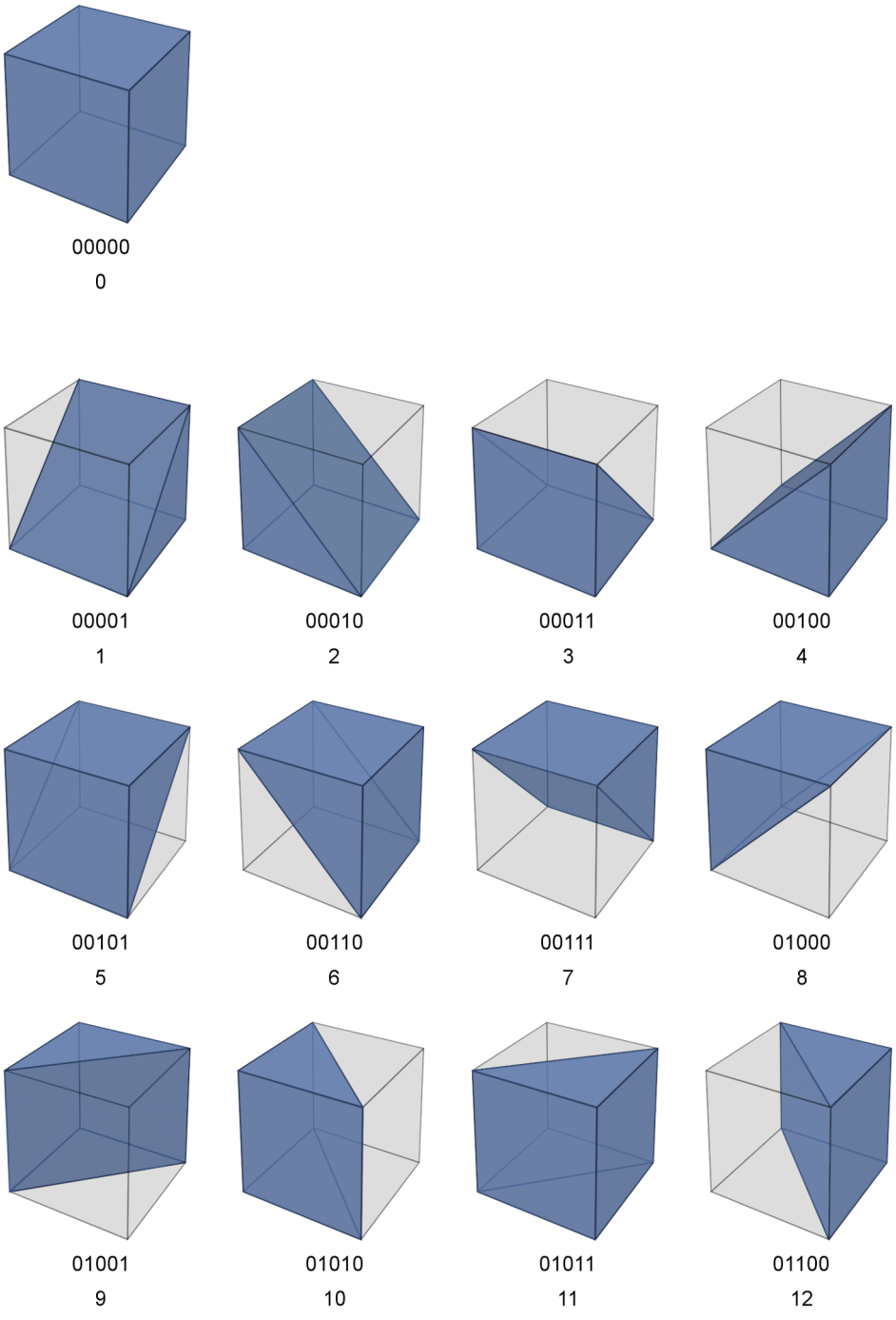
TD Žánr počítačových her tzv. „věžovka“ (z angl. tower defense)

UML Specifikace pro navrhování a dokumentaci programových systémů (z angl. Unified Modeling Language) [24]

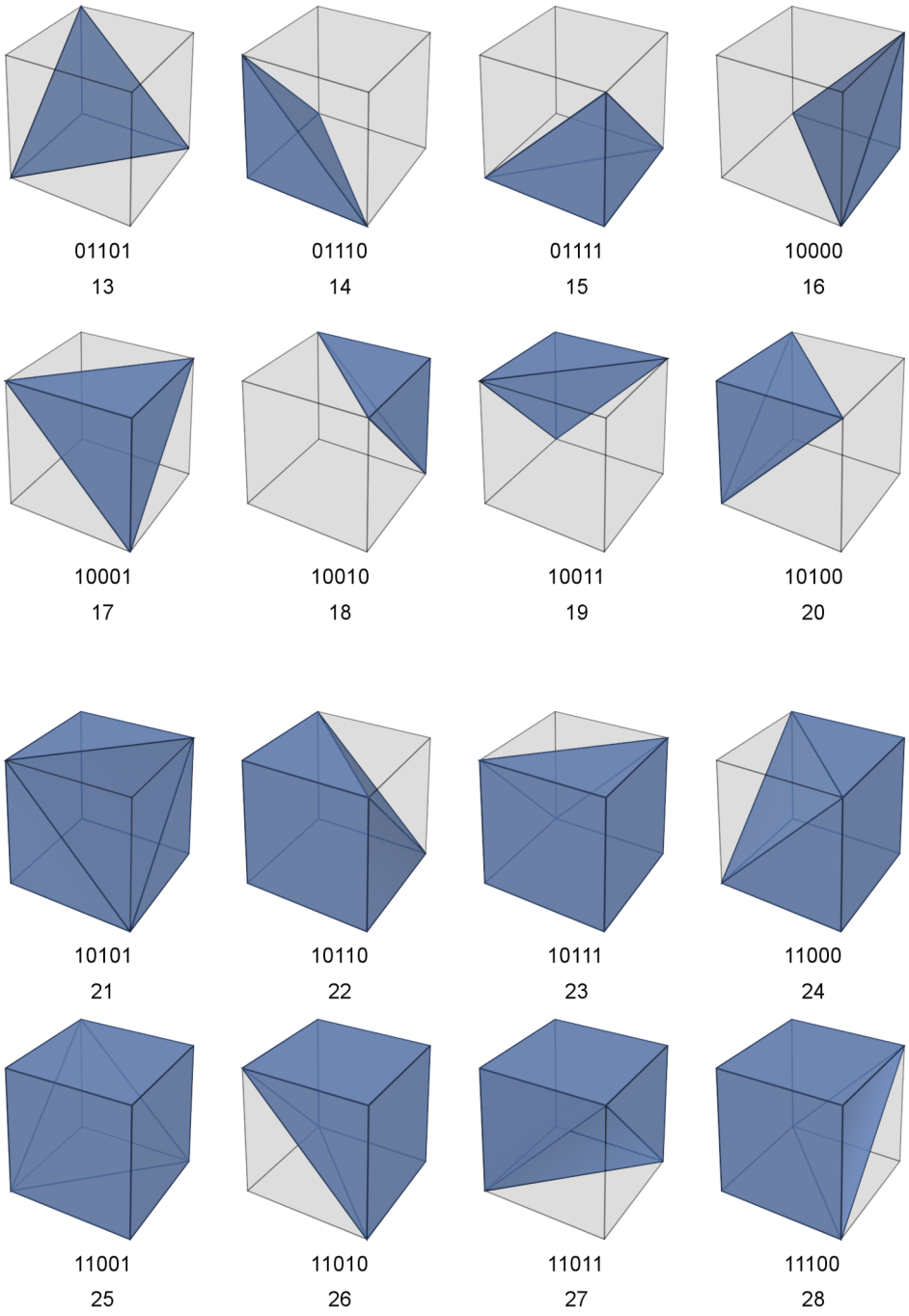
PŘÍLOHA B

Kompletní seznam tvarů buněk terénu

Terén (herní svět) je sestaven z buněk různých tvarů, nejen krychlí (více viz 4. kapitola: *Popis implementace*, sekce *Herní svět*). Definováno bylo celkem 29 různých tvarů buněk terénu. Jejich kompletní seznam je uveden na obrázcích B.1 a B.2.



Obrázek B.1: Tvary buněk terénu - 1. část.



Obrázek B.2: Tvary buněk terénu - 2. část.

PŘÍLOHA C

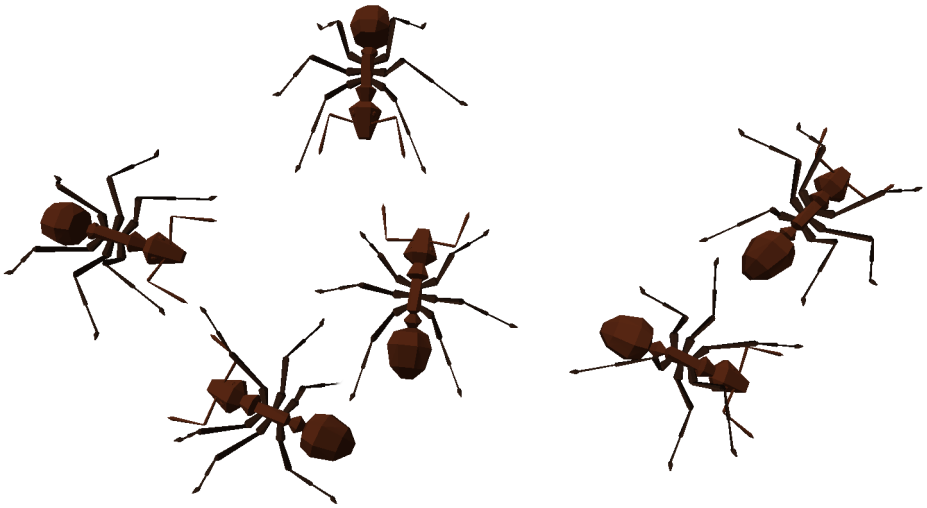
Galerie obrázků ze hry

Na obrázcích C.1 a C.2 je vyobrazen hlavní hrdina hry mravenec - dělnice.

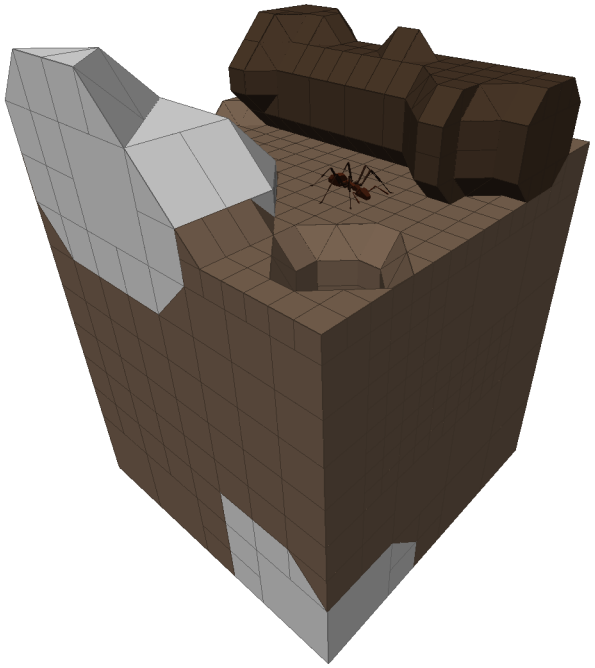
Následující obrázky C.3 a C.4 představují ukázkové bloky terénu ze dvou různých pohledů. V obou případech se jedná o stejné dva bloky nad sebou. Spodní, který je zcela pod povrchem, reprezentuje podzemní blok. Druhý obsahuje tenkou vrstvu půdy (tloušťka jedné buňky nejmenší velikosti) a prázdný prostor nad ní - nadzemní blok. Světlejší hnědá barva představuje základní půdu (písek), tmavší hnědá zastupuje dřevo (zde kousek větvičky) a konečně šedá barva reprezentující pro mravence neprostupné kameny.

Poslední dvojice obrázků C.5 a C.6 zobrazuje vnitřní část podzemního bloku, která má představovat vyhloubené mraveniště s přístupovou chodbou. První obrázek zachycuje mraveniště z této chodby, zatímco na druhém obrázku je opačný pohled z podzemního prostoru do ústí chodby.

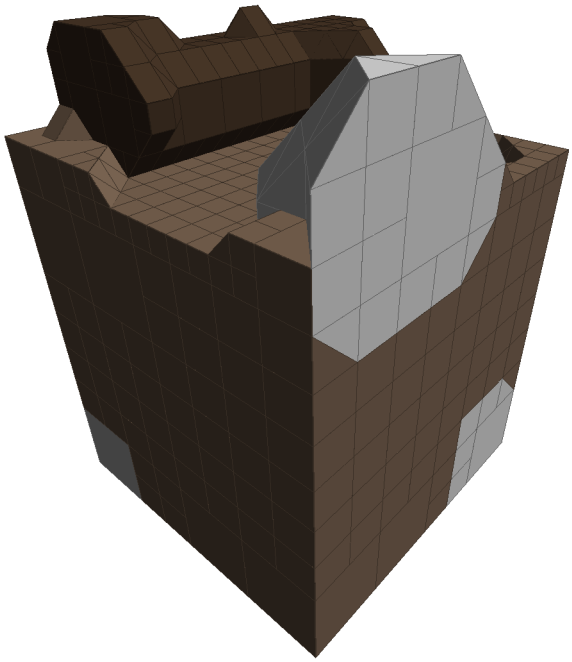
Obrázek C.1: Ukázka vyrenderovaného mravence.



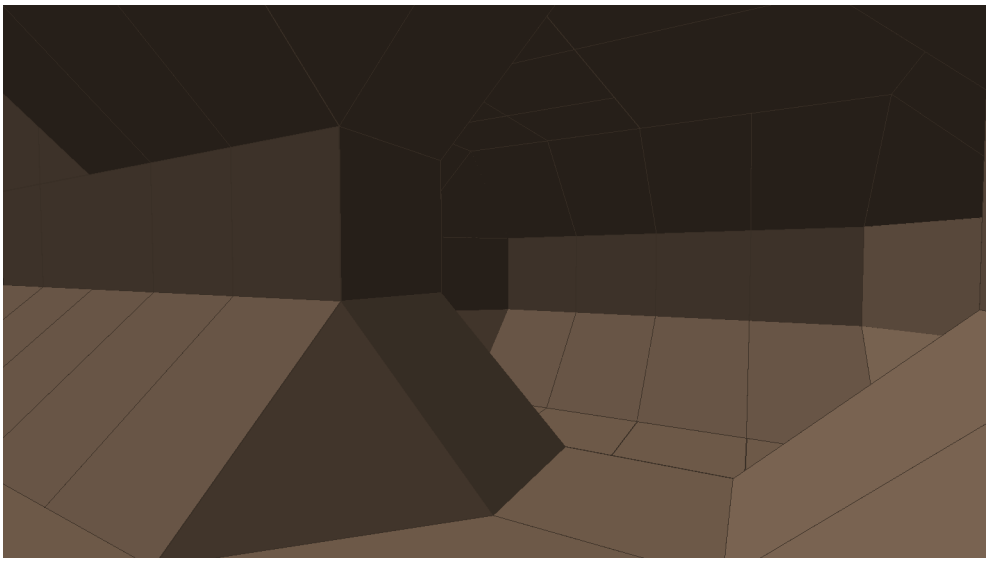
Obrázek C.2: Ukázka skupiny mravenců z ptačí perspektivy.



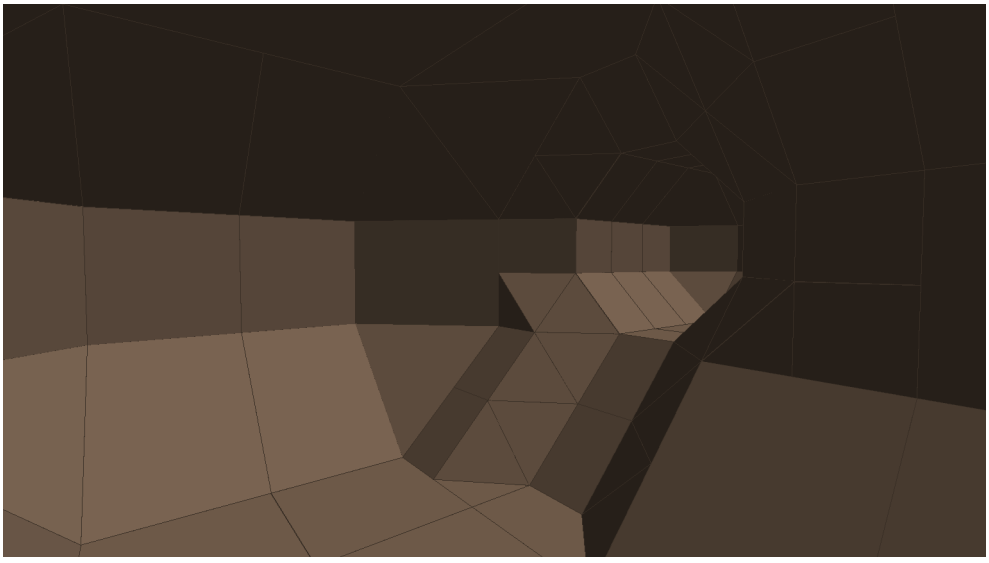
Obrázek C.3: Ukázka dvou bloků terénu - 1. pohled.



Obrázek C.4: Ukázka dvou bloků terénu - 2. pohled.



Obrázek C.5: Ukázka terénu - podzemní mraveniště.



Obrázek C.6: Ukázka terénu - ústí podzemní chodby.

PŘÍLOHA D

Obsah přiloženého CD

- `README.txt` ..... stručný popis obsahu CD, návod na ovládání aplikace
- `build` ..... adresář se spustitelnou aplikací
- `doc` ..... dokumentace zdrojových kódů
- `gallery`
  - `photo` ..... galerie obrázků ze hry
  - `video` ..... ukázková videa ze hry
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `designdoc.pdf` ..... design dokument hry ve formátu PDF