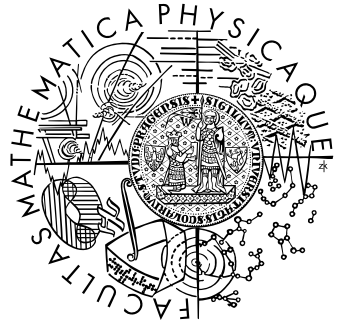
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Štěpán Poljak

Tahová strategie

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.

Studijní program: Informatika, Obecná informatika

2010

Děkuji vedoucímu mé bakalářské práce RNDr. Filipu Zavoralovi, Ph.D. za pomoc, rady a připomínky při jejím vypracování. Děkuji také svým rodičům a celé rodině za všeobecnou podporu při studiu.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 3.8.2010 Štěpán Poljak

Obsah

Název práce: Tahová strategie
Autor: Štěpán Poljak
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.
e-mail vedoucího: Filip.Zavoral@mff.cuni.cz

Abstrakt: Cílem práce je návrh a implementace síťové tahové strategie pro dva a více hráčů, kde jednotliví hráči spravují své město, armády, zdroje surovin a jednotky, bojují proti ostatním hráčům a zdokonalují své hrdiny. Hra využívá 2D grafické rozhraní a umožňuje hru přes lokální síť na architektuře klient-server. Program dále obsahuje umělou inteligenci pro počítačem řízeného hráče. Práce se také zabývá srovnáním podobných existujících projektů, popisem možných přístupů k řešení umělé inteligence v oblasti rozhodování a analýzou implementačních problémů převážně v oblasti síťové komunikace a umělé inteligence.

Klíčová slova: *Tahová strategie, umělá inteligence, rozhodování, síťová aplikace, Heroes, programování her*

Title: Turn-based strategy
Author: Štěpán Poljak
Department: Department of software engineering
Supervisor: RNDr. Filip Zavoral, Ph.D.
Supervisor's e-mail address: Filip.Zavoral@mff.cuni.cz

Abstract: The aim of this thesis is the design and implementation of a network turn-based strategy for two and more players where every single player governs his towns, armies, resources and units, fights against other players and develops his heroes. Game contains 2D graphic interface and provides a possibility to play over local network based on client-server architecture. Program also contains artificial intelligence for computer player. Thesis also contains comparison with similar existing projects, description of possible aprroaches to solution of artificial intelligence in area of decision making and analysis of implementation problems, especially in area of network communication and artificial intelligence.

Keywords: *Turn-based strategy, artificial inteligence, desicion making, network application, Heroes, game programming*

1 Úvod

1.1 Motivace

Naše práce je inspirována převážně herní sérií Heroes of Might and Magic, která se řadí k nejúspěšnějším zástupcům svého žánru, tedy tahových strategií. Motivací bylo implementovat zjednodušenou verzi této hry, která by se hrála po lokální síti a včetně vytvoření vlastního počítačového protihráče.

Podle předlohy je také pojmenovaný náš program, který je součástí práce. Jeho pracovní jméno je **HINT**, což je zkrácený název pro **H**eroes **In**spired **T**urn-based Strategy[1]. Zvolené pojmenování by mělo věrně vystihovat myšlenku práce. V následujícím textu se na náš program budeme odkazovat převážně pod tímto názvem.

1.2 Cíle práce

Cílem práce je navrhnout a implementovat tahovou strategickou hru pro dva a více hráčů, kde alespoň jeden z hráčů je řízený člověkem a další hráči mohou být řízeni jak člověkem, tak případně také počítačovým hráčem, tedy umělou inteligencí. Jedná se o hru určenou primárně pro lokální síť, avšak je možné hrát ji také na jednom sdíleném počítači. S tím je spojena především analýza síťové komunikace a analýza počítačového hráče, kde je nutné najít pro něj rozhodovací techniku. Hra by měla také poskytnout jednoduchou herní grafiku se základními animacemi a intuitivní uživatelské rozhraní.

1.3 Struktura práce

Práce se skládá z několika po sobě navazujících částí.

První část je tvořena druhou a třetí kapitolou a poskytuje základní úvod do problematiky. V první řadě jsou ve druhé kapitole stručně shrnuty základní herní principy, které je dobré pro další analýzu znát. Třetí kapitola se zabývá srovnáním s nejpodobnějším softwarem.

[1] Překlad do češtiny by zněl zhruba „Tahová strategie inspirovaná Heroes“. Skloňovat budeme podle vzoru mužského neživotného „hrad“, tedy například 2. pád HINTu jako hradu.

Druhá část se skládá ze čtvrté kapitoly a popisuje analýzu problémů ohledně síťové architektury a komunikace.

Třetí část je pokryta pátou, šestou, sedmou a osmou kapitolou a zabývá se analýzou a teorií okolo počítačového hráče. Začíná v páté kapitole určením problémů, které musí počítačový hráč hlavně řešit a určením důležitosti těchto vybraných problémů. Navazující šestá kapitola je teoretická a podává přehled rozhodovacích technik, které by bylo možné pro počítačového hráče použít. Nakonec kapitoly sedmá a osmá na základě dvou předchozích kapitol vybírají a odůvodňují výběr rozhodovacích technik pro jednotlivé herní části a věnují se jejich konkrétnějšímu rozboru.

Čtvrtá část je tvořena devátou kapitolou. Zabývá se implementací.

2 Základní principy hry

HINT spadá do kategorie tahové strategie odehrávající se hlavně na velké strategické mapě. Hráč zde vlastní několik hrdinů, neboli velitelů, kteří vedou svou armádu jednotek a obsahují také základní prvky her na hrdiny, pomocí kterých mohou ovlivňovat a podporovat jak svou armádu, tak také svoje vlastní dovednosti.

Každý hrdina má dvě primární dovednosti útok a obranu, které se přenášejí na účinnost armády v podobě aditivního bonusu na příslušné vlastnosti jednotek v hrdinově armádě a pak dvě další primární dovednosti moc a znalosti, které ovlivňují přímo hrdinovu schopnost v oblasti kouzlení. Pomocí kouzel může hrdina podporovat své jednotky v boji. Každé kouzlo má svou magickou cenu. A konečně má každý hrdina dovednost reprezentující, kolik má many, tedy magické energie, za které může vyvolávat kouzla.

Hrdiny je možné přesouvat na strategické mapě a zabírat tak především města a zdroje surovin, sbírat bonusové předměty či volně se na mapě nacházející suroviny, nebo navštěvovat rozličná bonusová místa. V tom jim brání jednak neutrální příšerky, jednak také hrdinové nepřátelských hráčů. Příšerky jsou neutrální armády, které se nemohou volně pohybovat, pouze stále nečinně stojí na mapě a to obvykle právě v těch místech, která jsou pro hráče z nějakého důvodu cenná, čímž tato místa chrání, protože na danou cennou lokaci nejde vstoupit, dokud hráč tuto příšerku neporazí. Hrdinové nepřátelských hráčů se pohybovat samozřejmě mohou. Příšerky a hrdinové jsou na strategické mapě reprezentovány vždy jako jediný objekt.

V případě konfliktu dvou hrdinů, nebo hrdiny s příšerkami se hra přepne do soubojové části, ve které proti sobě bojují jednotky těchto dvou armád. Právě zde jsou pak nejvíce využívány zmíněné hrdinovy dovednosti a kouzla. Za vítězný boj hrdina vždy získává zkušenosti, díky kterým při určitém množství postupuje na další úrovně a tím si své dovednosti dále vylepšuje.

Města jsou na strategické mapě také reprezentována jako jediný objekt a práce s nimi probíhá v samostatné části. Ve městech hráč může stavět různé typy

budov. Některé slouží k nakupování jednotek pro posilování hrdinových nebo hradních armád, jiné pro vylepšování hrdinových dovedností a další zase pro pravidelné zásobování surovinami. Některé budovy mohou jako předpoklad vyžadovat předchozí postavení jiné budovy. Všechny budovy mají svou cenu v surovinách. Proto je potřeba suroviny průběžně doplňovat buď sbíráním jejich volných výskytů na mapě, nebo zabíráním dolů. Každý herní den však lze postavit nejvýše jednu budovu pro každé město. Jednotky také mají svou cenu, jsou však navíc omezeny týdenním přírůstkem.

Herní čas se počítá na herní dny, které se skládají z herních kol všech hráčů, pro každého připadá jedno kolo na den v předem daném pořadí. Ve svém kole hráč může provádět výše zmíněné akce, jako je pohyb s armádami, rozvíjení měst, doplňování armád a podobně. Sedm herních dní tvoří herní týden. Na začátku každého týdne se zvyšují počty jednotek k nákupu ve městě o předem daný přírůstek.

Cílem hry je obvykle porazit všechny armády a hrady nepřátel.

3 Programy podobného zaměření

Vzhledem k tomu, že hra HINT je inspirovaná herní sérií Heroes of Might and Magic[2], nabízí se především srovnání s ní.

3.1 Heroes of Might and Magic

Heroes jsou úspěšná série tahových strategií, vydávaná mezi lety 1995 – 2007. Původně byla hra vytvořená firmou New World Computing, která však byla později odkoupena firmou 3DO. Nakonec byla práva roku 2003 prodána firmě Ubisoft. (1)

Jak už samotný název napovídá, nejedná se pouze o tahovou strategii, ale hra navíc obsahuje základní prvky her na hrdiny. Odehrává se ve fantasy prostředí Might and Magic, což je herní svět zavedený v další úspěšné herní sérii Might and Magic, opět vytvořené firmou New World Computing. (2)

Obrázek 3-1 ukazuje první díl celé série, Obrázek 3-2 zatím poslední pátý díl série. (1)

Obrázek 3-1 Heroes of Might and Magic: A Strategic Quest

[2] V textu už dále jen zkráceně Heroes, používané v množném čísle a originálním znění, český překlad by zněl Hrdinové Síly a Magie

Obrázek 3-2 Heroes of Might and Magic V

Následující podkapitoly popisují pouze základní přehled vlastností a odlišností mezi jednotlivými díly HINTem. Popisovány jsou však jenom základní verze všech dílů, případná další rozšíření některých z nich totiž už nepřinášejí podstatnější změny do pravidel.

3.1.1 Základní rozdíly v naší práci a Heroes

Heroes obsahují velmi podobná základní pravidla jako HINT, která však ještě v několika oblastech dále rozšiřují o doplňující prvky. Pokud bychom měli být zcela přesní, tak vzhledem k tomu, že pravidla HINTu vychází z pravidel Heroes, bylo by zřejmě korektnější označovat naše pravidla spíše jako zúžení pravidel Heroes, než mluvit o rozšíření v opačném směru, pro lepší čitelnost se přesto mluví o rozšíření.

Kampaně a příběh

Kromě rozšíření pravidel je podstatným rozdílem také přítomnost příběhu, s čímž je spojená herní kampaň, ve které hráč postupně prochází jednotlivé mise, plní případně zadané úkoly a tím se mu tento příběh krok za krokem odkrývá. Cílem pak tedy nemusí být zničit všechny nepřátele, ale například ubránit po zadaný čas některé město, nebo naopak obsadit nepřátelské měst, nashromáždit určitý počet jednotek, najít zvláštní artefakt a podobně. V naší práci příběh ani kampaně nejsou. Společným prvkem je však možnost hrát samostatné mapy, které jsou v Heroes označované jako scénáře. Tyto mapy většinou poskytují, nebo by alespoň měly poskytovat pro všechny hráče stejné šance a cílem je pak obvykle zničit všechny armády nepřátel a zabrat jejich

hrady, v čemž se liší od map z kampaní, které jsou spíše postaveny tak, aby vyhovovaly příběhu či zadanému úkolu. Není to však v Heroes pravidlem. Jednotlivé scénáře mohou mít podrobnější podmínku k vítězství podobnou podmínkám z kampaní a tedy nevyváženou mapu.

Příkrov neznáma

Jedním z těchto rozšiřujících prvků je přítomnost příkrovu[3], který představuje neprozkoumané oblasti a zahaluje tedy větší či menší části strategické mapy. Hráč proto nevidí, co se v těchto místech mapy pod příkrovem ve skutečnosti odehrává a pokud snad nezná mapu předem, například z předešlého hraní, neví dokonce ani to, jak tam herní svět vlastně vypadá. Je proto ve hráčově zájmu prozkoumávat tyto neprobádané oblasti, odstraňovat tak příkrov a získávat mnohem podrobnější znalosti o herním světě. Obvykle na začátku hry příkrov zahaluje téměř celou část mapy až na objekty v hráčově vlastnictví. V HINTu se příkrov nenachází a hráč tedy vždy vidí celou herní mapu.

Rasy

V naší práci se jednotky ani hrdinové nijak nedělí do ras. Heroes obsahují hned několik ras, které se liší typem hrdinů, zvláštními dovedností, případně také rozdílnými speciálními budovami.

S novějšími díly hry se zvyšují rozdíly jak mezi jednotlivými rasami, tak také mezi hrdiny jednotlivých ras. Například v prvním díle jsou čtyři rasy. Barbar, Rytíř, Kouzelnice a Černokněžník. První dvě rasy jsou zaměřené převážně na sílu, což znamená, že v souboji spoléhají hlavně na účinnost a nebezpečnost jednotek, naproti tomu dvě ostatní rasy kladou důraz hlavně na kouzlení. Mezi hrdiny jedné rasy už však v tomto díle nejsou žádné další rozdíly, kromě jejich portrétu. V dalších dílech se však rozdíly více prohlubují. (3) (4)

Primární dovednosti[4]

Každá hrdinova armáda, respektive její hrdina, má kromě pěti zmíněných primárních dovedností, které jsou jak ve všech dílech Heroes, tak v naší práci, také určitou hodnotu morálky a štěstí, které ovlivňují, jak velká je při souboji

[3] Anglicky Fog of War
[4] V anglickém originále Skills, což by se také mohlo překládat jako schopnost, avšak tento překlad je ponechán pro jiný herní prvek, Abilities

pravděpodobnost, že jednotka, která je právě na tahu dostane tah navíc, respektive způsobí při útoku dvojnásobné škody. (5)

Na morálku má vliv také složení armády, kde kombinace jednotek z některých rozdílných ras výrazně morálku snižuje, případně naopak kombinace jednotek z jedné rasy morálku zvyšuje.

Primární dovednosti Útok, Obrana, Moc a Znalosti se zvyšují vždy při postupu hrdiny na vyšší úroveň. Nezvyšují se však hned všechny, ale pokaždé pouze jedna z nich. Pravděpodobnost, se kterou se určitá dovednost zvýší, záleží na druhu hrdiny, tedy hlavně na tom, jestli je zaměřený spíše na sílu, nebo spíše na kouzlení. Hrdina pak obvykle dostane na výběr ze dvou dovedností na sílu nebo naopak ze dvou dovedností na kouzlení.

Primární dovednosti je možné zlepšovat také navštívením speciálních lokací na mapě. V případě útoku, obrany, moci a znalostí se jedná o trvalé zvýšení, v případě many, morálky a štěstí pouze o dočasné, dokud se morálka či štěstí nezmění kvůli složení jednotek, případně mana se nevyčerpá.

Druhotné dovednosti

Počínaje druhým dílem mají hrdinové ještě také druhotné dovednosti, které přestavují specializovanější zaměření na určitou oblast ať už v rámci armády, hrdiny, nebo dokonce celého království. Například pro souboje lepší útok pro střelce, větší štěstí nebo morálka, vylepšení kouzlení, pro strategickou mapu větší pohyblivost hrdiny, pro celé království bonusy k hospodaření a další. (6)

Druhotné dovednosti mají vždy více úrovní a platí, že čím vyšší úroveň hrdina ovládá, tím silnější je daná dovednost. Druhotné dovednosti se hrdina může učit a také vylepšovat vždy při přechodu na další úroveň a má přitom vždy na výběr mezi dvěma a více dovednostmi[5]. Počet dovedností ve výběru se liší v jednotlivých dílech. Hrdina také může navštěvovat speciální lokace na mapě, kde se dají některé druhotné schopnosti naučit. Počet druhotných dovedností, které se hrdina může naučit, je však omezený. Hrdina si tedy může průběžně při učení vybírat pouze určitou podmnožinu. Pravděpodobnost, s jakou se

[5] Počet dovedností ve výběru se liší v jednotlivých dílech. Neplatí v případě, že již nemá pro další dovednosti místo, v tom případě už nemusí na výběr dostat žádnou dovednost, případně pouze jednu a podobně.

hrdinovi určitá dovednost při postupu na další úroveň nabídne, silně závisí právě na druhu daného hrdiny. Například hrdinové zaměření na sílu, jako třeba Barbar nebo Rytíř, dostanou při výběru s mnohem vyšší pravděpodobností dovednost opět zaměřenou na sílu, než dovednost zaměřenou na kouzlení.

Ve druhém a třetím díle mají dovednosti tři úrovně, jsou to základní, pokročilá a expertní. Ve čtvrtém díle se přidává úroveň mistrovská a velmistrovská.

V pátém díle je systém druhotných dovedností ještě mnohem komplexnější. Kromě dovedností jsou zde navíc ještě schopnosti[6], které se vážou vždy na některou z dovednosti. Pokud se tedy hrdina naučí určitou dovednost, dostane při učení k dispozici několik schopností, které s danou dovedností souvisí. Opět jsou zde úrovně dovedností, většinou tři až čtyři. Schopnosti pak na této úrovni také závisí, protože hrdina se nemůže určitou schopnost naučit, dokud nemá příslušnou schopnost na dostatečné úrovni.

Dobývání hradu

Při útoku na město neprobíhá souboj jako při útoku na armádu nebo příšerky. V tomto případě může mít hráč bránící město obranné hradby, za kterými jsou jeho jednotky lépe chráněné. Hradby mají vstupní bránu a při dalším vylepšení také obranné věže, které při svém tahu střílí na protivníka. Útočící hráč má zase k dispozici katapult, který ve svém tahu střílí na hradby, věže či bránu a snaží se tak o vytvoření průniku útočících jednotek do města.

Čím novější díl, tím opět větší účast hráče při vývoji opevnění. Například v prvním díle mohl hráč pouze postavit hrad a tím rovnou získat hradby, bránu a věže, v druhém díle už mohl stavět přímo podle vlastního výběru a pořadí jednotlivé věže, příkop, nebo u některých ras silnější hradby. V dalších dílech hráč také buduje opevnění postupně, ale už v pevně daném pořadí v rámci celkového vylepšení hradu.

S tím také souvisí bojové stroje. V prvních dvou dílech má armáda k dispozici pouze katapult na ničení opevnění, ve třetím díle se přidává ještě balista, stan první pomoci a vozík s municí. Na rozdíl od předchozích dvou dílů se zde také bojové stroje dají ostatními jednotkami zničit. Ve čtvrtém díle opět zůstává

[6] V anglickém originále Abilities, jako rozlišení oproti dovednostem, tedy Skills

pouze katapult. Balista zde sice zůstává, ale nemá zvláštní postavení bojového stroje, je pouze další obyčejnou jednotkou jedné z ras. Pátý díl se v oblasti bojových strojů vrací k provedení ze třetího dílu.

Hrdina a boj

Ne ve všech dílech má hrdina stejnou účast v boji. V prvním, druhém a třetím díle je situace stejná. Hrdina je nezranitelný, může pouze kouzlit a to nejvýše jednou za každé soubojové kolo, vždy při tahu jednotky své armády. Pokud tedy v nějakém soubojovém kole nekouzlí, akce mu propadá. Ve čtvrtém díle se situace výrazně změnila. Hrdina se zde plně účastní boje, je tedy zranitelný a může kromě kouzlení ještě dělat stejné akce jako ostatní jednotky v armádě. To vše však pouze když na něj přijde v souboji řada, nemůže tedy hrát během tahu jiné jednotky ve své armádě. V pátém díle se vytváří kompromis mezi oběma přístupy. Hrdina může kouzlit a je nezranitelný jako v prvních třech dílech, může však také útočit a hraje pouze, když na něj přijde řada, jako ve čtvrtém díle. Nemůže se však pohybovat.

V HINTu hrdinova účast v boji odpovídá nejvíce pátému dílu Heroes.

Pokročilé stupně jednotek

Kromě čtvrtého dílu je možné většinu jednotek vylepšovat na druhou úroveň, ve druhém díle lze jednu jednotku vylepšit dokonce na třetí úroveň. Vylepšená jednotka má obvykle mírně zvýšené základní vlastnosti, jako například lepší pohyblivost nebo útok a občas získává další zvláštní schopnosti jako například dvojitý útok nebo neomezený počet odvetných útoků. Také se mírně změní její vzhled.

Ve čtvrtém díle tento systém neplatí. Místo toho má hráč pro většinu jednotek každé rasy vždy na výběr mezi dvěma jednotkami stejné úrovně, to znamená přibližně stejné síly. Omezení platí vždy na jedno konkrétní město. Pokud si vybere, že bude ve městě nakupovat jednotky prvního druhu, zablokuje si tím hned v tomto městě nákup jednotky druhého druhu.

Systém magie

Základ systému magie je společný všem dílům. Hrdina se kouzla učí převážně navštívením města, ve kterém je cech mágů, případně v menším množství ve zvláštních lokacích na mapě opět jejich navštívením. Cech mágů, který může hráč ve městě vybudovat má více úrovní a v některých dílech je jeho nejvyšší povolená úroveň omezená rasovým typem města. Platí, že čím vyšší úroveň cechu, tím mocnější kouzla poskytuje. Za kouzla se již dále neplatí, jejich naučení probíhá automaticky.

Ve třetím díle se k tomu ještě kouzla dělí do dalších kategorií podle přírodních živlů, tedy do magie vody, ohně, země a větru. Některá kouzla jsou společná všem kategoriím, ale většina kouzel se liší a odráží v sobě charakteristiky této kategorie, například kouzla kategorie ohně jsou převážně útočná a podobně. Navíc je s každou kategorií kouzel spojená hrdinova schopnost ovládání dané magie, kdy s rostoucí úrovní schopnosti hrdina sesílá kouzlo v silnější podobě. Podobný přístup je také ve čtvrtém díle, kde se však kategorie neřídí živly, ale přímo jednotlivými rasami. V pátém díle systém částečně přibližuje třetímu dílu, kategorie se zde dělí na destruktivní magii, světlou magii, temnou magii a magii přivolávaní.

V HINTu se hráč také učí kouzla ve městě a to automaticky jeho navštívením armádou, jejíž velitel má dostatečně vysoké dovednosti moudrost a znalosti. Požadovaná úroveň těchto dovedností by pak měla odrážet sílu kouzla.

Hrdinové

Ve všech dílech Heroes je několik druhů hrdinů. Každá rasa má vlastní hrdiny a každý hrdina má specifické vlastnosti, které se mezi jednotlivými hrdiny v pozdějších dílech čím dál víc prohlubují. Hrdinové se nakupují ve městě. Hráč nemusí nakupovat pouze hrdiny rasy, za kterou hraje, je však mnohem pravděpodobnější, že je ve městě dostane na výběr.

V HINTu jsou tři druhy hrdinů. Nemají však druhotné dovednosti, ale liší se stejně jako v Heroes ve svých primárních dovednostech a v rychlosti růstu těchto dovedností.

3.1.2 Shrnutí rozdílů

Tabulka 3.1 ukazuje shrnutí rozdílů v základních pravidlech mezi HINTem a jednotlivými díly Heroes.

Tabulka 3. 1 - Shrnutí rozdílů pravidel Heroes a naší práce

	Tato práce	Heroes 1	Heroes 2	Heroes 3	Heroes 4	Heroes 5
Hrdina v souboji, kouzlení	Jednou za kolo ve svém tahu	Jednou za kolo během tahu své jednotky	Jednou za kolo během tahu své jednotky	Jednou za kolo během tahu své jednotky	Jednou za kolo ve svém tahu	Jednou za kolo během tahu své jednotky
Hrdina v souboji, zranitelnost	Nezranitelný	Nezranitelný	Nezranitelný	Nezranitelný	Zranitelný	Nezranitelný
Stupně jednotek	Žádné	Obvykle dva	Obvykle dva, případně tři	Obvykle dva	Žádné, výběr mezi dvěma jednotkami	Obvykle dva
Kouzla, kategorie	Bez kategorií	Bez kategorií	Bez kategorií	Kategorie dělená podle přírodních živlů	Kategorie dělená podle rasy	Kategorie dělená podle účinků kouzla
Kouzla, učení	Automaticky ve městech	Automaticky ve městech s cechem a ve zvláštních lokacích	Automaticky ve městech s cechem a ve zvláštních lokacích	Automaticky ve městech s cechem a ve zvláštních lokacích	Učení automaticky ve městech s cechem a ve zvláštních lokacích	Učení automaticky ve městech s cechem a ve zvláštních lokacích
Primární dovednosti	Útok, Obrana, Moc, Znalosti, Mana	Útok, Obrana, Moc, Znalosti, Mana, Morálka, Štěstí	Útok, Obrana, Moc, Znalosti, Mana, Morálka, Štěstí	Útok, Obrana, Moc, Znalosti, Mana, Morálka, Štěstí	Útok, Obrana, Moc, Znalosti, Mana, Morálka, Štěstí	Útok, Obrana, Moc, Znalosti, Mana, Morálka, Štěstí
Druhotné dovednosti	Ne	Ne	Ano, více stupňů	Ano, více stupňů	Ano, více stupňů	Ano, více stupňů + schopnosti
Vylepšování dovedností	Postupem na další úroveň	Postupem na další úroveň, speciální lokace	Postupem na další úroveň, speciální lokace	Postupem na další úroveň, speciální lokace	Postupem na další úroveň, speciální lokace	Postupem na další úroveň, speciální lokace
Obrana města	Neliší se od klasického boje	Věž, hradby, brána	Věž, hradby, brána, silnější opevnění, příkop	Věž, hradby, brána, příkop	Věž, hradby, brána, příkop	Věž, hradby, brána, příkop
Budování obrany města	Ne	Staví vše jednorázově najednou	Staví dle vlastního pořadí	Vylepšuje hrad, tím staví v pevném pořadí	Vylepšuje hrad, tím staví v pevném pořadí	Vylepšuje hrad, tím staví v pevném pořadí
Rasy	Ne	4	6	8	6	6

Odvetné útoky	Ano	Ano	Ano	Ano	Ne, odveta probíhá současně	Ano
Bojové stroje	Ne	Katapult, nezničitelný	Katapult, nezničitelný	Katapult, další bojové stroje, vše zničitelné	Katapult, nezničitelný	Katapult, další bojové stroje, vše zničitelné
Dobývání hradu	Ne	Ano	Ano	Ano	Ano	Ano
Nákup jednotek	Ve městě	Ve městě a speciálních lokacích	Ve městě a speciálních lokacích	Ve městě a speciálních lokacích	Ve městě a speciálních lokacích	Ve městě a speciálních lokacích
Nákup jednotek, omezení	Ano, týdenní přírustek	Ano, týdenní přírustek	Ano, týdenní přírustek	Ano, týdenní přírustek	Ano, týdenní přírustek	Ano, týdenní přírustek
Příšerky na mapě	Ano, pouze hlídají	Ano, pouze hlídají	Ano, pouze hlídají	Ano, pouze hlídají	Ano, hlídají a útočí na nejbližší okolí	Ano, pouze hlídají

3.2 Free Heroes 2 Engine

Free Heroes2 Engine (7) je volně šiřitelná implementace Heroes of Might and Magic 2. Na rozdíl od HINTu se však snaží o přesnou podobnost s původní hrou a to jak v oblasti pravidel, tak v grafickém pojetí.

3.3 Další podobný software

V některých ohledech bychom jistě našli určitou podobnost také například s tahovou strategií Battle For Wesnoth (8), dále potom s rozsáhlou herní sérií Civilizace (9), případně z dalších tahových strategií například s hrou Fantasy General (10). Je to však již podobnost velmi vzdálená a nebude už proto blíže popisována.

4 Analýza síťové komunikace

Hra je primárně určena pro hraní po lokální síti, případně hře proti počítači. Často je však mnohem pohodlnější, či dokonce nezbytné, aby bylo možné také hrát ve více hráčích na jednom fyzickém počítači, protože více klientských počítačů nemusí mít hráči k dispozici. Nebo případně přístupy kombinovat, tedy zvládat situace, kdy někteří hráči hrají ve skutečnosti z jediného počítače, další hráči z jiného počítače a do toho je třeba ve hře zapojen také počítačový hráč.

4.1 Síťová architektura a komunikace

Z povahy aplikace je nejvhodnější použít síťovou architekturu klient-server. Server zde potom reprezentuje celkovou hru a provádí herní výpočty, zatímco hráči s ním jako jednotliví klienti komunikují a vznášejí tím své požadavky na provedení herních akcí.

Pro samotnou komunikaci je pak samozřejmě možné zvolit z více technologií. Nabízí se například přímá práce se sockety nebo naproti tomu volání metod vzdálených objektů, tak zvaný remoting. Sockety nabízí větší sílu a výkonnost, na druhou stranu postrádají transparentnost, jednu z velkých výhod remotingu. Protože naše práce implementuje tahovou strategii a komunikace serveru s klientem proto určitě nebude příliš intenzivní, převládá zde spíše výhoda transparentnosti a jednoduchosti remotingu, než požadavku na co nejvyšší výkon.

4.2 Oddělení herní logiky od prezentace

Jedním z užitečných přístupů v programování pro čistější a znovupoužitelný kód je oddělení grafického rozhraní od aplikační logiky, tedy prezentace od datového modelu.

Vzhledem k tomu, že HINT představuje síťovou aplikaci, je zde vhodné již od začátku klást důraz na využívání tohoto přístupu. Server v tomto případě obsahuje veškerou herní logiku, potřebnou pro herní výpočty, zatímco všechna práce zabývající se prezentací je ponechána na klientovi. Jinak řečeno, server

na základě herních požadavků pouze určuje, jaké herní akce se mají vykonat, jak se mají vykonat a jak se tím změnil herní stav a podle toho následně posílá klientovi odpovídající logická data, avšak nechává už jenom na něm, jak bude tuto informaci prezentovat uživateli.

Klient však přesto může některé výpočty provádět, například ověřit, zda je některá akce v dané situaci možná a podle toho zasílat požadavky na server. Například by měl rozpoznat, že cílové souřadnice při posunu jednotky na mapě odpovídají stromu či skále a vůbec negenerovat pro server požadavek pro pohyb. Naproti tomu už by měl na serveru ponechat nalezení cesty, nebo výpočet následků a podobně.

4.3 Vzdálené objekty na serveru

Server musí zajišťovat nejen herní akce, ale také operace jako připojování či odpojování klientů, zakládání her, registrace hráčů do založené hry a podobně. Proto by bylo vhodné oddělit metody spojené přímo s hrou a metody spojené spíše se síťovou komunikací. Server by tedy nenabízel klientům pouze jediný vzdálený objekt, ale minimálně dva, jeden pro zpracování herních akcí a další pro obsluhu komunikace.

Pokud však budeme uvažovat hru samotnou, z hlediska herní logiky a ne jako síťový server, tak zjistíme, že hra potřebuje komunikovat především s hráči, ne s klienty. Proto by bylo lepší, kdyby server přijímal požadavky od hráčů a také hráčům odpovídal. Neměl by se už příliš zabývat skutečností, že hráč je klient na vzdáleném počítači. Jedna třída by tedy reprezentovala hru a další třídy pak jednotlivé hráče, společně se všemi pro hráče relevantními daty. Až tato třída reprezentující hráče by skutečně komunikovala s klientským počítačem, od kterého by přijímala herní požadavky a následně je po ověření hře přeposílala. Vytvořením tohoto oddělení hráče od klienta budeme také moci jednodušeji a více intuitivně přidat počítačového hráče, který bude opět reprezentován touto hráčskou třídou a nebude tak součástí kódu herní logiky. Hra vnímá hráče, ne síťový objekt, přijímá od něj požadavky a zasílá mu odpovědi. Jestli je hráč nakonec lidský hráč umístěný na vzdáleném počítači, nebo je to počítačem řízený hráč, není v tomto směru pro hru zásadně důležité a komunikuje s nimi přes stejné rozhraní.

Ve výsledku by tedy byl jeden vzdálený objekt, který by zajišťoval počáteční síťovou komunikaci, jako připojení klienta k serveru a byl by společný všem klientům. Dále pro každého logického hráče jeho reprezentace na serveru, používaná také klienty jako vzdálený objekt pro zadávání herních požadavků.

4.4 Rozložení dat

V našem rozdělení jsou tedy veškerá herní data, potřebná k herním výpočtům, uložena na herním serveru, protože kromě hry samotné je také reprezentace hráčů uložena na serveru. Klienti žádné výpočty provádět nemusí a ani by to nebylo vhodné, protože by pak mohli mít prostor pro chybu či podvádění. Proto žádná logická data neukládají, pouze vzdáleně volají metody hráče na serveru.

Klienti si však u sebe uchovávají částečné kopie těchto dat, aby se předcházelo neustálému zasílání opakovaných dat. Server vždy klientovi zašle jen ta data, která jsou pro klienta nová, nebo změněná. To však může znamenat i zaslání části dat, která klient zná, ale souvisí s daty zasílanými.

Například pokud klient již zná informace o vlastnostech určité jednotky, ale vlivem nějaké akce, třeba útoku nebo kouzla došlo ke změně některých jejích vlastností, zašle se klientovi nová kompletní informace o vlastnostech této jednotky, nikoliv pouze o konkrétní změněné vlastnosti.

Pokud by se zasílala vždy jen konkrétní vlastnost, tak by se sice ušetřila přenosová kapacita, ale na druhou stranu by bylo třeba zavést mechanismus takového posílání pro jednotlivé vlastnosti, což je však poměrně těžkopádné vzhledem k tomu, že se jedná o komunikaci přes lokální síť, kde je kapacita dostatečná, navíc v případě tahové strategie se nezasílají data až příliš často.

Klient však nevyužívá tato data primárně pro výpočty, ale pro správné zobrazení uživateli. Potom musí mít zobrazované informace někde uložené, není možné, aby například při vykreslování jednotky a hodnoty jejího zdraví ve formě čísla při každém vykreslení prováděl dotaz na server ohledně toho, jaká je hodnota tohoto zdraví a podobně.

4.5 Zprávy a volání vzdálených metod

Ve vybraném přístupu bude tedy server nabízet klientům vzdálený objekt, přes který budou moci provádět herní příkazy, registrovat se do hry a podobně, vše transparentní formou voláním metod tohoto vzdáleného objektu.

V obráceném směru, tedy při směru komunikace od serveru ke klientovi, by byla opět možnost použít volání vzdálených metod. Server by si tedy stejně jako klient v předchozím případě nejdříve zaregistroval vzdálený objekt, který by byl umístěný u klienta a voláním jeho metod by opět prováděl komunikaci směrem ke klientovi. Pro každou odpověď od serveru by u klienta existovala vzdálená metoda, která by tuto odpověď zpracovávala.

Jinou možností by se mohlo zdát prosté použití návratových hodnot při volání vzdálených metod klientem. Zde však nastává problém při hře více hráčů, kdy je také více klientů. Pokud by totiž klient provedl pomocí vzdálené metody na serveru nějakou akci, jejíž následky by se měly vrátit v návratové hodnotě této metody, tak by se výsledek této akce dozvěděl pouze aktivní volající klient a ne další hráči. Mnoho herních akcí má však jistě vliv na více hráčů a musí tedy být možnost, jak o změně herního stavu ostatní hráče informovat, což však prosté navracení hodnot nenabízí. Použití návratových hodnot však najde své uplatnění v rámci akcí, které samotný herní stav nijak nemění, ale pouze například žádají o ověření, zda je určitá akce vůbec v dané situaci možná. Také se hodí jako potvrzování úspěchu či neúspěchu při provádění herní akce.

Další možné řešení je použít zasílání zpráv, které lépe odráží povahu samotné odpovědi. Serverová odpověď klientovi je totiž určitá zpráva, která popisuje potřebné změny v herním stavu po provedení klientem žádané akce a koncept zasílání zpráv tedy tuto situaci nejlépe reprezentuje. Navíc je tento koncept intuitivnější v rámci asynchronní komunikace, jak je popsáno v dalších podkapitolách.

4.6 Synchronní a asynchronní komunikace

Ve zvolené architektuře klient-server máme při komunikaci možnost použití dvou různých druhů přenosu, synchronního a asynchronního. Zvolený způsob pro jednotlivé směry komunikace je však dán vlastnostmi aplikace.

4.6.1 Komunikace směrem klient-server

Pro spojení od klienta k serveru není v zásadě potřeba asynchronní přenos, protože se jedná o tahovou hru a tak se předpokládá, že všichni klienti budou komunikovat s herním serverem pouze v případech, kdy bude klientský hráč na tahu. To je dáno právě povahou aplikace, protože pokud hráč na tahu není, neměl by ani mít důvod se na server dotazovat. Pokud by přesto nějaký klient potřeboval se serverem komunikovat, musel by jeho požadavek počkat, až server vykoná předchozí požadavky a bude na něj mít čas.

4.6.2 Komunikace směrem server-klient

Naproti tomu při komunikaci v opačném směru, tedy od serveru ke klientovi, se již hodí asynchronní přenos, protože na určitou herní akci je většinou nutné zaslat potřebné informace o změně stavu hry více klientům. Zpracování této informace na straně klienta však už může být časově náročnější, především v případech, kdy klient také graficky znázorňuje tuto informaci uživateli, tedy překresluje grafickou prezentaci. Proto je lepší zasílat informace jednotlivým hráčům asynchronně, nezávisle na ostatních. Díky tomu, zatímco jeden klient už přijatá data dále zpracovává, tak už se současně zasílají data ostatním klientům a klient tak nikoho nezdržuje.

Jinou možností by bylo zasílat informace synchronně s tím, že klient informaci přijme, ale nezačne ji hned zpracovávat. Místo toho si ji uloží a teprve na svém dalším vlastním vlákně tato data dále zpracuje. Server by tak hned mohl pokračovat v zasílání informací dalším klientům. Tento přístup má však tu nevýhodu, že server v tomto případě musí spoléhat na klienta v tom, že nebude server zdržovat zpracováním dat a opravdu je zpracuje až na vlastním vlákně. Server pak nemůže zaručit, že ho některý klient nebude zdržovat. Proto je vybrán radši první způsob a přesun odpovědnosti přímo na server.

4.7 Řazení zpráv

Protože jsme pro zasílání zpráv zvolili asynchronní přenos, je nutné, aby každá zpráva měla kromě vlastních dat také své jedinečné pořadové číslo, které slouží k jejímu správnému zatřídění po jejím doručení, protože zprávy nemusí kvůli asynchronní komunikaci dorazit v pořadí odeslání. Často sice dokonce

při asynchronním přenosu dojdou zprávy skutečně ve stejném pořadí, v jakém byly původně odeslány, avšak rozhodně se na to nelze spoléhat a tato špatná zpráva, která by mohla dorazit před jinou, před kterou však logicky nepatří, by mohla způsobit nemalé komplikace při zpracování.

4.8 Adresování zpráv

Často je po určité herní akci potřeba zaslat několika klientům stejnou zprávu. Proto by bylo nejintuitivnější, kdyby server vždy vytvořil pouze jednu kopii dané zprávy a až pak v rámci skutečného odesílání ke klientovi, kdy se tato zpráva musí před odesláním serializovat, by se tím také zároveň provedla už automaticky skutečná replikace zprávy pro každého klienta.

Pokud však vytvoříme vždy jedinou zprávu a necháme až na serializaci, aby vytvořila její replikaci, nebude možné uvést ve zprávě adresáta zprávy. Jedná se totiž ve skutečnosti pořád o jeden a ten samý objekt sdílený před odesláním více hráči. Pokud bychom chtěli adresáta přidat, museli bychom sami vytvářet kopie stejné zprávy a před serializací přidat do zprávy tuto adresu.

Ve skutečnosti však adresát ani není potřeba. Server sám nejprve určí, komu je potřeba zprávu zaslat a následně pošle tuto zprávu všem vybraným hráčům, avšak nikomu dalšímu. Klient pak nemusí vůbec kontrolovat, zda se jedná o zprávu určenou pro něj, protože zprávy, které by pro něj určené nebyly, vůbec nedostane.

Na druhou stranu však na straně klienta může hrát více hráčů na jednom počítači, kteří jsou na server připojení jako jeden klient. Pak je potřeba u klienta rozpoznat, kterému z nich zpráva opravdu patří, protože ne oba hráči musí dostávat vždy stejné zprávy. Avšak po odeslání zprávy klientovi už klient nemůže skutečného adresáta nijak rozpoznat, pokud číslo adresáta není součástí zprávy, což ale nechceme, protože některé zprávy jsou určeny pro více hráčů a tak ani nemohou být přímo adresovány jednomu hráči. Proto je vložena mezi server a odeslání ještě další vrstva, která před odesláním zprávy vytvoří „obálku“, do které zabalí zprávu společně s číslem hráče, kterému patří. Zpráva se tedy opět skutečně replikuje až při serializaci, pro každého hráče se pouze zvlášť vytváří tato jednoduchá „obálka“.

5 Analýza AI: Problémy, které je třeba řešit

Před podrobnější analýzou a rozhodnutím, který konkrétní přístup na řešení umělé inteligence zvolit, je potřeba si blíže přiblížit a popsat problémy, které má počítačový hráč vlastně řešit.

Základní problémy, kterými se počítačový hráč musí zabývat, jsou podobné problémům, které řeší také lidský hráč. Ve strategické části se jedná především o stavění budov, obsazování zdrojů surovin, sbírání odměn, obrana vlastních měst, rekrutování hrdinů a jednotek a útoky na nepřátelské armády a města. V soubojové části potom o vedení boje s protihráčem. Zásadní však je, do jaké míry by měl počítačový hráč tyto problémy řešit a na co by měl klást důraz.

5.1 Upřesněné terminologie

V následujícím textu používáme termín **„hráč“** jak pro lidského hráče, tak pro počítačového hráče. Neznamená to tedy výhradně lidský hráč, který hru opravdu hraje, ale spíše hráč z pohledu herní logiky, který může být řízen jak člověkem, tak počítačem. V žádném případě se už tedy v této části nejedná o síťového klienta.

Někdy však také potřebujeme mluvit o hráči jako o „člověku, který tohoto logického hráče ovládá“. Zda se jedná o logického hráče nebo o člověka, který hráče ovladá, by však mělo být vždy patrné z kontextu.

5.2 Neomylnost vs. hratelnost

V první řadě není až tolik důležité, aby AI hrála až tak dobře, že by hráče vždy porážela, protože AI by měla nabízet především zábavu a hratelnost, což také znamená nehrát proti hráči až příliš chytře, na druhou stranu však nesmí hrát hloupě, protože taková AI pak není hráčovi důstojným protivníkem a mnoho zábavnosti tedy do hry nepřinese.

Hlavní důraz by měl být kladen na strategickou část, spíše než na soubojovou. Neznamená to, že by v soubojové části neměla AI co ztratit, avšak přeci jenom už díky odvetným útokům jednotek by měla při použití základních heuristik vycházejících z povahy soubojových pravidel hrát poměrně rozumně.

Na strategické mapě už má AI mnohem více možností něco pokazit a musí tedy být mnohem více opatrná. To platí především v těch situacích, které hráč jednoduše zaregistruje, protože působí vyloženě špatně. Například pokud má hráč v jednom dni na výběr mezi sebráním odměny a obsazením hned vedle umístěného zdroje surovin[7], ale opravdu jen jednoho z toho, protože na další už by mu v tu chvíli nezbyla pro příslušnou armádu pohyblivost, nevypadá jistě jako velký prohřešek, pokud zvolí odměnu, ačkoliv dobrý hráč by zřejmě ve většině situací upřednostnil právě zdroj surovin před odměnou a tím oproti odměně získal jeden zásobovací den odpovídající suroviny navíc. Na druhou stranu například bezhlavý útok na silnější nepřátelskou armádu či město, nebo dále nezájem o obranu svých měst při hrozbě jejich napadení nepřítelem, jsou mnohem závažnější prohřešky, kde špatné rozhodování obvykle počítačového hráče dovede k rychlé prohře.

5.3 Základní priorita problémů

Prioritu řešení těchto jednotlivých herních problémů napovídá cíl hry. Tím je získat všechna města na mapě a eliminovat všechny nepřátelské armády. Jak už z tohoto výherního cíle plyne, měla by být pro hráče nejdůležitější právě obrana svých měst, zabírání slabších nepřátelských měst a eliminace slabších nepřátelských armád. Zde je však podstatná zmíněná podmínka, že cíl útoku je slabší, protože útok na silnějšího protivníka ničemu nepomůže. Protivník však samozřejmě už od začátku slabší nebude, proto je nutné posilovat své armády. Posilování armád znamená hlavně doplňování jednotek, k čemuž je potřeba postavit ve městě příslušné budovy a následně jednotky koupit, což znamená mít k tomu dostatečný počet příslušných surovin. Z toho tedy hned plyne další důležitý cíl, kterým je sbírání odměn a zabírání dolů.

5.4 Herní předpoklady

Aby se usnadnil okruh problémů, které má AI na starosti, je dobré vymezit na hru určité předpoklady. Jedná se však pouze o takové předpoklady, které hru nijak neomezují, naopak spíše vycházejí z požadavku na celkovou hratelnost.

[7] Pro připomenutí, odměna po sebrání armádou dodá jednorázově bonus k počtu hráčových surovin, zdroj surovin po zabrání dodává každodenní bonus k počtu surovin. Bonus v odměně je sice vyšší než bonus ve zdroji surovin, ale je pouze jednorázový.

Jde v podstatě pouze o to, nezavádět do hry takové principy, které lidský hráč okamžitě rozpozná jako slepé uličky, které mu rozhodně nijak nepomůžou, takže se jim stejně radši vyhne. Avšak AI by pak při jejich existenci musela uvažovat další faktory a být tedy ve svém rozhodování mnohem opatrnější.

Předpoklady jsou o to podstatnější, že velká část vlastností herních objektů je v naší práci definovaná v externím XML souboru a proto by teoreticky mohly být definice objektů poměrně nevyvážené. Proto je dobré předpokládat, že ten, kdo s tímto souborem pracuje, v podstatě ví, co dělá a neklade proto úmyslně do hry slepé uličky a překážky. V praxi by samozřejmě tento soubor upravoval pouze herní specialista, v jehož zájmu jistě vyváženost a hratelnost je.

5.4.1 Hodnota surovin

Jedním z předpokladů je hodnota surovin vzhledem k určitému nákupu.

„Předpokladem je, že pokud něco stojí hodně peněz, musí to být velmi užitečné a stejně tak pokud to stojí jen malé množství peněz, pravděpodobně to přináší malý užitek“. (11 str. 325)

Znamená to, že hodnota v surovinách bude přibližně odpovídat užitečnosti kupované věci. Vyjádřit takovou hodnotu přesně by bylo značně náročné a vyžadovalo by důkladné testování hratelnosti na základně stanových cen, ve skutečnosti však není vůbec nutné, aby ceny byly zcela přesné, ale aby zhruba odpovídaly a nebyly vyloženě předražené.

5.4.2 Příšerky a chráněné suroviny

Další předpoklad se týká příšerek na mapě a odměn či zdrojů surovin, kde je dobré, aby suroviny, pokud jsou příšerkami chráněné, byly opět chráněné tak silnou příšerkou, aby náročnost její eliminace odpovídala užitečnosti následně získané odměny nebo zdroje surovin. Opět platí, že pokud by malá odměna byla chráněna velice přehnaně silnou příšerkou, nemělo by pro hráče ani smysl s touto příšerkou o surovinu bojovat. Tento předpoklad není tolik důležitý jako předchozí, opět však usnadňuje rozhodování.

5.5 Konkrétnější priorita problémů

Nyní jsou tedy dané hlavní problémy, kterými se musí AI ve strategické části zabývat, společně s jejich základními prioritami. Další problém je však přímo mezi jednotlivými kategoriemi problémů, jako na který hrad armádou zaútočit, na kterou armádu, kterou odměnu sebrat, který zdroj surovin zabrat, kterou budovu postavit a podobně.

5.5.1 Stavba budov

Budovy jsou definovány externě, ne přímo v kódu a je proto nutné podle jejich definice stanovit, která budova je užitečnější než jiná a jaká je proto priorita jejich postavení. Jednotlivé budovy mohou mít jiné budovy jako předpoklady, znamená to tedy určit strom stavění společně s prioritami jednotlivých budov.

Priorita budov závisí na jejich typu. U budov produkujících jednotky platí, že čím silnější jednotku budova poskytuje, tím je lepší. U budov přidávajících bonus k primárním dovednostem záleží na velikosti bonusu, který hrdinovi přidají a konečně u budov, které dodávají suroviny je situace podobná, jako u zabírání dolů, tedy čím je určitá surovina potřebnější, tím více se vyplatí koupit danou zásobovací budovu.

Užitečné by bylo také prioritu propagovat do závislostí, tedy pokud je nějaké budova hodně důležitá a má určité předpoklady, pak tyto předpoklady jsou tím pádem také o něco důležitější, když jsou nutné pro tuto důležitou budovu.

5.5.2 Odměny a zdroje surovin

Co se týče zdrojů surovin a odměn, AI by měla vycházet z toho, co je potřeba postavit a nakoupit. Každá budova má svou cenu a také prioritu, jak moc je potřeba tuto budovu postavit. Z toho plyne nutnost získávání surovin na její stavbu. Čím větší prioritu budova má, tím větší prioritu by mělo mít získání příslušných surovin a podle toho tedy také sbírání odměn a zabírání dolů. Jen tato priorita však sama o sobě nestačí. AI by také měla brát v úvahu, kolik už určité suroviny vlastní. Pokud tedy potřebuje nějakou surovinu na postavení budovy, je tato surovina ještě navíc tím potřebnější, čím méně jí hráč má. Pokud by například AI opravdu nutně potřebovala zlato, bylo by lepší jít do místa, kde je možné zlato získat, než do místa, kde je velké množství dřeva,

které však není ke stavbě zrovna tolik potřeba, případně ho má hráč dostatečné množství.

Zajímavým, avšak ne už tolik kritickým a nutným vylepšením by ještě bylo, pokud by AI zároveň dokázala preferovat taková místa se surovinami, kde je jich co nejvíce, samozřejmě ale stále s ohledem na aktuální potřeby.

Také by bylo užitečné, kdyby AI dokázala určit nebezpečnost každého políčka na cestě k odměně nebo zdroji surovin a podle toho zvážit, zda tam vůbec jít. Například je poměrně nerozumné jít pro poklad na místo, kde hrozí útok od silnější protivníkovy armády.

5.5.3 Útoky na město

Při útoku na města by měla AI preferovat taková města, která jsou co nejméně chráněná a co nejvíce užitečná. Užitečnost města je daná hlavně postavenými budovami, protože postavené budovy jednak ušetří peníze za jejich stavění, navíc přispívají buď surovinami, nebo bonusy nebo dostupnými jednotkami. Užitečnost postavených budov je pak dána stejnými principy, jako stavba vlastních budov, protože zabráním města s těmito budovami hráč budovy získá, postaví. Také by bylo užitečné uvažovat jednotky, které se dají ve městě nakoupit a tím posílit hráčovu armádu. Na druhou stranu by ale tyto jednotky mohl při obraně dokoupit nepřítel, bylo by proto dobré uvažovat jak jejich užitečnost pro hráče, tak také jejich možnou nebezpečnost.

Stejně jako v případě surovin by se AI měla v ideálním případě raději vyhnout nebezpečným místům.

5.5.4 Obrana města

Obrana města by měla být tím podstatnější, čím užitečnější dané město je, kde užitečnost je opět stejná jako při zvažování útoku na nepřátelské město.

5.6 Potřeba koordinace

Potřebujeme mít strategický rozhodovací mechanismus, který by byl schopen koordinovat více objektů najednou, nikoliv aby rozhodoval pouze individuálně pro jednotlivé objekty. Hráč by měl totiž přidělovat úkoly například armádám hromadně podle toho, která armáda se pro daný úkol zrovna nejvíce hodí. Tak

například v situaci, kdy je potřeba zaútočit na nepřátelský hrad, je potřeba vybrat takovou armádu, která je jednak v dostatečné vzdálenosti, jednak dost silná na poražení nepřítele. V případě, že by pak poblíž hradu byla dostatečně silná armáda, bylo by dobré s touto armádou hrad zabrat a ne přijet s armádou, která je sice silnější, ale musí urazit zbytečně velkou vzdálenost. Pokud by se však armády rozhodovaly individuálně a nebyla mezi nimi žádná koordinace, mohlo by se stát, že by se pro tento úkol zbytečně vybrala špatná vzdálenější armáda, která by místo toho mohla zastat jiný užitečnější úkol.

6 Teorie pro AI: Možné rozhodovací techniky

Počítačového hráče budeme chápat jako agenta. Znamená to tedy, že získává znalosti z herního prostředí, na jejich základě se pak rozhoduje, jaké akce by měly provést a následně je provádí, čímž může herní prostředí také měnit.

Cílem rozhodování je najít v dané situaci nejlepší možnou akci, nebo případně celou posloupnost akcí z množiny všech potenciálních akcí. Zdrojem pro toto rozhodování jsou agentem získané znalosti, které můžeme dále rozdělit na externí a interní. Externí znalosti jsou takové informace, které reprezentují herní stav, například pozice armád na mapě, vlastnictví měst a podobně. Naproti tomu interní znalosti jsou takové informace, které reprezentují vnitřní stav agenta. Nalezené akce pak opět mohou ovlivnit jak externí, tak interní stav agenta.

Rozhodnutí můžeme dělit na strategická a taktická. Strategická rozhodnutí jsou rozhodnutí vyšší úrovně. Zabývají se dlouhodobými cíli, týkají se spíše celých skupin jednotek a zahrnují větší množství akcí. Taktická rozhodnutí jsou rozhodnutí nižší úrovně. Soustředí se spíše na krátkodobé cíle a týká se přímo jednotlivých jednotek. Zároveň je však přitom právě ovlivněno vyššími strategickými cíli.

V následujících podkapitolách je stručný přehled základních rozhodovacích technik, které lze aplikovat jak na taktická, tak na strategická rozhodnutí. Názvy některých technik jsou pro nedostupnost českého ekvivalentu raději ponechány bez překladu. Většina těchto technik se nepoužívá jen v herní AI, ale také v mnoha dalších oborech a často se jedná o formálnější a někdy také složitější modely, než se kterými pracuje herní AI. V tomto textu se proto věnujeme pouze popisu z pohledu herní AI.

6.1 Rozhodovací stromy

Rozhodovací stromy (anglicky decision tree) (12 str. 303) patří k jedné z nejjednodušších rozhodovacích technik a to jak z hlediska implementace, tak samotného fungování. Jejich velkou výhodou je kromě rychlosti také dobrá přehlednost.

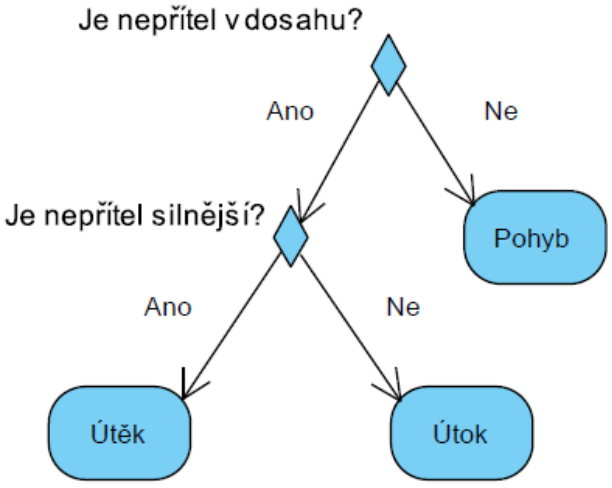
V rozhodovacím stromě pro nalezení konkrétní akce slouží podmínky, které právě rozhodovací strom svou strukturou realizuje. Obrázek 6-1 zobrazuje příklad jednoduchého rozhodovacího stromu.

6.1.1 Definice

Rozhodovací strom je kořenový strom, který obsahuje právě dva typy uzlů. Vnitřní **rozhodovací** uzly, které obsahují test na podmínku a odpovídají tak jednotlivým rozhodovacím krokům. Pro každý výsledek testu, který vede k nějaké akci, z nich vychází hrana do dalšího potomka. Pokud by tedy některý výsledek testu již žádnou akci nepřipouštěl, tak pro tento výsledek žádná hrana nepovede. Dále obsahují listy, které představují výslednou akci vzhledem k vyhodnocení jednotlivých testů na cestě od kořene do listu.

6.1.2 Rozhodovací algoritmus

Algoritmus rozhodování začíná v kořeni. Od něj postupuje po jednotlivých rozhodovacích uzlech až k odpovídajícímu listu. V každém rozhodovacím uzlu se vyhodnotí příslušný test a tím se určí, do kterého potomka daného uzlu se následně vydat. Pokud v nějakém rozhodovacím kroku pro výsledek testu hrana do potomka neexistuj, tak vyhledávání končí a žádná akce vrácena není. Takto algoritmus postupuje, dokud nedorazí do listu, kde vrátí příslušnou akci.



Obrázek 6-1 Příklad rozhodovacího stromu

Rozhodovací stromy nevyžadují žádný vlastní formát znalostí a není tedy potřeba překládat z formátu, který používá hra, do formátu, který by používal strom. Proto také většinou rozhodovací strom přistupuje přímo k hernímu stavu.

6.1.3 Výhody

- Jsou jednoduché na implementaci, snadno pochopitelné a poměrně přehledné.
- Poskytují mnoho možností pro další optimalizace a při rozšíření také možnost učení.

6.1.4 Nevýhody

- S rostoucím počtem podmínek se stávají méně přehledné a hůře se udržují.

6.2 Konečné automaty

„Konečné automaty, jsou v současné době bezpochyby nejvíce používanou technologií v programování umělé inteligence ve hrách. Jsou koncepčně jednoduché, efektivní, snadno rozšiřitelné a přesto dostatečně silné, aby dokázaly zvládnout rozličné situace.“ (13)

„Konečné automaty tvoří převážnou většinu rozhodovacích technik používaných v současných hrách.“ (12 str. 357)

Konečné automaty[8] berou při svém rozhodování v úvahu herní svět, podobně jako rozhodovací stromy. Avšak zatímco u rozhodovacího stromu probíhalo rozhodování vždy znovu od začátku a každá akce byla potenciálně dosažitelná, u konečného automatu je toto rozhodování obohaceno o jednoduchou paměť, která reprezentuje agentův současný stav. Při rozhodování jsou brány v úvahu pouze akce vzhledem k agentovu stavu. S každým stavem je potom spojeno agentovo chování, tedy určité akce, které agent provádí a také pravidla, která určují přechod do dalších stavů.

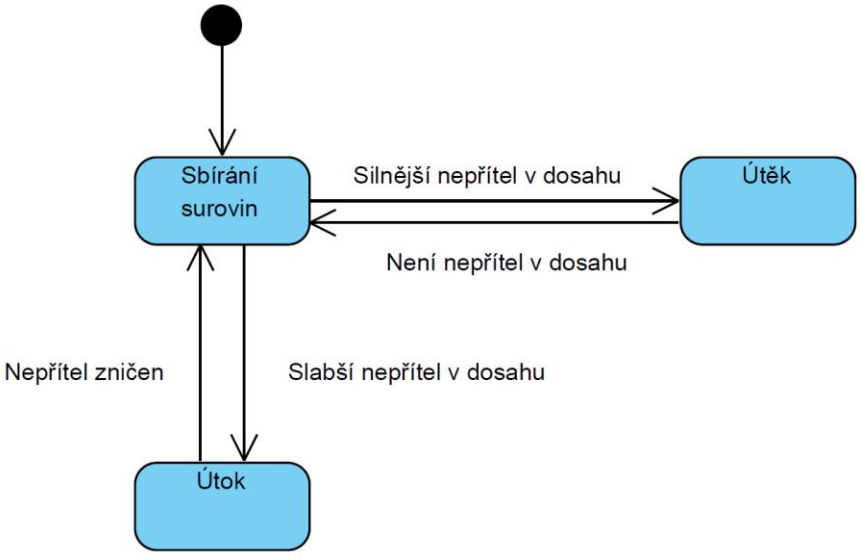
Obrázek 6-2 ukazuje příklad jednoduchého konečného automatu.

[8] Anglicky final state machine, zkráceně FSM

6.2.1 Definice

Pojem konečného automatu sice pochází z teorie vyčíslitelnosti, ale model používaný v herní AI se od tohoto teoretického modelu v mnoha ohledech liší. Model používaný v herní AI má hlavně následující odlišnosti:

- Protože zde každý stav reprezentuje agentovo chování, mají všechny stavy také odpovídající akce a s tím spojený programový kód, takže vždy, když se změní stav, změní se také agentovo chování.
- Přechodová funkce je zapouzdřena spíše v jednotlivých stavech, než v samotném automatu, což velmi usnadňuje přehlednost a podporuju dekompozici problému na menší části. Každý stav tedy sám zná pravidla, která určují přechod do jiného stavu.
- Přijímací stavy nejsou příliš podstatné, protože automat přijímá vstup tak dlouho, dokud je potřeba. A protože stavy reprezentují agentovo chování a záleží tedy jen na tom, jak se agent chová a ne na tom, zda přijímá nebo ne, není přijímací stav potřeba. Při rozšíření je však možné ho přesto využít například pro signalizaci konce práce automatu a přechod na jiný automat.



Obrázek 6-2 Příklad konečného automatu

Na samotné generování akcí se u herních konečných automatů často používá kombinace metod Mooreho a Mearlyho automatu a tím tak vzniká následující rozšířené schéma:

V každém kroku automatu, ve kterém se neuskutečnil přechod, se volají akce příslušné odpovídajícímu aktivnímu stavu, tedy akce odpovídající setrvávání v tomto stavu. Při změně stavu se postupně volají akce odpovídající opuštění současného stavu, což může při skutečné implementaci odpovídat mimo jiné místu pro úklid před přechodem do dalšího stavu, akce odpovídající tomuto vyvolanému přechodu a nakonec akce odpovídající vstupu do nového stavu, což opět může při skutečné implementaci být mimo jiné místo pro počáteční inicializaci.

6.2.2 Možnosti implementace

Pro složitější automaty je téměř nutné nepoužívat pouze naivní implementaci, například pomocí konstrukce switch nebo přechodové tabulky, protože taková implementace by už byla s rostoucí složitostí automatu velmi špatně čitelná a udržovatelná. Je proto většinou lepší mít generickou implementaci a použít stavy jak pro funkcionalitu, tak pro udržování přechodové funkce. Nejen stavy, ale také přechody jsou zde samostatné objekty a mohou obsahovat libovolný kód.

Skutečná práce tak probíhá přímo uvnitř jednotlivých stavů a automat pouze provádí pomocí aktivního stavu a jeho přechodů odpovídající aktualizace. Celý problém je tím dekomponován na samostatně fungující celky a je proto mnohem lépe udržovatelný, přehlednější a snadno rozšiřitelný.

6.2.3 Výhody

- Jsou rychlé a poměrně snadno implementovatelné.
- Dají se dobře ladit, což platí zejména u generické implementace, kdy se agentovo chování dekomponuje na menší celky. Při výskytu problémů se do stavů vloží sledovací kód, díky kterému lze zjišťovat, jak se automat v průběhu práce chová.
- Jsou flexibilní a dobře se rozšiřují přidáváním nových stavů.

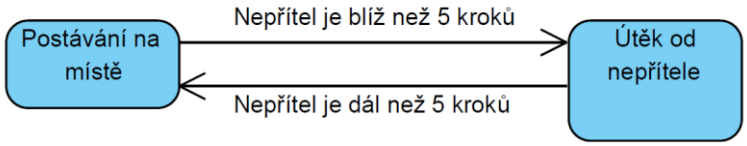
- Jedná se o dobře zavedenou a velmi často používanou herní techniku nejen v problému rozhodování.

6.2.4 Nevýhody

- Ačkoliv se dobře rozšiřují, tak v případě přidávání většího množství nových stavů se snižuje přehlednost z důvodu rychle rostoucího počtu přechodů mezi stavy.

- Jejich chování je předvídatelné, což se však dá aspoň částečně odstranit použitím náhodného výběru ze splňujících přechodů.

- Může se vyskytovat problém oscilace. Ten nastává v případě, kdy je určitá událost mezi dvěma stavy příliš proměnlivá, tedy snadno vzniká a následně hned zaniká. To pak má za následek, že se odpovídající stavy příliš rychle vzájemně střídají mezi sebou, což způsobuje dojem nerealistického chování. Příklad zobrazuje Obrázek 6-3, kde by agent při útěku stále zastavoval a opět utíkal podle toho, jak by se k němu přibližoval protivník. Problém oscilace však v případě tahové strategie v podstatě odpadá.



Obrázek 6-3 Automat s problémem oscilace

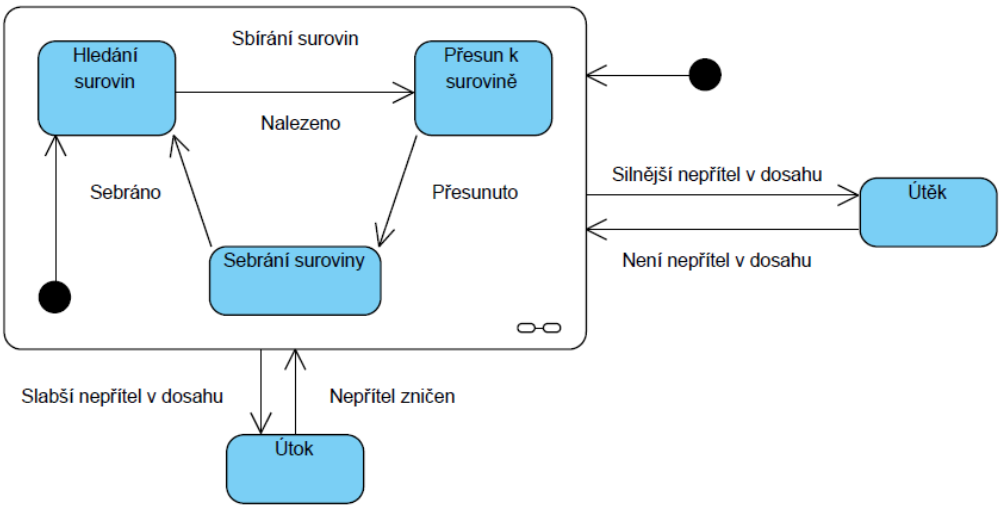
6.3 Zásobníkový konečný automat

Jedním z rozšíření základního modelu je zásobníkový konečný automat (anglicky stack-based FSM) (14 str. 295). Díky němu je umožněno vytvářet agenty, kteří mohou například přerušit určitou akci, kterou právě provádějí a zpracovat neočekávané situace a následně se vrátit do původního stavu.

6.4 Hierarchické konečné automaty

Ještě silnější rozšíření modelu automatu se zásobníkem je hierarchický konečný automat (anglicky hierarchical FSM, nebo HFSM). V tomto případě mohou jednotlivé stavy představovat konečné automaty. Proto jsou stavy

s touto vlastností nazývány hierarchické stavy. Obrázek 6-4 zobrazuje příklad hierarchického konečného automatu.



Obrázek 6-4 Příklad hierarchického konečného automatu

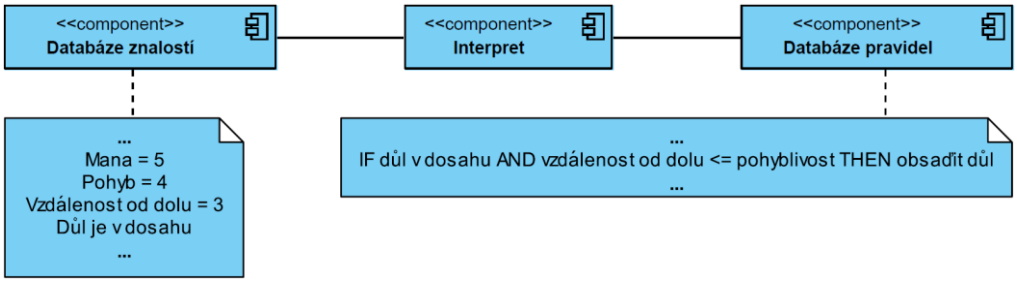
Hlavní z výhod hierarchických konečných automatů je, že ještě lépe podporují dekompozici problému na menší, lépe udržovatelné a přehlednější části. Čím větší automat musí být, tím více je tato výhoda relevantní. Navíc pomáhají odstraňovat duplicity kódu, protože umožňují vložit některé společné chování do odděleného automatu, který může být sdílen mnoha různými stavy původního automatu, případně opět dalšími automaty.

6.5 Produkční systémy

„Produkční systémy modelují lidské uvažování řešením problémů krok za krokem, jako když odborníci aplikují své zkušenosti. Znalosti jsou uloženy ve vysoce implicitní reprezentaci, což znamená, že mnohá další fakta musí být odvozena.“ (15 str. 124)

6.5.1 Definice

Produkční systémy se skládají ze tří hlavních částí. Jsou to **databáze znalostí** (označována také jako pracovní paměť (15 str. 125)), **databáze pravidel** a **interpret**. Strukturu zobrazuje Obrázek 6-5.



Obrázek 6-5 Struktura produkčního systému

Databáze znalostí obsahuje všechna systému známá fakta. Databáze pravidel je množina if-then konstrukcí, které se skládají z hlavy, což jsou podmínky pro splnění pravidla a z těla, což jsou akce nebo aktualizace, které se mají při splnění pravidla provést. Interpret spojuje obě části dohromady. Prochází databázi znalostí a zjišťuje, které pravidlo je splněno a které by tedy mělo být vybráno a také řeší případné konflikty při splnění více pravidel současně, protože je vždy potřeba vybrat jen jedno pravidlo.

Tělo pravidla tedy může obsahovat jak aktualizace, tak skutečné herní akce. Aktualizace neprovádí žádnou skutečnou změnu v herním prostředí, ale zato mohou měnit databázi znalostí přímo, čímž tedy mění spíše vnitřní stav systému. Naproti tomu herní akce zasahují přímo do herního prostředí. Produkční systém sám přímo akci neprovádí, ale pouze ji vrací hernímu prostředí, které právě může herní akci skutečně vykonat a tím případně opět měnit databázi znalostí.

6.5.2 Metody odvozování

Pro produkční systémy existují dvě metody odvozování: dopředné řetězení (anglicky forward chaining) a zpětné řetězení (anglicky backward chaining). Liší se v přístupu, jak jsou pravidla na databázi znalostí aplikována.

Dopředné řetězení začíná se známými znalostmi v databázi a opakovaně aplikuje splněná pravidla, dokud nedosáhne zamýšleného výsledku. Každý cyklus tohoto procesu se dá rozdělit do tří hlavních fází (15 str. 130): určení množiny splňujících pravidel, výběr akce včetně řešení případných konfliktů a vykonání akce. Aplikace pravidel přitom průběžně mění databázi znalostí, ať už přímo přes aktualizace, nebo nepřímo přes herní akce.

Zpětné řetězení naopak začíná v zamýšleném výsledku a snaží se k němu zpětně dopracovat tím, že se snaží vrátit se z tohoto cílového stavu zpět do počátečního stavu. Každý cyklus procesu se dá opět rozdělit do tří hlavních fází: určení množiny těch pravidel, která, pokud by byla aplikována, by vedla k současnému stavu, výběr jednoho z nich včetně řešení případných konfliktů a aktualizace, kdy se stav databáze znalostí změní tak, aby odpovídal stavu před aplikací vybraného pravidla, čímž se získá nový současný stav.

Dopředné řetězení se hodí v situacích, kdy databáze znalostí obsahuje větší množství známých fakt a je potřeba se z nich dopracovat k určitému závěru. Příkladem může být situace z medicíny, kdy je potřeba určit diagnózu na základě symptomů. Zde jsou symptomy právě počáteční znalosti v databázi a cílem je odvodit z nich plynoucí diagnózu. (16)

Naproti tomu zpětné řetězení se hodí v případě, pokud je potřeba ověřit, zda určitý fakt může být odvozen z počáteční databáze znalostí. Příkladem může opět být situace z medicíny. V případě epidemie může systém předpokládat, že určitý jedinec má také příslušnou nemoc a snaží se tedy zpětně ověřit, zda je tato diagnóza korektní vzhledem k dostupným znalostem.

Vylepšením metody dopředného řetězení je Reteho algoritmus, který je také standardem pro hledání splňujících pravidel (12 str. 461).

6.5.3 Řešení konfliktů

Protože při testování může často nastat situace, kdy několik pravidel splňuje podmínky pro použití, je potřeba umět rozhodnout, které pravidlo skutečně vybrat a provést. Možností, jak v tomto případě dále postupovat je mnoho, uvedeny jsou zde proto nejobvyklejší metody.

První splňující pravidlo

Jedná se o nejjednodušší a nejrychlejší postup, při kterém se vždy vybere první splněné pravidlo. Je proto nutné, aby všechna pravidla v databázi byla předem pevně seřazena a při testování se procházela v tomto daném pořadí.

Nejdéle nepoužité pravidlo

Vybere se pravidlo, které bylo nejdéle nepoužité. Je proto nutné uchovávat si časy posledního použití každého pravidla.

Náhodný výběr pravidla

Vybere se náhodně některé z pravidel z množiny všech splňujících pravidel. Přístup tak poskytuje nepředvídatelnost.

Nejlepší pravidlo

Každému pravidlu se předem přiřadí jeho priorita. Při konfliktu je pak vybráno pravidlo s nejvyšší prioritou. Priorita může být také určována dynamicky, tedy podle toho, jak je pravidlo užitečné vzhledem k současné herní situaci.

Nejkonkrétnější pravidlo

Vybere se pravidlo, které má nejvíce podmínek. Přístup je založený na předpokladu, že čím více podmínek pravidlo má, tím je relevantnější.

6.5.4 Výhody

- Flexibilita. Databáze znalostí má vlastní reprezentaci, která se dá různě modifikovat tak, aby se dosáhlo potřebných výsledků. Také je možné přidávat nové symboly do této reprezentace. To plyne z toho, že tato pravidla jsou spíše data, než přímo kód.
- Modularita. Poměrně snadno se rozšiřují, protože jednotlivá pravidla reprezentují znalosti na atomické úrovni a je tedy možné je upravovat či kombinovat poměrně nezávisle na ostatních.

6.5.5 Nevýhody

- Pro použití produkčních systémů je velmi důležité, aby bylo možné všechny znalosti o řešeném problému uchovávat ve formě if-then pravidel, a aby rozsah řešeného problému nebyl příliš rozsáhlý. Pokud je totiž potřeba příliš mnoho pravidel, může se systém stát až moc výkonnostně náročný a těžko udržovatelný.
- Nejsou příliš expresivní, protože mnoho znalostí se musí postupně odvodit a je tak těžké předvídat chování pouze z daných pravidel.

- Činnost pravidel je někdy omezena syntaxí vnitřní reprezentace a není tak dostatečně silná. Navíc v případě zpětného řetězení bývá často tělo pravidla omezeno na jediný výraz, čímž se síla opět snižuje.

- *„V herní AI zůstávají jako poměrně neobvyklý přístup, částečně proto, že podobného chování může téměř vždy být dosaženo jednodušším způsobem použitím rozhodovacích stromů nebo konečných automatů.“ (12 str. 442)*

6.6 Goal-oriented behavior[9]

V tomto přístupu se agent snaží prostřednictvím svých akcí splnit vybraný cíl, který považuje v danou chvíli za nejdůležitější. Posloupnost takových akcí se nazývá plán.

6.6.1 Definice

Cíl představuje určitý úkol, který agent chce splnit. Agent se vždy řídí právě jedním aktivním cílem a tímto aktivním cílem je tedy určeno jeho chování. Každému cíli je možné spočítat jeho současnou důležitost, což je potřeba při výběru konkrétního cíle, kterým se agent bude dále řídit. Každý cíl má také stanovené podmínky, které určují, zda již došlo k jeho splnění.

Agent se snaží splnit cíl prostřednictvím plánu, tedy posloupnosti akcí. Tento plán může být pro všechny cíle buď statický, tedy vždy pevně určen, nebo naopak dynamicky sestavován až při výběru cíle v závislosti na herním stavu. Potom je tento přístup označován jako plánování, nebo přesněji Goal-oriented action planning, zkráceně GOAP.

K nalezení plánu pro vybraný cíl agent používá plánovač. Plánovač prochází prostor všech možných akcí a snaží se najít takovou posloupnost akcí, která tento plán bude realizovat, tedy splní vybraný cíl. Pokud plánovač plán najde, tak se agent řídí tímto plánem tak dlouho, dokud se cíl nesplní, neselže nebo se nepřeruší. Provádění plánu totiž může trvat delší dobu a mezitím může dojít ke změně herní situace, která již daný plán dále neumožňuje, případně vyžaduje jiný důležitější cíl.

[9] Označováno také Goal-oriented approach, Goal-directed approach, Goal-directed behavior

Celý proces se dá rozdělit do pěti základních na sebe navazujících částí. (17)

1. **Analýza aktuální herní situace,** kdy se pro každou oblast zájmu zjistí její aktuální relevance ve formě jediného čísla v předem definovaném rozmezí, například 1 až 100. Většinou se používá korespondence jedna funkce na jednu oblast zájmu, kdy pak každá funkce spočítá příslušnou oblast zájmu. Je však důležité, aby analýza dávala pro různé agenty v identické situaci vždy stejné výsledky a také aby použité rozmezí bylo konzistentní, čili aby dané funkce dokázaly využít vždy kompletní rozmezí.

2. **Určení cílů a vyhodnocení**, kdy se pro každý cíl zkontrolují oblasti zájmu a zjistí se, zda je daný cíl relevantní. Každý cíl může záviset na více než jen jedné oblastí zájmu. V této části je také možno zanedbat některé cíle, které získají příliš nízké hodnocení.

3. **Výběr nejdůležitějšího cíle.** V této části se vybere nejdůležitější cíl ze všech relevantních cílů. Nejjednodušší je seznam cílů seřadit a vybrat první z nich, je zde však také možnost používat složitější výběr než pouze na základě předešlého hodnocení a dále hodnocení upravovat různými modifikátory, například preferovat ofenzivní cíle a podobně.

4. **Nalezení nejlepšího plánu** pro vybraný cíl.

5. **Vykonání akcí nalezeného plánu.**

6.6.2 Hledání plánu

K samotnému hledání se velmi hodí mírně upravený algoritmus A* (18 str. 97). Pro jeho správnou funkci je potřeba ještě zavést cenu plánu. Úkolem plánovače je pak najít nejlevnější plán. Nejlevnější plán může být například nejkratší plán v počtu akcí nebo v délce trvání. Případně je možné při výpočtu ceny uvažovat další potřebné zdroje, jako například herní suroviny nebo magickou energii. Konečná cena však musí být jediná hodnota. Dále je ještě potřeba zavést heuristickou funkci, která bude odhadovat, jak daleko je průběžně sestavený plán od cíle, což znamená, jak moc se liší herní svět od toho, ve kterém je cíl splněn.

Ještě lepší řešení je použít algoritmus IDA*[10] (18 str. 101). Některé cíle totiž nemusí vůbec být dosažitelné, což by plánovač mohl zjistit až po úplném prozkoumání všech možných plánů. Pokud by však nebylo na plány žádné omezení, tak by se mohlo stát, že by plánovač stále přidával další akce a stejně by nikdy správný plán nesestavil. Je proto nutné stanovit plánům limit v rámci počtu akcí. Nutnost zavedení limitu pak nabízí použití algoritmu IDA*. Také je vhodné použití transpozičních tabulek, aby se zabránilo opětovnému prohledávání již dříve uvažované posloupnosti akcí. Algoritmus je blíže popsán v (12 str. 395).

Hledání plánu může mít dvě alternativy, odlišné tím, kterým směrem se snaží plán najít. První je dopředné hledání[11], kdy algoritmus začíná v současném stavu a snaží se najít posloupnost akcí vedoucí k cílovému stavu, to znamená stavu, ve kterém je cíl splněn. Druhou je zpětné hledání[12], které naopak začíná v cílovém stavu a snaží se zpětně najít posloupnost akcí, kterou by se pak v obráceném pořadí dostal ze současného stavu do cílového. Zpětné hledání může být intuitivnější, protože agent se nejprve snaží najít akci, kterou cíl splní, čímž opět dostane nový podcíl a snaží se znovu najít splňující akci, dokud nedojde do současného stavu.

6.6.3 Reprezentace stavu

Pro plánování je potřeba, aby bylo možné z následků všech akcí předpovědět nový herní stav a také umět z každého herního stavu generovat akce, které půjdou v tomto stavu použít. Dále je nutné používat reprezentaci herního stavu, kterou je snadné měnit, aniž by se měnil skutečný herní stav. Bylo by však neefektivní ukládat kompletní kopii herního stavu. Možné řešení je používat seznam odlišností, kdy se ukládají pouze informace, které se liší v novém stavu od původního (12 str. 339). Jiné řešení, popsané v (19), používá seznam vlastností, které popisují herní stav a představují minimální množinu informací, které jsou relevantní k danému cíli, vzhledem k němuž se plánuje. Každá akce má pak seznam předpokladů, což jsou vlastnosti, které musí před

[10] Iterative Deepening A*
[11] Anglicky Forward Search
[12] Anglicky Backward Search

použitím akce platit a seznam následků, což jsou změny ve vlastnostech po použití dané akce.

6.6.4 Výhody

- Adaptabilita. Agent se vždy snaží určit svůj cíl a najít plán na jeho splnění v závislosti na současném herním stavu.
- Agentovo uvažování působí díky plánování a adaptibilitě více jako lidské.

6.7 Minimax

Minimax se obvykle používá pro hru dvou a více hráčů v deskových hrách. Nejčastější použití má však pro hru dvou hráčů, proto dále budeme taky uvažovat pouze dva hráče. Myšlenka formálně vychází z teorie her.

Pro algoritmus je podstatné, aby se jednalo o hru s nulovým součtem a úplnou informací. Nulový součet znamená, že zisk jednoho hráče znamená ztrátu pro druhého. Úplná informace znamená, že je vždy možné s jistotou předpovědět následky všech možných tahů a všechny tyto tahy jsou také známé. Což ovšem neznamená, že by hráč věděl, co protivník opravdu zahraje.

Minimax rekurzivně prochází herní strom, který reprezentuje všechny stavy hry, do kterých je možné se dostat od počátečního stavu nějakou posloupností tahů obou hráčů. Začíná od kořene, dokud nedojde do listu, odkud se vrací zpět a pro každý zpětně navštívený uzel počítá jeho minimax hodnotu, která vyjadřuje ohodnocení stavu pro hledajícího hráče.

Nejlepší možná akce teoreticky vede k optimální strategii, tedy takové, která dává přinejmenším tak dobré výsledky jako libovolná jiná strategie hraná proti neomylnému protivníkovi. Najít optimální strategii však někdy vůbec nemusí být možné, protože kvůli příliš velkému počtu stavů nelze prohledat celý herní strom, tedy dostat se při hledání až do koncového stavu a zhodnotit, zda určitý tah vede k vítězství.

Asymptotická složitost algoritmu minimax je $O(b^m)$, kde b je faktor větvení a m je hloubka stromu.

6.7.1 α-β ořezávání

Problém příliš velké velikosti herního stromu částečně řeší technika známá jako α-β ořezávání, která při procházení stromu odstraňuje takové podstromy, do kterých bychom se stejně určitě ve hře nedostali, protože do nich nevede optimální strategie. Díky tomu tedy zjednodušíme výpočet a přitom neztratíme optimální řešení.

Účinnost metody do značné míry závisí na pořadí potomků jednotlivých uzlů, tedy na pořadí, v jakém jsou prohledávány jednotlivé tahy z příslušné pozice, protože čím dříve metoda zjistí, že určitý uzel už uvažován být nemusí, tím větší neprozkoumaný podstrom ořízne.

Pokud by se povedlo realizovat takové perfektní uspořádání, snížil by se faktor větvení na svou odmocninu, tedy asymptotická složitost by byla $O(b^{m/2})$, kde b je faktor větvení a m je hloubka stromu.

6.7.2 Neoptimální strategie

Předchozí metody stále musí prohledávat herní strom až do listů. Tím sice teoreticky zaručují nalezení optimální strategie, ovšem pro větší stromy už by takové hledání probíhalo příliš dlouho.

Možností je hledat ne nutně optimální řešení, tedy ukončit prohledávání dříve a použít pro hodnocení heuristickou funkci, která bude odhadovat zisk pro hledajícího hráče v daném stavu. V tomto případě už listy herního stromu nemusí být koncové stavy hry. Účinnost algoritmu pak silně závisí na kvalitě této heuristické funkce. Odhad by měl samozřejmě být co nejpřesnější, což znamená správně řadit kvalitu koncových stavů ve stejném smyslu jako předchozí funkce zisku a pro nekoncové stavy odpovídat šanci na vítězství. Také by však neměl být příliš časově náročný.

Na tento upravený algoritmus lze ve stejném smyslu aplikovat α-β ořezávání.

6.7.3 Transpoziční tabulky

Dalším problémem spojeným s velikostí herního stromu je opakování stejných stavů. Často je totiž možné dostat se do stejného stavu různými kombinacemi

tahů. Potom je však zbytečné znovu takové stavy prohledávat, pokud už na ně algoritmus narazil při hledání někdy dříve a tedy už je ohodnotil. Právě k tomu slouží transpoziční tabulky, které si pamatují již jednou ohodnocené stavy. Obvykle se implementují pomocí hašování, protože je opět důležitý požadavek na rychlost takového ukládání a vyhledávání.

6.7.4 Náhoda

Minimax a jeho rozšíření sice předpokládají hru s perfektní informací, je však možné do nich také zakomponovat podíl náhody ve hře tak, že se vytvoří další typ uzlu, zvaný pravděpodobnostní uzel, nebo také uzel šance, kde větve takového uzlu představují jednotlivé diskrétní pravděpodobnostní hodnoty, které mohou v daném případě nastat (například házení kostkou ve hře, kde náhodu způsobuje hrací kostka). Použitím těchto uzlů se však také výrazně zvýší hloubka stromu, což například pro verzi algoritmu bez ořezávání zvyšuje asymptotickou složitost na $O(b^m n^m)$, kde n je počet pravděpodobnostních možností (například počet stěn kostky). Podrobnosti viz (18 str. 175).

6.8 Další možné přístupy

Mezi další možné přístupy patří například **architektura typu tabule**, což však není rozhodovací mechanismus sám o sobě, ale spíše slouží jako koordinátor mezi několika rozhodovacími mechanizmy, které nemusí být nutně všechny stejného typu a odráží tak intuitivní představu sdílené tabule, kam jednotliví experti přidávají své návrhy na řešení rozebíraného problému. (12 str. 475)

Silnou, avšak už poměrně komplexní možností jsou **genetické algoritmy** nebo **neuronové sítě**, mezi jejichž velké výhody patří mimo jiné možnosti učení.

7 Analýza AI: Vybrané řešení pro soubojovou část

Nyní po shrnutí teoretických předpokladů je potřeba rozhodnout, který přístup na řešení AI v jednotlivých částech zvolit.

7.1 Podněty pro výběr řešení

V soubojové části se nabízí výběr především mezi minimaxem a hladovým přístupem, což je takový přístup, který vždy v danou chvíli rozhoduje tak, aby maximalizoval svůj zisk. Proti minimaxu však stojí několik problémů.

Prvním je velký počet stavů, které mohou v soubojové části nastat. Na jednu stranu je sice situace zjednodušená oproti například šachům tím, že táhnout může vždy v danou chvíli jen jedna jednotka, na druhou stranu se ve hře vyskytují ještě kouzla, kterých může být velmi mnoho, a které počet možností ještě zvyšují. Není to však nejhorší problém.

Druhým a horším problémem je výskyt pravděpodobnosti při boji. Všechna kouzla mohou mít pravděpodobnost, že se povedou a také způsobené škody při úderu jsou dány rozsahem, ze kterého se vždy vybírá náhodně. Minimax sice podporuje pravděpodobnostní uzly, ale tím se opět zvyšuje jeho časová složitost, což je nežádoucí.

Třetím problémem je právě požadavek na rychlost při rozhodování. Minimax při větší hloubce prohledávání potřebuje samozřejmě více času, což je však pro boj nežádoucí, v něm má být AI hlavně rychlá, než vševědoucí. Tento problém by se dal částečně obejít tím, že by AI přemýšlela předem už při tahu protihráče. Bylo by pak ale potřeba zavést přerušovací mechanismy, které by dokázaly ukončit prohledávání ve chvíli, kdy by protihráč provedl tah a měl by tedy rychle táhnout počítačový hráč.

Lepší je proto použít jednodušší, ale zároveň mnohem rychlejší rozhodování na základě známých principů plynoucích z pravidel souboje. Mezi ně hlavně patří: (20)

- Útok na oddíly, jejichž přední jednotka už byla hodně zraněna, čili útok může ubrat hodně jednotek z oddílů.

- Útok na střelecké jednotky, které mohou střílet, což znamená, že ještě mají střelivo a nejsou pokryté, což je situace, kdy vedle nich stojí pro ně nepřátelská jednotka. V takovém případě už musí útočit nablízko a je jim tedy vrácen úder.
- Útok na jednotky, od kterých nehrozí protiúder při útoku, protože už vyčerpaly své odvetné útoky.
- V případě, že nelze na nikoho zaútočit, přesunout jednotky směrem k protihráči přes taková místa, kde hrozí nejmenší nebezpečí.

7.2 Konkrétní popis vybraného řešení

Svou strukturou, tedy použitím pravidel a způsobem rozhodování mezi nimi se zvolený systém inspiruje především myšlenkou **rozhodovacích stromů** a také **produkčních systémů**, ačkoliv se nejedná přímo o zmíněné metody.

Nyní zbývá analyzovat jednotlivé důležité kroky při rozhodování u vybraného řešení.

7.2.1 Rozhodovací pravidla

Vzhledem k předešlým postřehům je soubojová AI implementována pomocí posloupnosti if-then konstrukcí, které tedy představují jednotlivá rozhodovací pravidla. Při rozhodování se tato pravidla procházejí od prvního do posledního v již pevně daném pořadí. První splněné pravidlo je vybráno a je vykonána odpovídající akce.

Pro další popis pravidel předpokládejme, že síla jednotky, lépe řečeno oddílu, se počítá podle vzorce:

Síla jednotky = Útok * Průměrné škody * Počet jednotek v oddílu

Pravidla jsou následující:

1. Pokud je právě na tahu hrdina.
    a. Vyber náhodně mezi kouzlem a útokem. Pokud je vybrán útok, postupuj stejně jako v případě střelecké jednotky, jinak vyber náhodně jedno z možných kouzel.

2. Pokud je právě na tahu obyčejná jednotka.
    a. Pokud je právě na tahu střelecká jednotka.
        i. Pokud jednotka může provést útok na dálku.
            1. Najdi nejsilnější jednotku, která ti může ublížit. Pokud existuje, zaútoč na ni.
            2. Pokud neexistuje, najdi nejsilnějšího střelce. Ten už je tedy nutně pokrytý. Pokud existuje, zaútoč na něj.
            3. Pokud neexistuje, najdi nejsilnější jednotku. Ta už tedy musí existovat, jinak by bitva skončila, navíc nemůže být střelec ani nemůže jednotce ublížit. Zaútoč na ni.
        ii. Pokud už nemůžeš střílet na dálku, zkus najít naprosto bezpečné místo, kde tě nepřítel nemůže ohrozit. Pokud existuje, přesuň se tam.
        iii. Pokud takové místo neexistuje, postupuj jako jednotka nablízko.
    b. Pokud je právě na tahu jednotka nablízko.
        i. Najdi nejsilnější jednotku v dosahu svého útoku, která už nemá odvetný útok. Pokud existuje, zaútoč na ni.
        ii. Pokud neexistuje, najdi nejsilnější jednotku v dosahu. Pokud existuje, zaútoč na ni.
        iii. Pokud neexistuje, najdi nejsilnějšího střelce, který není pokryt. Pokud existuje, jdi směrem k němu.
        iv. Pokud neexistuje, najdi nejsilnější jednotku. Ta už musí jistě nutně existovat, jinak by už totiž bitva skončila. Jdi směrem k ní.

7.2.2 Problém kouzel

Problémem jsou kouzla. Kouzla jsou totiž definovaná externě, a proto nelze pevně do zdrojového kódu zahrnout pravidla na kouzlení, když nevíme, jaká kouzla nakonec ve hře opravdu budou. Pokud by měla AI kouzla vyvolávat rozumně a ne náhodně, musela by nejprve získat základní pochopení, co dané kouzlo dělá. To se dá získat částečně díky tomu, že máme kouzla rozdělená do kategorií podle cílové vlastnosti, kterou ovlivňují a také podle toho, zda jsou přátelská, tedy podporují hráčovu armádu, nebo nepřátelská, tedy oslabují nepřátelskou armádu. Takže například u nepřátelského kouzla, které ovlivňuje počet životů, můžeme poznat jeho účinek a pravděpodobně se jedná o bojové kouzlo ubírající cíli životy. Proto se pak takové kouzlo hodí v situaci, kdy chce hráč například dorazit silnou jednotku.

Takto by se dalo uvažovat, pokud by každé kouzlo ovlivňovalo jen jedinou cílovou vlastnost. V našem případě se však kouzlo skládá z efektů a jednotlivé efekty jsou pak společně s kouzlem také externě definovány. Některé kouzlo pak může být například takové, že je přátelské a přidává naší jednotce 2 body k vlastnosti útok, ale odebírá 2 body od vlastnosti obrana. Rozhodnutí pro použití takového kouzla je pak jistě mnohem komplikovanější, protože kouzlo nemá jednoznačně pozitivní nebo negativní účinek.

Poznamenejme, že v takovém případě má výhodu právě minimax, protože ten pouze postupně prohledává jednotlivé možnosti a nemusí proto vůbec řešit, co kouzlo vlastně dělá, ale jenom mu stačí znát následky uvažovaného kouzla pro určení dalšího stavu po aplikaci kouzla. Naneštěstí je však u minimaxu stále problém s pravděpodobností u kouzel.

7.3 Zkušenosti s metodou

Použitá metoda představuje kompromis mezi rychlostí a kvalitou počítačem řízeného hráče. Jak už bylo popsáno dříve, nejedná se zde o kritický bod, co se týče dopadů rozhodování počítačového hráče a proto tato hladová strategie za využívání popsaných bojových postřehů a principů pro bojovou část plně postačuje.

Navíc díky tomu je metoda velmi rychlá a je proto možné ji použít také při tahu hráče na velké strategické mapě v těch situacích, kdy bojuje počítačový protivník proti jinému počítačovému hráči nebo příšerkám na mapě. V tom případě totiž nehraje lidský hráč a tedy tento souboj „počítače proti počítači“ vůbec nevidí. Zde by pak při použití případné pomalejší metody byl problém s čekáním, které je jistě pro hráče většinou nepříjemné.

Uvažme totiž situaci, kdy se boje účastní lidský hráč přímo a tedy v podstatě vždy čeká pouze na tah soupeře, než zase dostane možnost hrát. V takovém případě se celková doba souboje v podstatě rozloží mezi jeho jednotlivé tahy a pokud jeden soubojový tah trvá o něco déle, není to pořád pro hráče tak citelné. Pokud však hráč není v souboji, nasčítá se samozřejmě tato doba na celý souboj a hráč tedy čeká velmi dlouho, než se opět dostane na tah. Ve hře by pak taková situace vypadala tak, že by na velké strategické mapě jedna armáda zaútočila na jinou armádu, pak by hráč dlouho čekal na vyhodnocení boje a pak teprve by opět protihráč prováděl další tahy na velké mapě.

Alternativou by bylo odhadovat výsledek souboje podle síly armád, ale takové vyhodnocení by mělo sklony k nepřesnosti. Případně by bylo možné použít složitější rozhodování pro souboje, kterých se lidský hráč účastní a rychlé pro souboje, kde se neúčastní.

7.4 Možná vylepšení

Metoda sice dává dobré výsledky, přesto je zde prostor k vylepšení, především ve zmíněné oblasti kouzel, kde současná metoda nemá a proto ani nevyužívá znalosti o vlastněných kouzlech. Porozumněním kouzlům by AI mohla sesílat kouzla mnohem efektivněji podle aktuální potřeby, případně šetřit manu před vyvoláním dražšího kouzla v situacích, kdy se hodí použít jiné levnější kouzlo. Také by se mohl udělat odhad výsledků boje a podle toho rozhodnout, zda AI nemá raději utéct pryč, než boj prohrát.

8 Analýza AI: Vybrané řešení pro taktickou část

V taktické části už se nabízí více z popsaných možností, jak rozhodování řešit.

8.1 Podněty pro výběr řešení

V následujících částech se nachází diskuse nad nabízenými řešeními.

8.1.1 Rozhodovací stromy

Problém u rozhodovacích stromů je, že se spíše hodí na taktické rozhodování v rámci jednotlivce, čili například armády, ale nehodí se už tolik na globální rozhodovací mechanismus pro všechny armády najednou. Odpovídající strom by byl pravděpodobně až příliš složitý, protože by musel ošetřovat obrovské množství případů. Jak však bylo poznamenáno, je potřeba mít mechanismus, který bude koordinovat akce mezi více objekty.

8.1.2 Konečné automaty

Stejně jako u rozhodovacích stromů je u konečných automatů problém s tím, že potřebujeme koordinovat více objektů. To je však u základního konečného automatu problém. Potřebujeme totiž, aby jednotlivé armády měli různé stavy, například některá by měla zrovna rekrutovat jednotky, jiná by měla zabrat důl nebo zaútočit na nepřítele. Pokud bychom však měli pouze jeden automat pro hráče, který by měl všechny armády najednou řídit a koordinovat, nebylo by možné popsat toto odlišné chování v danou chvíli u jednotlivých armád. Bylo by samozřejmě možné mít nějaký velký složený stav hráče, který by se skládal ze stavů jednotlivých armád, avšak zde jsou dva problémy. Zaprvé armád může být různé množství, čili by tento stav musel také uvažovat a proto ještě nějak kódovat případy, kdy nemá plný počet armád. Zadruhé, i kdyby to bylo možné, byl by takový automat neúměrně složitý.

Proto určitě nestačí mít jen jeden automat pro hráče. Na druhou stranu, pokud by byly automaty pro každou armádu zvlášť, tak by stejně právě bylo potřeba mít nad těmito automaty další koordinační mechanismus. Například v případě, že je potřeba zabrat důl a zároveň jinde zaútočit na nepřítele, tak by měl tento koordinační mechanismus vybrat pro každý úkol správnou armádu. To ale

znamená, že nestačí se pro každou armádu zvlášť podívat, co by mohla udělat v závislosti na jejím automatu a podle toho pro ni zvolit akci.

Předpokládejme například, že bychom pro armádu měli stav, ve kterém se jen potuluje po mapě a sbírá suroviny, který by měl přechod, že pokud armáda vidí silnějšího nepřítele, tak okamžitě uteče a když slabšího, tak na něj zaútočí. Takové jednoduché pravidlo je naneštěstí špatné, protože pokud by měl hráč v dostatečně blízkém okolí ještě silnější přátelskou armádu, tak by stačilo s tou silnější armádou nepřítele porazit a slabší armáda by ani nemusela utíkat, ale mohla by místo toho například sebrat nedalekou surovinu. Aby však toto bylo možné, musely by se jednotlivé automaty, řídící armády, také dívat na činnost ostatních automatů a podle toho se dále řídit, čímž by se poměrně jednoduchý koncept automatu stal velice složitým a nepřehledným.

8.1.3 Produkční systémy

Nepříjemným problémem u produkčních systémů je v našem případě to, že nelze jednoduše udělat databázi znalostí, například ohledně toho, jaké má hráč k dispozici budovy, protože tato data jsou uložená a definovaná externě. Proto nelze znalosti a pravidla natvrdo zakódovat do programu, ale musely by se nějak vytvořit právě na základě načtených dat ze souboru, což přináší mnoho komplikací. Databáze znalostí a pravidla zde totiž vyžadují jistou formálnější reprezentaci a všechna tato data by se tedy musela do této reprezentace ještě převádět. Není to však problém zásadní, přesto se lépe hodí jiný přístup, jak je popsáno dále.

8.1.4 Minimax

Pro strategickou část se minimax příliš nehodí, protože počet všech možných stavů hry je příliš obrovský a procházení příslušného herního stromu už do malé hloubky by proto bylo značně výpočetně náročné.

8.1.5 Goal-oriented behavior

Z předchozí analýzy vychází nejlépe goal-oriented behavior, protože v největší míře odráží podstatu problému, kdy chceme jednotlivým armádám přiřazovat určité úkoly podle jejich priority. Rozhodovací mechanismus nebude v tomto případě rozhodovat pouze v rámci jednotlivce, ale v rámci všech relevantních

hráčových armád, případně měst a bude tím tedy také poskytovat požadovanou koordinaci.

8.2 Konkrétní popis vybraného řešení

Použité řešení pro strategickou část se tedy v největší míře inspiruje a vychází z rozhodovací metody **Goal-oriented behavior** bez plánování. Velmi volně se také inspiruje **architekturou typu tabule**. Rozhodovací metoda pracuje tak, že na základě prostředků a aktuální herní situace určí možné cíle, vybrané cíle následně ohodnotí a podle toho vybere nejlepší, který se pak snaží naplnit prostřednictvím úkolů. Potom určí, jestli má stále volné prostředky a podle toho případně generuje další cíle. Například nejlepší cíl tedy může zaměstnat jednu armádu, která je pro něj nejvhodnější a pro další armádu už se vybere cíl jiný, opět v závislosti na tom, jak moc se daná armáda pro uvažovaný cíl hodí, což je právě chování, kterého jsme chtěli dosáhnout.

Proto je nutné uvažovat hlavně následující komponenty. Jsou to prostředky k dispozici, cíle, úkoly, správce cílů a sdílená tabule.

8.2.1 Komponenty metody

Prostředky k dispozici

Prostředky k dispozici znamenají především hráčovy armády, které mohou vykonávat určitý jim přiřazený cíl, dále hráčovy hrady, suroviny, doly, volné tahy. Například armáda jako prostředek dává možnost bojovat, sbírat suroviny a další, hrad jako prostředek nabízí jednotky, budovy a kouzla, suroviny jsou zase potřeba pro nakupování a podobně.

Cíle

Cíle reprezentují určitý zájem nebo stav, kterého chce AI dosáhnout. Pro své vyhodnocování a funkci využívají prostředky k dispozici, které si mezi sebou rozdělují. Samotné cíle však ještě nejsou herní akce, ale jedná se o abstrakci určitého zájmu. Ke konkrétnímu naplnění cíle slouží úkoly, které cíl pod sebou sdružuje.

Každý cíl tedy obsahuje skupinu úkolů, které mohou tento cíl splnit. Určitá skupina úkolů má totiž často společný jeden či více cílů. Například pro každou

armádu nepřítele existuje cíl zničit tuto konkrétní armádu. To však lze provést více hráčovými armádami a proto pro každou hráčovu armádu existuje pod tímto útočícím cílem sdružený úkol, který odpovídá skupině akcí potřebných pro splnění cíle touto armádou. Zároveň podrobnosti například o síle nebo nebezpečnosti nepřátelské armády jsou využívány všemi těmito sdruženými úkoly, proto je vhodné úkoly takto seskupovat. Také pokud by nebyl útok na armádu už aktuální, zneplatní se také všechny příslušné úkoly.

Pro každý možný herní zájem tedy existují odpovídající cíle. Jedná se o cíle **útok na armádu, útok na hrad**, **zabrání dolu**, **sebrání suroviny**, **postavení budovy**, **posílení armády**, **posílení hradu**, **spojení armády**. Cíl představuje pouze hlavní poslání zájmu, ale armáda může například při plnění cíle „útok na armádu“ sebrat cestou nedalekou surovinu. Stejně tak například při posílení armády se hráčův velitel také naučí ve městě dostupná kouzla a podobně.

Úkoly

Úkoly jsou vždy přidruženy k jednomu či více cílům a slouží ke skutečnému vykonání těchto cílů. Vě většině případů se skládají z několika herních akcí, někdy však obsahují pouze jedinou akci, v závislosti na typu cíle a úkolu. Tak například postavení budovy znamená jedinou herní akci, zatímco třeba sebrání surovin může znamenat přesun armády na patřičné místo, poražení blokující příšerky a nakonec sebrání požadované suroviny. Ještě komplexnější úkol je pak například útok na armádu, kde ještě navíc hráčova armáda může cestou také sbírat případné na cestě ležící suroviny.

Úkoly jsou podobné jako cíle, až na vyjímky platí téměř korespondence jeden druh cíle na jeden druh úkolu. Zvláštní skupinu úkolů tvoří armádní úkoly, což jsou úkoly, které provádí některá z hráčových armád. Tak například pro cíl **„útok na hrad“** existuje několik armádních úkolů **„útok na hrad“**, jeden pro každou hráčovu armádu. Armádní úkoly jsou specifické především v tom, že potřebují uvažovat prostředí na mapě, hledat nejkratší cesty a podobně. Také v jejich vyhodnocování hraje výraznou roli vzdálenost, kterou by musely armády při úkolu urazit.

Správce úkolů

Správce úkolů obsahuje všechny cíle a spravuje jejich vyhodnocování, výběr, aktivaci, rušení a další s nimi spojené věci. K tomu využivá sdílenou tabuli. Představuje tedy hlavní rozhodovací mechanismus, který vše řídí. Každý hráč řízený počítačem má svého spráce úkolů.

Sdílená tabule

Sdílená tabule obsahuje všechna data, která je potřeba sdílet mezi úkoly. Často se jedná o průběná vyhodnocení, která pak jednotlivé cíle využívají ke svému hodnocení. Například aktuální potřeba surovin, ohrožení jednotlivých armád, hradů a další.

Mapa vlivu[13]

Ačkoliv český překlad termínu možná zní poněkud kostrbatě, je mapa vlivu velmi podstatnou součástí rozhodování. Udržuje totiž informace o vlivu jednotlivých armád, který je určen jejich sílou a vzdáleností od odpovídajících souřadnic. Díky tomu AI hráč dokáže rozpoznat nebezpečná místa a vyvarovat se jich. Obecně může matice vlivu obsahovat také další informace, například bohatost oblasti na suroviny, v našem případě se však jedná pouze o vliv armád. Mapa tedy obsahuje pro jednotlivá políčka hodnotu nebezpečnosti podle nedalekých armád. Hodnota nebezpečnosti políčka je určena jako součet nebezpečnosti těchto armád, kde nebezpečnost od jedné armády je daná poměrem síly a vzdálenosti armády. Nebezpečnost armády se tedy nepromítá na celou herní mapu, ale pouze do určité nastavené vzdálenosti, obvykle do vzdálenosti dvou tahů.

Hradní a armádní vyhodnocení

Hradní a armádní vyhodnocení sdružuje informace o vyhodnocení všech hradů a také armád na strategické mapě. Slouží především k tomu, aby hráč mohl získávat informace o užitečnosti hradů a podle toho vážil útok či obranu a také aby věděl, která armáda je víc nebezpečná než jiná a podobně.

Částečně se tedy funkce vyhodnocení shoduje s funkcí mapy vlivu. Samotná mapa vlivu však na toto nestačí, protože je potřeba uchovávat více informací,

[13] V anglickém originále influence map

než jen poměr sil na určitých souřadnicích. Je také potřeba znát hráče, který jednotku vlastní, vzdálenost například armády k hradu a podobně. Důležitost dalších informací je popsána v další části textu.

Armádní plánovač

Armádní plánovač sdružuje informace o úkolech přiřazených určité armádě. Existuje vždy jeden plánovač pro každou hráčovu armádu. Díky plánovači je možné zjišťovat, zda armáda už neplní nějaký úkol a není proto k dispozici pro další úkol, nebo zda je pro další úkol volná.

8.2.2 Princip práce metody

V následující části popíšeme, jak probíhá hledání cíle, úkolu pro něj a akce, kterou AI v dalším kroku provede.

Postup se dá rozdělit do několika fází, jsou to vyhodnocení cílů, generování úkolů, třídění úkolů a výběr nejlépe ohodnoceného, provedení akcí vybraného úkolu a pokud zbývají další prostředky, tak opět od začátku.

Vyhodnocení relevantních cílů

První fáze při rozhodování je vyhodnocení cílů. Zde správce cílů prochází všechny cíle a zjišťuje, zda jsou v současné situaci aktuální. Pokud ano, tak provede jejich vyhodnocení podle aktuální situace.

Generování možných úkolů

Ve druhé fázi jsou generovány úkoly. Správce cílů prochází opět všechny cíle a pro každý zkontroluje všechny jeho úkoly a zjistí, jak moc jsou jednotlivé úkoly pro tento cíl užitečné. Pokud usoudí, že úkol je dostatečně užitečný a je možné ho pak provést, tak ho zařadí do seznamu možných úkolů pro další zpracování. V části generování můžou cíle zvyšovat ohodnocení některým úkolům. Například cíl obrana hradu může zvýšit prioritu všem úkolům, které směřují směrem k příslušnému hradu a budou v něm dřív než útočník.

Výběr nejlepšího úkolu

Ve třetí fázi už má správce seznam možných úkolů, společně s jejich určeným ohodnocením. Podle toho určí nejlepší úkol, který by se měl provést. K tomu

teoreticky stačí setřídit zadaný seznam. Alternativou by mohlo být průběžné pamatování nejlepšího úkolu, což však nestačí, jak je patrné dále.

Provedení vybraného úkolu

Ve čtvrté fázi se správce pokusí provést vybraný úkol, takže určí jeho podakce a pokusí se je provést. V této části však nemusí být úkol úspěšně proveden. Pokud úkol selže, je potřeba úkol prohlásit za selhaný a vybrat jiný úkol. Právě proto nestačí mít uložený pouze jeden nejlepší úkol, protože je možné, že bude potřeba vybrat druhý nejlepší úkol v případě selhání prvního, případně třetí atd.

Provádění úkolu může trvat delší dobu. Během provádění podčásti tak musí AI čekat na zpracování, například při přesunu armády musí počkat, než se armáda v herní logice přesune. Proto je každý úkol možné suspendovat, kdy úkol čeká na skončení určité herní akce a po jejím skončení opět pokračuje v práci. Pro informování o skončení očekávané akce slouží správce cílů, který přijímá od hráče přeposlané herní odpovědi. Podle nich pozná, zda je potřeba úkol opět aktivovat.

Zhodnocení situace a případné opakování

V poslední fázi správce zjistí, zda zbývají volné prostředky, pro které by mohl hledat cíle a generovat úkoly. Pokud ano, tak provede opět hledání dalšího úkolu. Zde totiž nestačí pouze vybrat další úkol v řadě, protože situace se může výrazně změnit provedením posledního vybraného úkolu.

8.2.3 Principy cílů a úkolů

V následující části už nebudeme popisovat do podrobností jednotlivé cíle nebo úkoly, protože do značné míry odrážejí předchozí analýzu a předpokládáme tedy, že jejich podrobnější výpis by byl na tomto místě zbytečně zdlouhavý. Jsou však obsaženy v dodatku v elektronické podobě na přiloženém datovém nosiči.

8.3 Hledání cest a mapa závislostí

Zatím jsme se nezmínili, jak AI vlastně určuje vzdálenost při vyhodnocování armádního úkolu nebo jak probíhá výpočet mapy závislostí. Přitom pro výběr

hlavně armádních úkolů je velmi důležité znát délku nejkratší cesty od armády do příslušného cíle. Vzdálenost totiž velmi podstatně ovlivňuje prioritu těchto úkolů.

V ideálním případě by AI znala nebo dokázala určit délku nejkratší cesty mezi libovolnými dvěma body, čímž by mohla řádně ohodnocovat příslušné úkoly. Není však z důvodu velké časové složitosti možné, aby AI při vyhodnocování každého úkolu vždy prohledala odpovídající cestu pro každý vyhodnocovaný úkol. Řešením by se možná mohlo zdát ukládání dříve nalezených cest. Zde je však problém s tím, že dříve nalezená cesta už nemusí v novém herním stavu platit. Nejkratší cesta totiž musí uvažovat nejen statické překážky, ale také příšerky na mapě, armády protihráče a také vliv těchto armád, kde nechceme poslat hráčovu armádu na nebezpečné území. Takové cesty by se přesto mohly při jistých úpravách používat jako předběžný odhad.

Protože nelze prohledat pokaždé cestu pro všechny úkoly, je potřeba provádět pouze odhad. Odhad se provádí jako vzdálenost vzdušnou čarou od počátku do cíle. Až pokud se úkol vybere jako nejlepší, provede se skutečné hledání cesty. Díky tomu odpadne hledání obrovského počtu hledaných cest v každé iteraci rozhodování. Kvůli tomu se však může stát, že po výběru úkolu se už nepodaří takový úkol provést, protože cesta ve skutečnosti existovat nebude. V tom případě pak úkol selže a vybere se jiný. Odhad vzdušnou čarou pak také velmi nepříjemně narušují překážky na mapě, kdy například hory brání nepřátelské armádě v cestě k hráčovému hradu, ale hrubý odhad přitom dává velmi malou vzdálenost. AI by pak kvůli pocitu ohrožení zbytečně bránila hrad a nemusela se vůbec hnout z místa. Proto alespoň pro takové situace používá AI skutečné hledání. Armád a měst totiž není tolik, aby bylo takové počítání přehnaně náročné a zároveň tak dává mnohem lepší výsledky, než kdyby AI uvažovala pouze vzdálenosti vzdušnou čarou.

Protože odhad je poměrně nepřesný, bylo by vhodné, kdyby AI mohla určit skutečnou vzdálenost alespoň pro takové úkoly, které se pravděpodobně také vyberou. Pro prioritu je podstatná vzdálenost, proto se dá předpokládat, že až na vyjímky jako například obrana hradu vybere AI takové úkoly, které nemají příliš dlouhou cestu. Na to se hodí pro změnu Dijkstrův algoritmus, který určí

nejkratší cesty z počáteční pozice do všech kolem do zadané vzdálenosti. Díky tomu má AI přesný odhad vzdáleností pro nejbližší úkoly, které pak budou vybrány s větší pravděpodobností a protože je pro ně předem zaručená cesta, tak provádění takového úkolu už po případném výběru neselže. Hledání mezi nejbližšími pozicemi provádí do vzdálenosti odpovídající až dvěma tahům.

Dijkstrův algoritmus používá AI také pro výpočet mapy vlivu pro nepřátelské armády. Opět provádí výpočet jen do zadané omezené vzdálenosti.

8.4 Zkušenosti s metodou

Použité řešení vytváří poměrně rozumnou AI a mělo by také do značné míry řešit požadavky, které jsme si na vytvářenou AI kladli, co se týče koordinace a skutečnosti, že všechny herní akce, které AI provede, mají určitou prioritu, se kterou by se měly vykonat. Z hlediska vývoje a ladění má takové řešení nepochybně své klady a zápory.

Na jednu stranu umožňuje v případě potřeby jednoduše přinutit AI soustředit se na námi požadované úkoly, například na sebrání určité suroviny. To lze provést velmi jenoduše tak, že přidáme ručně úkolu dostatečně velký bonus k prioritě. Čím větší bonus potom dostane, tím spíše se na něj dostane při výběru řada. Takový zásah je pak možno provést bez zásahu do dalších úkolů. Také je možné poměrně snadno sledovat, jak AI přemýšlí, kde je potřeba podívat se na vygenerované úkoly a kontrolovat přiřazené priority.

Na druhou stranu právě zmíněné priority jsou v některých případech poměrně těžko odhadnutelné a je potřeba klást velký důraz při určování vzorců pro tyto priority. Co se může zdát jako dobrý přínos pro úpravu priority jednoho úkolu, může ovlivnit výběr úkolu natolik, že další úkoly se nebudou vybírat správně. Například pokud AI bude silně preferovat určitou surovinu, například dřevo, kvůli požadavkům na stavbu, je otázka, jestli je její rozhodnutí vydat se pro něj kvůli nedostatku přes polovinu mapy takové chování, které bychom chtěli. Stejně tak může být problém třeba propagování priorit při obraně města, kdy zvyšujeme prioritu úkolům, které jsou blízko města a proto svým provedením neodvádějí armádu pryč od jeho blízkosti, pokud je jeho obrana potřeba. Při příliš velké prioritě pak nemusí AI vůbec od města odjíždět, protože se o něj

bude „bát“, při malé se však zase nebude chtít do města vrátit a raději bude „nerušeně sbírat suroviny“.

Kladem řešení je také rozdělení rozhodování do cílů a podúkolů, díky čemuž je složitější problém dekomponován na menší celky. Každý cíl má zde svou vlastní reprezentaci a lze do něj tedy zasahovat bez zásahu do dalších části. Až samozřejmě na skutečnost, že ovlivnění priority jednoho cíle má ve výsledku vliv také na ostatní úkoly.

Domníváme se tedy, že zvolené řešení vystihuje poměrně dobře požadované chování, bylo by však potřeba velmi důkladné testování a vyvažování vzorců při výpočtu vyhodnocení jednotlivých úkolů, aby AI hrále opravdu dobře a nepůsobila občas vlivem nastavení priorit trochu zmateně.

8.5 Možná rozšíření

Kromě vyvažování priorit jednotlivých úkolů, které se týká spíše testování, než programování, by bylo dobré rozšíření průběžné ukládání nalezených cest či vyhodnocení úkolů, protože ne všechny cesty a vyhodnocení se musí změnit po vykonání akce. Díky tomu by se mělo velmi zkrátit přemýšlení, které AI provádí. Doba čekání sice odpadne díky použití vláken pro AI hráče, jak je popsáno v implementační části této práce, ale přesto by takové zrychlení jistě nebylo na škodu.

9 Programátorská dokumentace

V následující části se budeme věnovat skutečné implementaci. Velká část myšlenek je popsána už v analýze, proto implementace spíše už jen rozvádí daná témata.

9.1 Platforma a programovací jazyk

Program je napsaný v jazyce C++/CLI na platformě .NET. Nepoužívá však společně spravovaný a nespravovaný kód, ale používá výhradně spravovaný kód.

Síťová komunikace je implementována pomocí technologie .NET Remoting. Pro implementaci uživatelského rozhraní je používáno pouze Windows Forms, je však navrženo tak, že využívá ve značné míře vlastní skupinu tříd pro grafické objekty, které kombinuje do větších celků, díky čemuž by případně bylo v budoucnu možné při drobnějších úpravách bez větších potíží Windows Forms nahradit rychlejší grafickou knihovnou.

9.2 Moduly programu

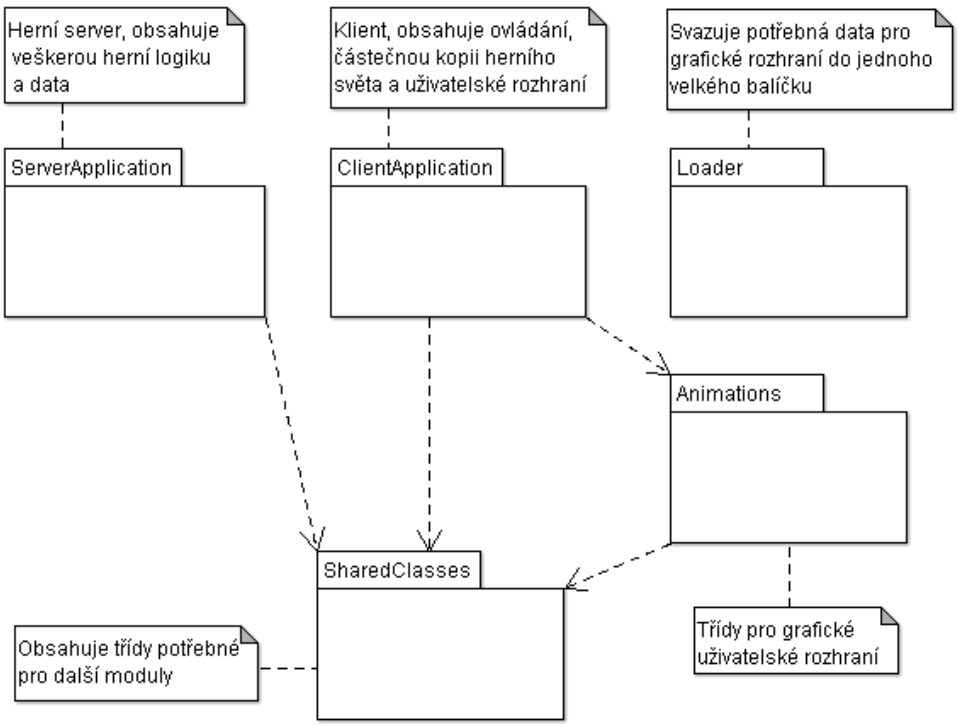
Program se dělí na několik hlavních modulů, které na sobě do určité míry závisí a vzájemně spolupracují. Dohromady program obsahuje pět modulů, z čehož první tři moduly představují samostatné programy, jsou to **ServerApp**, **ClientApp** a **Loader** a další dva fungují pouze jako knihovny tříd pro ostatní moduly, jsou to **SharedClasses** a **Animations**. Obrázek 9-1Moduly programu zobrazuje moduly programu.

9.2.1 Modul ServerApp

Modul **ServerApp** slouží pro implementaci herního serveru, ke kterému se připojují klienti, a také spravuje veškerou herní logiku, což znamená, že přijímá herní požadavky od klientů, které dále zpracovává, pokud jsou v danou chvíli relevantní a následně příslušným klientům zasílá zpět odpovědi.

9.2.2 Modul ClientApp

Modul ClientApp slouží pro implementaci herního klienta, představujícího vzdáleného klientského hráče. Zprostředkovává klientům komunikaci s herním serverem a tvoří pro ně potřebné uživatelské rozhraní.



Obrázek 9-1Moduly programu

9.2.3 Modul SharedClasses

Modul SharedClassed, neboli sdílený modul, představuje knihovnu, kterou využívají serverový, animační a klientský modul. Všechny třídy společné v těchto třech modulech jsou obsaženy právě v modulu **SharedClasses.**

9.2.4 Modul Animations

Modul Animations, tedy animační modul je grafická knihovna, která spravuje veškerou herní grafiku včetně herních animací a je využívána klientským modulem **ClientApp** pro zobrazování některých částí grafického rozhraní.

9.2.5 Modul Loader

Modul Loader je pouze jednoduchý program, který spojí potřebnou herní grafiku do jednoho souboru zdrojů[14], aby se urychlilo načítání při spuštění hry, které by jinak při načítání obrovského množství jednotlivých grafických souborů trvalo příliš dlouho.

9.3 Principy síťové komunikace

Síťová komunikace je implementována pomocí technologie .NET Remoting. Použitá architektura sítě je klient-server.

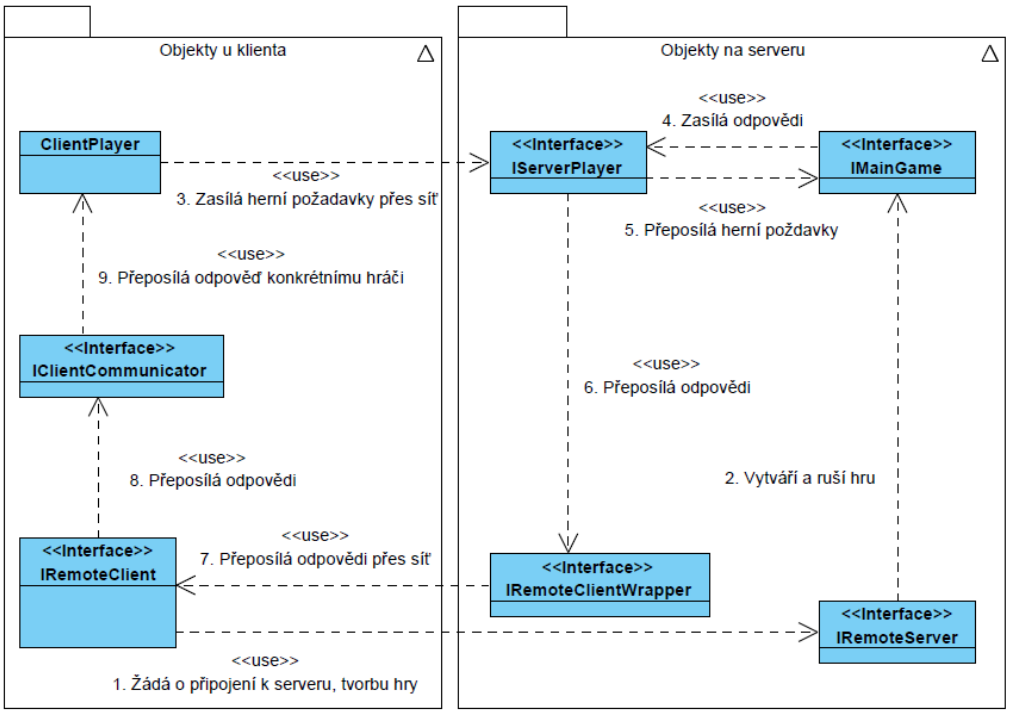
9.3.1 Druh přenosu

Jak bylo dříve rozebráno v analýze, pro spojení od klienta k serveru je použit synchronní přenos, od serveru ke klientovi asynchronní přenos.

9.3.2 Komunikace směrem klient-server

Pro komunikaci směrem od klienta k serveru slouží dvě třídy: **RemoteServer** a **ServerPlayer**. Obě třídy jsou odvozené od **MarshalByRefObject** a díky tomu mohou sloužit jako vzdálené serverové objekty. Pro **RemoteServer** se vytváří jediná instance, společná pro všechny klienty, pro **ServerPlayer** jedna instance pro každého hráče. Tyto objekty jsou uložené na serveru a klienti k nim pouze mají proxy jako ke vzdáleným objektům. Obrázek 9-2 ilustruje celkový pohled na síťovou komunikaci.

[14] V anglickém originále Resource file



Obrázek 9-2 Architektura síťové komunikace

Komunikace se pak realizuje tak, že klienti volají metody těchto vzdálených objektů. Liší se podstatně tím, jak pohlížejí na klienta, se kterým komunikují. **RemoteServer** zajišťuje připojování klientů k serveru, jejich registraci do hry, případně odpojování. Klienta, se kterým komunikuje, tak vnímá skutečně jako síťový objekt. Naproti tomu **ServerPlayer** slouží k přijímání herních příkazů a zajišťuje tak záležitosti spojené s herní logikou a klienta tedy vnímá spíše jako řídící prvek hráče aktuální hry.

Zasílání požadavku o akci pak probíhá následovně. Uživatel nejprve u klienta prostřednictvím uživatelského rozhraní určí nějakou akci, například pohyb armády. Uživatelské rozhraní nejprve překontroluje předběžnou oprávněnost akce, aby nepřeposílalo nesmyslné akce. Následně pomocí třídy **ClientPlayer** zašle tento požadavek na server prostřednictvím třídy **ServerPlayer**, ke které má proxy.

Některé herní akce se neprovádějí přímo, ale nejprve klient žádá o nalezení akce, čímž se akce také na serveru ověří, a pokud existuje, je vrácena. Až pak klient žádá o provedení nalezené akce. Slouží to k tomu, aby například klient mohl zobrazit nalezenou cestu hned už při pohybu myši a tím vlastně možnou

akci nabídnout uživateli. Teprve po vybrání akce kliknutím na dané souřadnice se tato akce pošle na server a provede se. Jiné akce takové předběžné nalezení nepotřebují a jsou prováděny požadavkem na server přímo.

Příklad - Pohyb armádou na strategické mapě

Na příkladě ukážeme provedení akce pohyb armádou. Jde o akci ověřovací, aby klient mohl vykreslit a nabídnout cestu ještě před provedením.

1. Uživatel má vybranou určitou armádu a pohne myší nad souřadnice, které odpovídají nějakému volnému místu. UI z toho rozpozná, že jde o akci „pohyb armádou“.

2. UI získá přes objekt třídy **ClientPlayer**, což je právě aktivní klientský hráč, proxy ke vzdálenému objektu **IServerPlayer**, což je odpovídající aktivní serverový hráč.

3. Přes **IServerPlayer** se volá metoda **FindWalkArmy.** V parametru se jí předají souřadnice počátku a cíle.

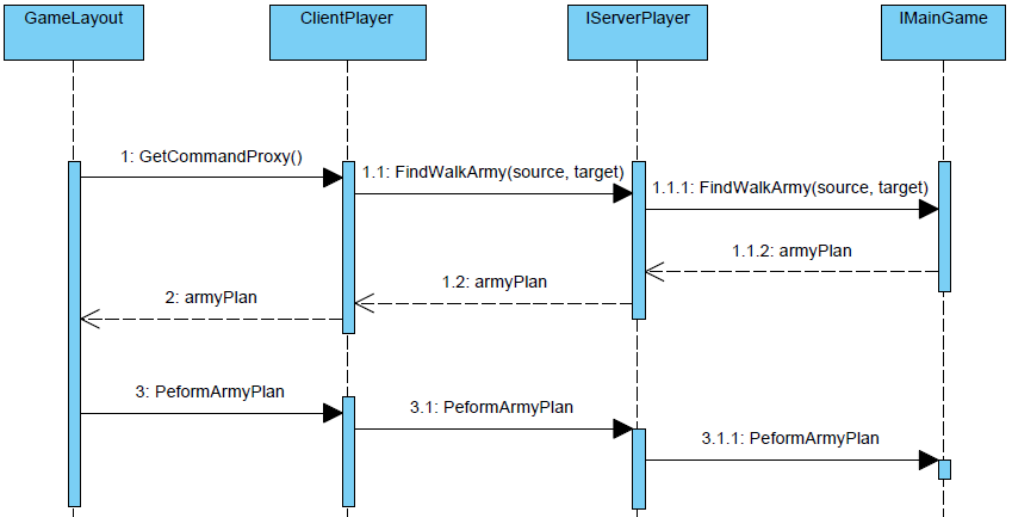
4. **IServerPlayer** ověří, že je daný hráč opravdu aktivní a pokud ano, tak jen přepošle požadavek třídě **TacticPart.**

5. Třída **TacticPart** zkusí najít odpovídající akci, což v tomto případě znamená najít pro armádu danou cestu. Pokud existuje, tak je vrácena v návratové hodnotě metody, jinak je vrácen jen nulový ukazatel.

6. UI uloží nalezenou akci. Pak uživatel klikne na stejné souřadnice. UI rozpozná, že se jedná o stejnou akci, kterou už zná, proto na server jen pošle požadavek na vykonání aktivní akce. Znamená to tak opět získat proxy a zavolat metodu **PerformArmyPlan** třídy **IServerPlayer**, který opět přepošle třídě **TacticPart.**

7. **TacticPart** vykoná akci a podle toho zašle případnou odpověď

Obrázek 9-3 ilustruje situaci hledání a provádění akce WalkArmy, tedy pohyb armádou.



Obrázek 9-3 Žádost o nalezení a provedení akce WalkArmy

9.3.3 Komunikace směrem server-klient

Komunikace tímto směrem již nevyužívá přímo volání vzdálených metod, ale funguje na principu zasílání zpráv. Aby však mohl server zasílat zprávy ke klientovi, potřebuje jednotlivé klienty nějak identifikovat a mít s nimi spojení. K tomu využívá třídu **RemoteClientWrapper**, která v sobě obsahuje proxy ke vzdálenému objektu **RemoteClient** u klienta. Zaslání zprávy ke konkrétnímu klientovi se potom realizuje jednoduše voláním metody **ReceiveRespond** třídy **RemoteClientWrapper**, která prostřednictvím metody **ReceiveRespond** třídy **RemoteClient** doručí zprávu až ke klientovi.

Příklad zaslání odpovědi na akci „pohyb armádou“

Na příkladě ukážeme, jak probíhá zaslání odpovědi na akci „pohyb armádou“, na kterou už byl zaslán a zpracován ověřovací dotaz a nyní se tedy provede.

- **TacticPart** obdrží požadavek na vykonání akce a provede tedy výpočet představující zadanou akci, čímž se změní herní stav. **TacticPart** proto vytvoří odpovídající zprávu jako objekt třídy **WalkArmyGR.**
- Tato událost se týká všech hráčů, proto **TacticPart** zašle zprávu všem hráčům pomocí funkce **TacticPart.SendRespondAll**, kde v parametru zadá vytvořenou zprávu.

- **SendRespondAll** iteruje přes všechny hráče, pro každého volá metodu **TacticPart.SendResond**, kde jako parametr předá zprávu a cílového hráče. Tato metoda pak už pouze pro zadaného hráče volá jeho metodu **ServerPlayer.ReceiveRespond**, jako parametr opět předá tuto zprávu.

- Pokud je **ServerPlayer** lidský hráč, hrající přes síť, tak předá zprávu objektu **RemoteClientWrapper**, který provádí zaslání zprávy přes síť.

- **RemoteClientWrapper** nejdříve ověří, zda snad stejnou zprávu právě nezasílal, což se může stát, pokud je jeden **RemoteClientWrapper** sdílen více hráči. Tento případ nastává, pokud na vzdáleném počítači hraje více hráčů najednou. Potom už nechceme, aby byla stejná zpráva posílání dvakrát, pokaždé pro jiného z těch hráčů. Je vytvořena obálka.

- **RemoteClientWrapper** volá **ClientCommunicator**.**ReceiveRespond**, kde jako parametr předá obálku se zprávou.

- **ClientCommunicator** zprávu přijme a musí ji nyní zatřídit, proto volá událost **ClientCommunicator.RespondAdded** pomocí **BeginInvoke**, aby nedošlo ke kolizím dvou vláken, serverového a klientského. Tuto událost sám zachytí a zpracuje jí tím, že zprávu zatřídí do fronty. Dále začne vyzvedávat všechny správně zatříděné zprávy a to tak dlouho, dokud ve frontě nějaké jsou. Každou předá odpovídajícímu lidskému hráči podle čísla obálky, případně ji zpracuje sám, pokud je to zpráva pro všechny hráče.

9.3.4 Formát zpráv

Všechny zprávy mají společného předka **IBasicRespond** a následně je pro každý druh zprávy odvozena vlastní podtřída. Díky tomu je k přenosu zprávy ke klientovi potřeba jediná metoda **RemoteClient.ReceiveRespond.** Druh přijaté zprávy se následně rozpozná podle výčtového typu zprávy a podle toho se s ní dále pracuje. Jinou možností by bylo rozpoznávat zprávu dynamicky přes typ její třídy, avšak předchozí způsob je efektivnější.

Kromě vlastních dat má každá zpráva ještě navíc svoje číslo, které slouží k jejímu správnému zatřídění u klienta, jak bylo popsáno v analýze.

RemoteClientWrapper představuje již v analýze popsanou vrstvu mezi třídou **IServerPlayer** a **IRemoteClient.** Přidává ke zprávě obálku, pomocí které pak určuje hráče na cílovém počítači, kterému zpráva skutečně patří.

9.3.5 Blokování zpráv

Server z hlediska herní logiky zasílá zprávy přímo hráčům, a proto se nestará, kde se ve skutečnosti hráči nacházejí. Může se ale stát, že na jednom počítači hraje více hráčů a pak v některých případech není potřeba stejnou zprávu zasílat vícekrát, jednou pro každého hráče na klientském počítači. A protože hra samotná se o klienty nestará, přenáší se tato odpovědnost na odpovídající **RemoteClientWrapper**. Ten před serializací a odesláním zprávy zkontroluje, zda stejná zpráva už nebyla klientovi poslána a pokud ano, tak prostě odesílání ignoruje.

9.3.6 Zpracování zpráv

Jak už bylo zmíněno v analýze, je potřeba zprávy po doručení ke klientovi správně seřadit a teprve podle tohoto pořadí zpracovávat. K tomu slouží třída **RespondQueue**, která vždy ví, jaké číslo má mít další příchozí zpráva a podle toho s novou zprávou nakládá. Pro ukládání zpráv pak využívá dvou struktur, fronty pro kompletní setříděné posloupnosti zpráv a bufferu pro setříděné, avšak nekompletní posloupnosti zpráv.

Při obdržení zprávy pak porovná očekávané číslo zprávy s číslem obdržené zprávy:

- Pokud má zpráva očekávané číslo, tak se hned zařadí do fronty zpráv ke konečnému zpracování.
- V opačném případě se nejprve dočasně uloží do bufferu a zkontroluje se, zda není v bufferu zpráva s očekávaným číslem. Pokud ano, tak se přesune do fronty zpráv ke konečnému zpracování a tento postup se opakuje, dokud jsou v bufferu odpovídající zprávy.

Po zatřídění zprávy do odpovídající struktury si třída **ClientPlayer** opakovaně vyzvedává zprávy z fronty s kompletní posloupností zpráv tak dlouho, dokud

jsou tam pro ni nějaké připravené. Podle jejich typu se pak rozhodne jak s nimi dále naložit.

9.3.7 Vytvoření serveru a připojování klientů

Nyní popíšeme jednotlivé kroky při vytvoření serveru a připojování klientů.

- Na serveru se vytvoří instance třídy **ServerApp**, která bude zajišťovat připojování klientů.
- V objektu třídy **ServerApp** se zaregistruje TCP kanál se zadaným portem a nastaví se tak, aby bylo možné v programu předávat reference v parametrech metod vzdálených objektů. K tomu je zapotřebí nastavit vlastnost kanálu **TypeFilterLevel** na hodnotu **TypeFilterLevel::Full.**
- Vytvoří se instance třídy **RemoteServer**. Protože tato třída je odvozená od třídy **MarshalByRefObject**, tak je možné ji nadále používat jako vzdálený objekt. Následně se tato instance nabídne klientům ke sdílení pomocí funkce **RuntimeServices.Marshal.**
- Na klientovi se vytvoří instance třídy **ClientApp**. Protože je odvozená od třídy **MarshalByRefObject**, bude možné ji později předat na server.
- V objektu třídy **ClientApp** se zaregistruje TCP kanál, nezadává se však už číslo portu, ale nechá se vybrat libovolný, který je k dispozici. Dále se vytvoří proxy ke vzdálenému objektu **RemoteServer**. Klienti však neznají skutečnou implementaci třídy, znají pouze její rozhraní, od kterého pak tato třída dědí.
- Klienti se následně připojují k hernímu serveru přes získaný vzdálený objekt **RemoteServer** metodou **RemoteServer.ConnectPlayer**, které předají v parametru odkaz na sebe, což bude sloužit k tomu, aby server mohl přes tento objekt zasílat klientovi potřebné zprávy.
- Pokud se připojení povede, **RemoteServer.ConnectPlayer** vrátí zpět klientovi odkaz na objekt třídy **RemoteClientWrapper**. Přes tento objekt pak klienti, představující lidské hráče, komunikují se serverem, představujícím hru. K tomu využívají metody vzdáleného objektu třídy

BasicPlayer, který je uložen na serveru, ale klienti k němu mají přístup přes získaný objekt třídy **RemoteClientWrapper.**

- Samotná hra pak už skutečnost, že pracuje přes síť, v podstatě vůbec nevnímá. Přijímá pouze od hráčů požadavky na herní akce, provede potřebné výpočty a upraví dále podle toho situaci. Pak může odpovědět hráčům metodou **BasicPlayer.ReceiveRespond**. A protože instance třídy **BasicPlayer** je vzdálený objekt a klienti na něj mají proxy, tak se dostane odpověď až k nim.

9.4 Reprezentace hry

Hlavní hra, představovaná třídou **MainGame**, se dělí na dvě hlavní podčásti, které reprezentují dvě odpovídající části hry. Jsou to části **BattlePart**, která představuje soubojovou část hry a dále **TacticPart**, která představuje taktickou a hradní část hry. Hradní část nemá samostatnou reprezentaci, je součástí taktické části, protože se vždy vztahuje ke konkrétnímu hradu. Ukázalo se, že oddělení na hrad a hradní část nepřináší žádné výhody, protože hradní část neměla jinou práci, než že pouze přeposílala příkazy odpovídajícímu hradu.

Základem pro zadávání herních příkazů je rozhraní **ICommand**, které sdružuje metody pro všechny herní akce, které může hráč ve hře provádět. Protože **BasicPlayer** implementuje toto rozhraní, je zaručeno, že hráč má k dispozici odpovídající metody. Samotné rozhraní **ICommand** ještě rozšiřuje rozhraní **IBattleCommand** a **ITacticComand**, které reprezentují herní příkazy pro odpovídající části hry.

Hráč je reprezentován třídou **ServerPlayer**, která implementuje **ICommand** a proto má stejné herní příkazy. Jedná se však o reprezentaci hráče jako logického herního subjektu, sdružující potřebné hráčovy metody a data, ne o reprezentaci klienta. Proto třída **MainGame** ani neřeší, zda se jedná o hráče komunikujícího přes síť, nebo o počítačového hráče. Samotný klient má pak přístup pouze k rozhraní **IServerPlayer** pro volání herních metod, samotná herní data jsou pro něj nedostupná.

Hráč prostřednictvím objektu třídy **ServerPlayer** komunikuje s herní logikou. Před zasláním požadavku herní logice vždy ještě **ServerPlayer** ověří, zda má

vůbec na daný požadavek právo, to znamená, jestli je hráč na tahu a také v odpovídající části hry, tedy bojové nebo taktické. Po ověření herní server zpracuje požadavek, a pokud je potřeba také odpovědět, což zpravidla je, zašle odpovídajícím hráčům zprávu se všemi potřebnými informacemi.

9.4.1 Reprezentace herních objektů

Všechny herní objekty na straně serveru jsou odvozené od třídy **BasicUnit.** Třída obsahuje pouze jednoznačné jméno objektu, používané jako identifikátor daného typu herního objektu v celém programu a celočíselný kód hráče, který tento herní objekt vlastní. Objekt však být vlastněn nemusí, v tom případě má kód hráče -2, nebo je vlastněn neutrálním hráčem, pak má kód hráče -1.

Například stromy či skály na taktické mapě vlastníka nikdy nemají, mají proto kód hráče -2, dosud neobsazený zdroj surovin má kód hráče -1 a armáda hráče má kód hráče od 0 výše, podle čísla hráče.

Na straně klienta je situace velmi podobná. Herní objekty jsou odvozeny od třídy **BasicInfo**. Každý objekt na serveru má pak protějšek u klienta. Například **TacticUnit** vs **TacticInfo**, **Castle** vs **CastleInfo** a podobně. Je tomu tak proto, aby klient mohl data o hře a jednotkách ukládat průběžně u sebe a nemusel se pořád dotazovat serveru. Na druhou stranu server potřebuje u těchto objektů uchovávat mnohé další informace potřebné pro herní logiku, které ale klient vůbec znát nemusí. Nabízelo by se proto odvodit vždy objekt na serveru od objektu u klienta, avšak byl zde pak problém s vícenásobnou dědičností. Chtěli bychom totiž zároveň, aby určitý objekt rozšiřoval jak klientský objekt, tak některý serverový objekt, což není možné.

Například máme třídu **Attacker** pro reprezentaci bojové jednotky v bojové části a chtěli bychom ještě odvodit **Sorcerer** pro jednotku, které navíc dokáže kouzlit. Pokud by ale **Attacker** už rozšiřoval klientský protějšek **AttackerInfo**, tak by nebylo takové odvození možné a museli bychom potřebné metody pro třídu **Sorcerer** psát znovu, místo jednoduchého podědění od třídy **Attacker**.

Vzhledem k tomu, že na serveru se často hodí využívat hierarchie dědičnosti jednotlivých objektů, je použito oddělené odvozování od jednotek na serveru a odděleně od jednotek u klienta. Jediná možnost, jak toto „vyřešit“ je použití

společného rozhraní, které by potom obě třídy, jak klientská tak serverová, rozšiřovaly. Bohužel nám to ale příliš nepomůže, protože rozhraní metody jen deklaruje a stejně se musí následně definovat jak u serverového objektu, tak u klientského objektu.

Každý herní objekt na serveru má proto virtuální **get()** vlastnost **BasicInfo ^ Details**, která pak v závislosti na skutečném typu objektu vytvoří a nastaví odpovídající třídu **BasicInfo**. Ta se potom používá, kdykoliv je potřeba poslat informace o herním objektu ze serveru přes síť klientovi. Kvůli tomu jsou třída **BasicInfo** a všechny třídy od ní odvozené označeny atributem **Serializable**, aby je bylo možné serializovat a poslat je přes síť klientovi. Zde se opět hodí oddělení od serverových protějšků, které tak vůbec tímto atributem označovat nemusíme.

9.4.2 Reprezentace herní mapy

Bojová herní mapa je reprezentována generickou třídou **BattleMatrixMap<T>,** což je třída zapouzdřující dvourozměrné pole, kde každá jeho buňka obsahuje příslušný objekt, stojící na odpovídajícím políčku

Taktická herní mapa je reprezentována generickou třídou **MatrixMap<T>,** což je třída také zapouzdřující dvourozměrné pole, kde však každá buňka obsahuje objekt třídy **AdvancedTile<T>,** což je generická třída, které reprezentuje celé taktické políčko. Obsahuje proto informaci o pozadí políčka, pokud nějaké je, dále informaci o objektu nacházejícím se na tomto políčku, jako jsou stromy, suroviny, armády, příšerky a podobně, a dále cestu, pokud tato přes políčko vede. Také obsahuje informaci o tom, odkud je dané políčko ovlivněné, pokud ovlivněné je. Ovlivnění znamená, že přes políčko není možné projít, protože stojí nedaleko nepřátelská příšerka, která by se musela nejprve eliminovat. Jedná se o informace užitečné při hledání cesty.

Třídy **MatrixMap<T>** a **BattleMatrixMap<T>** obsahují metody pro operaci s mapou, jako je zjištění objektu, cesty, či pozadí na zadaném políčku, dále jejich přidávání, odebírání, ověřování korektnosti souřadnic, atd.

Pro taktickou část se používají objekty třídy **TacticUnit** a od ní odvozené třídy na straně serveru a **TacticInfo** a odvozené na straně klienta. Pro bojovou část

jsou to **BattleUnit** a odvozené na straně serveru a **BattleInfo** a odvozené na straně klienta.

Protože na taktické mapě může stát na jednom políčku více herních objektů, je potřeba tuto situaci nějak zachytit. Například na daném políčku může být hrad společně s armádou, která hrad navštívila nebo zdroj surovin a na něm stojící armáda.

Z toho důvodu existuje odvozená třída od **TacticUnit,** třída **TacticDoubleLink.** Tato třída obsahuje odkazy jak na první objekt, například na hrad nebo zdroj surovin, tak na další objekt, který se nad ním nachází, zde tedy armádu. S tím je však potřeba počítat také při přesunu nebo mazání jednotek z míst, kde se nacházejí dva objekty najednou. Tak například při přesunu armády z hradu je potřeba nahradit **TacticDoubleLink** prvním odkazovaným objektem a druhý objekt, zde tedy armádu, odebrat a namísto toho tento objekt přidat na cílové souřadnice, kde tím však opět může vzniknout nový **TacticDoubleLink**, pokud je cílem hrad nebo zdroj surovin.

Zároveň některé objekty mohou zabírat více políček na mapě. Pro zachycení této situace slouží objekt třídy **TacticLink**, odvozený od třídy **TacticUnit.** Ten obsahuje souřadnice prvního políčka, na kterém se nachází skutečný objekt reprezentující tuto jednotku.

Například hrad s rozměry 2x2 bude na prvním políčku obsahovat skutečný objekt třídy **Castle** a na dalších třech políčkách už bude pouze **TacticLink** se souřadnicemi tohoto prvního políčka s objektem třídy **Castle.**

9.5 Herní příkazy

Herní příkazy zasílá hráč **ServerPlayer** herní logice a slouží k provádění akcí nebo k předběžným výpočtům. Dělí se na tři základní druhy. Požadavky totiž vznikají u vzdáleného klienta prostřednictvím uživatelského rozhraní, kde se nejprve provede výběr odpovídající akce a její předběžné ověření, zda vůbec přichází v úvahu a teprve potom, pokud je akce předběžně možná, se případně zašle požadavek na její výpočet na server. Samozřejmě se však ověřuje také na serveru.

9.5.1 Požadavky na nalezení a ověření akce

Jedná se převážně o akce v bojové nebo taktické části, kde je potřeba předem zjistit, zda je daná akce vůbec možná a pokud ano, tak nalézt řešení, aby se podle toho mohlo u klienta upravit uživatelské rozhraní.

Například nalezení cesty, kdy chceme nalezenou cestu zobrazit na taktické mapě, nebo ověřování, zda jednotka může zaútočit, přesunout se, atd.

Při těchto požadavcích server provede potřebná ověření, a pokud je daná akce možná, tak si uloží k ní potřebná data a zároveň je pošle zpět hráči v rámci návratu z funkce, protože není potřeba informovat ostatní hráče a tedy zasílat zprávy.

9.5.2 Požadavky na akce, které nepotřebují předběžné hledání

Jedná se o akce, kde není potřeba předem hledat cesty a podobně a ověření proběhne přímo před akcí, kde pokud by nebyla akce možná, tak se zkrátka neprovede. Jde převážně o akce spojené s hradem, jako třeba nákup armády, jednotek, budovy, atd.

Odpovědi se pak posílají odpovídajícím hráčům jako zprávy, protože už zde může být více adresátů, než jen hráč, který o akci požádal.

9.5.3 Požadavky na provedení předem nalezené akce

Jedná se pouze o zaslání požadavku o vykonání dříve nalezené akce, například provedení pohybu armády po již nalezené cestě.

Odpovědi se pak posílají odpovídajícím hráčům jako zprávy, protože už zde může být více adresátů, než jen hráč, který o akci požádal.

V bojové části se zmíněné požadavky provádějí voláním metody **PerformPlan** třídy **ServerPlayer**, která přijímá jako parametry počátek a cíl akce, případně také předposlední souřadnice na cestě, pokud je to potřeba, což je například při akci útoku jednotky, kde je potřeba specifikovat nejen, odkud jednotka přijde a kam útočí, ale také políčko, odkud má útok pocházet.

V taktické části se pak jedná o metodu **PerformArmyPlan** třídy **ServerPlayer**, která přijímá jako parametry počátek a cíl akce.

9.5.4 Optimalizace a ukládání nalezených akcí

Pro akce, které vyžadují předem ověření, je užitečné dočasně ukládat nalezené akce, aby nedocházelo k opakovanému vyhodnocování. Nebylo by totiž dobré, aby se klient opakovaně dotazoval serveru na stejnou akci, vždy když uživatel pohne myší na stejné souřadnice. Například pokud by měl uživatel označenou armádu a pohnul kurzorem myši nad prázdné políčko, a tedy by klient požádal server o vyhodnocení akce „pohyb armádou“, následně pohnul kurzorem myši jinam a zase zpátky, tak by klient opět žádal server o vyhodnocení stejné akce. Proto se vyplatí akce dočasně ukládat. Akce však nejsou ukládány dlouho, ale pouze do potvrzení akce a následně je tato historie promazána, protože jinak by se muselo složitě pro všechny nalezené akce ověřovat, že provedená akce je nezměnila. Například při akci „pohyb armádou“ se mohou změnit akce, které zahrnují cestu přes novou pozici armády, a tedy už nemusí platit a podobně.

K tomuto ukládání slouží třída **Pather**. Ten obsahuje především generickou kolekci **Dictionary**, indexovanou řetězcovým klíčem, který vznikne spojením počátečních a koncových souřadnic akce, oddělených mezi sebou pomlčkou, případně ještě v případě akcí s třemi souřadnicemi obsahuje klíč na konci také třetí souřadnici, opět oddělenou pomlčkou. Hodnotou je potom instance třídy **PathEntry**, která obsahuje nalezenou akci.

Při hledání akce se klient nejprve podívá pomocí třídy **Pather**, zda už snad daná akce není nalezená. K tomu slouží metoda **Pather.GetPathState**, která jako parametry přijímá dvě respektive tři souřadnice akce a tuto odpověď vrací jako výčtový typ **PathStates**, který nabývá tří možných hodnot:

- **EntryFoundPathExists**
    - Pokud akce už byla hledaná a je korektní. Potom klient získá akci metodou **Pather.GetPath** s odpovídajícími souřadnicemi jako parametry.
- **EntryFoundPathDontExists**
    - Pokud akce už byla hledaná, ale není korektní, například pro uvažovanou armádu neexistuje na dané souřadnice cesta. Pak se není na co dotazovat.

- **EntryNotFound**
    - Pokud akce ještě nebyla hledaná. Jedině v tom případě se bude skutečně hledat, tedy klient požádá server o její nalezení. Pokud je nalezena, uloží se do kolekce nová instance třídy **PathEntry** a uloží se do ní tato akce. V opačném případě se také vloží třída **PathEntry**, ale bez akce. Díky tomu lze poznat, že akce byla už hledána, ale neexistuje.

Stejný princip je na straně serveru, který také používá **Pather** a pracuje s ním stejně jako klient zde, samozřejmě však v případě, že je potřeba akci skutečně hledat, pouze provede odpovídající volání.

9.6 Načítání herních dat a uložení dat v paměti

Herní data, jako jsou informace o jednotkách, definice kouzel a podobně jsou uložena a načítána hned po spuštění serveru do paměti ze souboru Units.xml. Zároveň je použito validační schéma napsané v jazyce **Xml Schema** a uložené v souboru **Units.xsd**. Díky použití validace vůči schématu tak odpadá většina ověřování korektnosti souboru při samotném načítání v programu. Některé věci je však přesto nutno při načítání souboru ověřovat, protože se buď nedají schématem zachytit, nebo se zachytit dají, ale zatím zachyceny nebyly, mohly by tedy ještě být časem doplněny. Jedná se například o jednoznačnost definic nebo odkazy na předešlé definice, jako je seznam jednotek k nákupu ve hradě, kde dané jednotky musí samozřejmě být předem nadefinované.

Pro načítání je použit DOM, protože kód funkce pro načítání dat je potom dle mého názoru mnohem přehlednější, ačkoliv vzhledem k tomu, že se databáze pouze jednou při spuštění sekvenčně čte, by stačil i SAX. Jedná se ale o malou databázi a proto jsem tento fakt, že se celý strom dokumentu načte do paměti, nepovažoval za příliš omezující.

Na ukládání dat v paměti slouží třída **LogicDatabase**, která představuje celou herní databázi. Skládá se z několika hašovacích tabulek, konkrétně generické kolekce **Dictionary**, které obsahují odpovídající herní data a jsou indexovány řetězcem jako jménem herního objektu. Databáze obsahuje tabulky pro různé

kategorie herních objektů. Databáze také obsahuje informaci o ceně za nákup velitele armády.

Databáze jsou ve skutečnosti dvě, jednu si vytvoří server a jednu klient, aby se klient nemusel pořád zbytečně serveru dotazovat na definice jednotlivých dat.

9.6.1 Načítání mapy

Mapy jsou uloženy v XML souborech v adresáři Maps, vždy jeden pro každou mapu. Současně jsou v souboru Maps.xml stručné informace o těchto mapách, jako velikost a hlavně počet hráčů, aby tyto informace byly k dispozici před výběrem konkrétní hry a nemusely se kvůli tomu procházet a otevírat všechny soubory s mapami. Například už před načtením hry je totiž potřeba vědět počet hráčů ve hře, aby na ně server mohl před zahájením hry počkat a nemusel jen kvůli této informaci otevírat skutečný soubor s mapou.

9.7 Uživatelské rozhraní

Uživatelské rozhraní je vytvořeno použitím knihovny **Windows Forms**. Na zapouzdření téměř veškerého zobrazování je napsána vlastní abstraktní třída **GameLayout** a další od ní odvození třídy. **GameLayout** dědí od **Windows Forms** třídy **System::Windows::Form::Panel** a díky tomu je možné ho přidat do seznamu **Controls** aktivního okna a na tento panel pomocí funkcí skutečně zobrazovat potřebné informace. Od **GameLayout** se pak dále odvozují třídy **TacticLayout** pro taktickou část, **CastleLayout** pro hradní část, **BattleLayout** pro bojovou část, **DialogLayoutClass** pro herní dialogy, **ColumnLayout** pro layout používaný pro zobrazování podrobností o určité armádě či přesouvání jednotek v této armádě, dále od něj dědí **DialogLayoutClass** a **CastleLayout**, protože oba mohou využívat zobrazení určité armády.

Jednotlivé layouty pak sdružují potřebné metody a data pro odpovídající herní části, respektive pro herní dialogy.

9.7.1 Použití existujících obrázků

Hra používá pro téměř veškerou herní grafiku volně šiřitelné obrázky stažené z (21). Obrázky velitelů armád a některé kurzory jsou ze hry Battle for Wesnoth (8). Jde o kurzory pohybu armády (bota), obsazení dolu (lesní roh),

útoku (meč) a střílení (šíp). Kurzor hradu a také spojení (šipky) jsou ze hry Heroes of Might and Magic 2, ale nebyl by určitě problém je případně nahradit vlastními.

9.8 Klientský hráč

Klientský hráč slouží jako protějšek serverového hráče. Pomocí uživatelského rozhraní přijímá herní požadavky a ty po ověření přeposílá na server a čeká na odpověď, případně přijímá zaslané zprávy.

Protože na jednom klientském počítači ale může ve skutečnosti hrát více hráčů, je potřeba oddělit klienta od hráče. Proto je na straně klienta objekt třídy **ClientCommunicator**, který zastřešuje komunikaci se serverem, co se týče připojování a odpojování k serveru a registrace či odpojení hráče od hry. Také přijímá od serveru zaslané zprávy, které dále zpracovává, nebo přepošle klientskému hráči. Pro tuto komunikaci používá objekt třídy **RemoteClient**, který získal po připojení k serveru, kde se registroval ke vzdálenému objektu **RemoteServer**.

Nestará se však o zasílání herních požadavků, to je v kompetenci jednotlivých klientských hráčů, kteří jsou realizováni třídou **ClientPlayer**. Po registraci na serveru získají proxy na vzdálený objekt třídy **ServerPlayer**, který je uložený na serveru a pomocí něj pak provádějí herní požadavky, jak již bylo popsáno dříve.

9.9 Implementace AI: Principy

Hlavní myšlenky fungování umělé inteligence jsou popsané už v její analýze a mělo by tedy být možné získat základní představu o jejích principech už ze zmíněné části práce. Jsou zde tedy popsány pouze základní principy, protože podrobný popis by vyžadoval mnohem větší prostor.

Umělá inteligence je implementovaná hlavně ve třídě **AIPlayer**, která rozšiřuje třídu **ServerPlayer**. Díky tomu už není potřeba zasahovat do kódu pro herní logiku a implementace počítačového hráče tak od něj zůstává oddělená. Třída používá dvě hlavní třídy, **GoalManager** a **AnalysisData**. Při obdržení zprávy od serveru při taktické části pouze přepošle zprávu své instanci **GoalManager**.

9.9.1 Vlákna

Protože .NET Remoting je v základu synchronní, musí klient vždy po zadaní požadavku čekat na zpracování na serveru. V případě čekání zpracování své herní akce to není problém, protože takový výpočet je rychlý a čekání krátké. Opačná situace by mohla nastat při ukončení tahu. V takové chvíli herní server ukončí hráčův tah a pokud je nyní na tahu AI, provede její přemýšlení, tedy vyhodnocování. Protože ale při vyhodnocování vždy po provedení jednoho úkolu musí AI znovu úkoly vyhodnotit, aby opět našla ten nejlepší, musel by hráč čekat, dokud by AI nenašla celou posloupnost úkolů pro všechny volné prostředky a teprve pak by měl možnost zpracovat odpověď. Musel by potom čekat už o něco delší dobu.

Proto se hodí používat pro přemýšlení AI další vlákno. Díky tomu klient pouze ohlásí ukončení tahu, server zpracuje toto ukončení a podívá se, jestli teď není pro změnu na tahu nějaký AI hráč. Pokud ano, tak zašle tomuto AI hráči herní odpověď **StartThinkingGR**. Po jejím obdržení hráč hned uvědomí hru pomocí události **OnGetTurn**, kde jí předá odkaz na sebe. Hra potom ověří, zda už není nějaké vlákno pro AI vytvořeno a podle toho volá svou metodu **StartThinking** synchronně, nebo naopak asynchronně pomocí asynchronního volání delegáta. Pokud byl totiž předchozí aktivní hráč řízený počítačem, má hra jedno vlákno už vytvořené a proto není nutné vytvářet další, protože účelem vlákna bylo jen to, aby klient nečekal na celé zpracování počítačového hráče. Nepotřebujeme ale už, aby také jednotliví AI hráči nečekali na sebe navzájem, což je důsledek skutečnosti, že hra je tahová a proto musí počítačový hráč stejně jako lidský čekat na skončení tahu všech ostatních hráčů a přitom on sám žádné animace nevykresluje, takže skutečnost, že se dozví až konečné výsledky, mu vůbec nevadí.

Ať už je volané synchronně nebo asynchronně, **StartThinking** pouze provede volání **AIPlayer.StartThinking** pro zadaného AI hráče, čímž začne pro tohoto hráče přemýšlení.

Díky tomu může v tuto chvíli skončit zpracování požadavku „ukončení tahu“ a klient je opět aktivní. Zatím ale na serveru probíhá na novém vlákně výpočet AI. Po každém nalezeném úkolu a hned po provedení jedné jeho akce nyní hra

pošle odpověď klientovi, který díky tomu může tuto odpověď hned zpracovat a například provést animaci pohybu armády. Během tohoto vykreslování pak server stále počítá tah a proto klient musí čekat mnohem kratší dobu, protože obvykle už během průběhu animace má server dávno spočítané další výsledky. Navíc klient může například posouvat mapu nebo by mohl dělat případné jiné možné akce, na které má právo během tohoto čekání, třeba dívat se na stav svých armád nebo měst a podobně.

Bez použití takového vlákna by sice vzhledem k použitému návrhu klient také hned obdržel odpověď od serveru, avšak byl by z důvodu zpracovávání svého požadavku „ukončení tahu“ na serveru po celou tuto dobu zablokovaný, dokud by hra neskončila jeho tah tím, že by pro všechny AI hráče provedla výpočet.

Po skončení tahu všech AI hráčů se vytvořené vlákno ukončí a na tahu je opět lidský hráč.

Poznamenejme ještě, proč vůbec hra zasílá odpověď **StartThinking** a čeká na událost místo toho, aby hned volala metodu **AIPlayer.StartThinking**. Událost je potřeba proto, že hra může mít ve chvíli skončení hráčova tahu stále ještě ve frontě odpovědí k zaslání další nezpracované odpovědi. Počítačový hráč sice dostává odpovědi synchronně, ale přesto je lepší použít pro tuto situaci také metodu zasílání odpovědí, díky čemuž je jasně patrné, že přemýšlení AI čeká správně ve frontě a provede se až v pravou chvíli. Navíc pokud bychom snad z nějakého důvodu přešli na asynchronní posílání odpovědí pro AI hráče, bude přesto tento způsob pracovat správně.

9.10 Implementace AI: Bojová část

Pro bojovou část využívá třídu **AnalysisData** na provedení analýzy souboje dle zásad uvedených v části této práce zabývající se analýzou. Díky provedené analýze pak vybere příslušnou akci pomocí if-then pravidel dotazováním na **AnalysisData**. Tato třída obsahuje metody pro dotazování na splnění pravidel popsaných v analýze, jako například **GetStrongestEnemyInRange**, která vrátí nejsilnějšího nepřítele v dosahu, nebo také **GetStrongestShooter** pro získání nejsilnějšího střelce a podobně. If-then pravidly tedy nakonec AI vybere jednu z možností a tu následně vykoná. **AIPlayer** obsahuje tři posloupnosti pravidel.

Jednu pro střeleckou jednotku, jeden pro jednotku s útokem na blízko a jednu pro velitele armády. Každá z těchto jednotek totiž má trochu odlišné chování.

9.11 Implementace AI: Strategická část

Implementace ve strategické části je mnohem složitější, než v části soubojové, což je samozřejmě dáno mnohem větší složitostí rozhodování v této části.

9.11.1 Hlavní třídy pro správu úkolů

GoalManager

Hlavní třídou je zmíněná třída **GoalManager**, která implementuje spráce cílů popsaného v analýze. **AIPlayer** mu přeposílá obdržené zprávy a **GoalManager** sám určuje, jaká akce se dále provede a podle toho případně sám požádá hru o její provedení, protože má vlastní odkaz na třídu **TacticPart**. Zde by bylo lepší, kdyby **GoalManager** mohl provádět akce stejně jako lidský hráč pouze přes svou třídu **AIPlayer**, v současném stavu však využívá metody **TacticPart** přímo, což ale nijak nenarušuje jeho fungování.

Mezi nejhlavnější metody **GoalManageru** patří metody **EvaluateGoals**, pro vyhodnocení cílů, **GenerateTasks** pro generování úkolů pro cíle, **SortTasks** pro třídění úkolů a **ExecuteTasks** pro provedení nejlepšího úkolu. Všechny tyto metody pak využívá metoda **FindAndExecAction**, která podle popsaného postupu provede proces od vyhodnocení až po provedení úkolu.

SharingBoard

Třída **SharingBoard** představuje sdílenou tabuli popsanou v analýze. Každý GoalManager má svou instanci **SharingBoard**. Tato třída také sdružuje cíle, úkoly, armádní plánovače, mapu vlivu, hradní a armádní ohodnocení a potřebu surovin. Všechny cíle pak tuto tabuli mohou využívat pro dotazování a podle toho provést své patřičně ohodnocení.

Mezi nejhlavnější metody patří **GetBrigadeScheduler** pro získání armádního plánovače, **GetCastleEvaluation** pro získání celkového hradního vyhodnocení, **GetBrigadeEvaluation** pro získání armádního vyhodnocení, dále metody pro sdílenou analýzu, jako **GetArmySingleForce** pro získání síly jedné zadané

armády a take **GetArmyGlobalForce** pro získání síly zadané armády spolu s vlivem pro ni přátelských armád.

9.11.2 Reprezentace cílů

Všechny cíle jsou odvozené od třídy **BasicGoal**. Jedná se o abstraktní třídu, která obsahuje převážně vlastnost **BasePriority**, určující základní ohodnocení cíle a také pak metody **GenerateTask** pro generování úkolů a **Evaluate** pro vyhodnocení. Také obsahuje přiřazený úkol **AssignedTask** a poslední aktivní úkol, pod identifikátorem **LastTask**. Další důležitou metodou je **Valid**, která ověří, zda je daný cíl vůbec pro danou situaci relevantní a **CheckAssigment**, která ověřuje, zda už cíl nebyl přiřazen nějakému úkolu a tedy nemá cenu jej vyhodnocovat. Dále obsahuje pole **tasks** se všemi podúkoly, které používá při generování svých úkolů. Podstatnou součístí je také odkaz na sdílenou tabuli, kterou při práci využívá.

9.11.3 Reprezentace úkolů

Všechny úkoly jsou odvozeny od základní abstraktní třídy **BasicTask**. Třída obsahuje především odpovídající **BasicGoal**, ke kterému se váže. Mezi hlavní metody patří **Execute** a **ExecuteAgain**. První slouží k počátečnímu provedení úkolu, druhá slouží k pokračování úkolu po jeho přerušení, kde parametrem je právě herní odpověď na zadanou akci, kvůli které úkol čekal. Podstatnou součástí je také událost **OnFinish**, pomocí které signalizuje přerušení aktuální práce a **ChangeStateAndFinish**, která změní stav úkolu a ohlásí přerušení.

Pro přidělování úkolu slouží metody **Assign** a **Unassign**. Jejich funkcí je pro odpovídající cíl provést svázání s tímto úkolem, díky čemuž pak tento cíl ví, že při vyhodnocování a generování nemá generovat další úkoly.

9.11.4 Reprezentace hradních a armádních vyhodnocení

Hradní a armádní vyhodnocení jsou reprezentována třídami **CastleGlobalEval** a **BrigadeGlobalEval**. **CastleGlovalEval** obsahuje mimo jiné také seznam pro vyhodnocení armád vzhledem k tomuto hradu, což je třída **CastleSingleEval**. Ta obsahuje informace o nebezpečnosti armády pro hrad, vzdálenost, armádu a další potřebné informace. Stejně tak **BrigadeGlobalEval** obsahuje seznam na vyhodnocení hradů **CastleGlobalEval**, díky čemuž potom může při aktualizaci

armády rychle najít vyhodnocení hradu a upravit také patřičně jeho hodnocení po změně armády.

9.11.5 Reprezentace armádního plánovače

Armádní plánovač je reprezentován třídou BrigadeScheduler a implementuje vlastnosti popsané v analýze. Hlavními položkami jsou odkaz na přidruženou armádu, armádě přiřazený úkol **AssignedTask** a pak seznam vygenerovaných úkolů **PossibleTasks**, díky kterému může například cíl „obrana hradu“ nalézt úkoly armády a zvedat jim podle potřeby prioritu.

9.11.6 Reprezentace spojení armád

Pro spojení armád je potřeba vyhodnotit, jak bude vlastně spojení vypadat, protože silnější armáda dostane lepší jednotky a slabší armáda si nechá zbytek. Zároveň však chceme, aby slabší armáda měla dostatek jednotek. Navíc síla armády po spojení určuje prioritu takového úkolu. K tomu slouží právě třída **RecruitionDetails**, která však také může sdružovat informace o rekrutování jednotek, protože často jsou akce rekrutování a spojení spojené. Po spočítání spojení se po případném výběru úkolu zašle takové spojení herní logice, která pomocí něj rozdělí jednotky mezi dvě armády, nebo třeba mezi hrad a armádu při nakupování jednotek. Spojení se provádí metodou **PerformJoinSwapPlan**, rekrutování **PerformRecruitionPlan**. Obě metody dostávají v parametru právě spojení **RecruitionDetails.**

Po spojení AI také rozdělí jednotky v armádě, pokud je to možné, čím sníží zranitelnost takové armády a navíc v případě střelců získá možnost více útoků, kdy je užitečné, aby silný střelec nevyplýtval svůj útok na slabší nepřátelskou jednotku, ale aby v takovém případě jednotka dostala zásah pouze od menší části. Další rozdělené části pak mohou buď opět zaútočit na stejný cíl, pokud byl silný, nebo vybrat jiný a tím zranění rozdělit.

10 **Závěr**

Cílem práce bylo naprogramovat hru v základních principech inspirovanou herní sérií Heroes of Migt and Magic a pokusit se tak proniknout do některých problémů s tím spojených. V tomto ohledu se domnívám, že práce splnila svůj účel. HINT implementuje základní herní principy a umožňuje hrát hru jak více hráčů, tak hru proti počítačem řízenému hráči. K tomu poskytuje jednoduchou herní grafiku a snad také pohodlné a intuitivní uživatelské rozhraní a možnosti ukládání a načítání her.

Součástí práce byla také analýza umělé inteligence pro rozhodování AI hráče. Tato analýza mi velmi pomohla ujasnit si, co je konkrétně potřeba při takovém vývoji brát v úvahu a jakými problémy se zabývat. Teoretická část práce, kde se zabývám některými možnými rozhodovacími technikami pro řešení AI mi dala mnoho podnětů pro výběr a implementaci použitého řešení. Pro bojovou část se to týká především vyhodnocování pravidel podle vzoru jednoduchého rozhodovacího stromu, pro taktickou část pak hlavně Goal-oriented behavior a architektura typu tabule. Také mě inspirovala generická implementace při konečných automatech, kdy má každý stav své chování, kde jsem myšlenku částečně použil pro práci cílů a úkolů, čímž se problém velmi dekomponoval.

Domnívám se, že vytvořená AI funguje poměrně rozumně, ačkoliv by zde jistě bylo mnoho prostoru pro další vylepšení. Řešení pro bojovou část představuje kompromis mezi rychlostí a „chytrostí“ a díky vybranému řešení je možné jej použít také pro vyhodnocování tahů čistě jen počítačových hráčů. Řešení pro taktickou část by uvítalo důkladné vyvážení priorit pro rozhodování v rámci jednotlivých cílů, aby AI občas nevykazovala zvláštnosti v chování vlivem nadhodnocení, nebo naopak podhodnocení určitého úkolu, ale domnívám se, že z tohoho ohledu se jedná spíše o část testování, než programování.

Vzhledem k poměrně oddělenému kódu pro umělou inteligenci a herní logiku by hra mohla případně posloužit jako základní platforma, kde by po přidání dalších modulů, které by implementovaly jiný přístup pro řešení AI, bylo

zajímavé sledovat, jak který přístup dokáže řešit předložené problémy, které se jistě vyskytují v mnoha dalších hrách, nejen tahových strategiích.

10.1 Možná budoucí vylepšení

V oblasti umělé inteligence by bylo dobré rozšířit její schopnosti v používání kouzel, kdy by AI dokázala z externích dat porozumnět kouzlům a podle toho je sesílat co možná nejlépe. V taktické části by zase bylo možné udělat systém na ukládání dříve vyhodnocených úkolů a cílů, kde je potřeba velké opatrnosti kvůli kolizi mezi jednotlivými úkoly, kdy provedení jednoho úkolu může rušit platnost jiného a proto nelze jednoduše nalézt řešení jen jednou v každém tahu a pak najednou provést.

Další vylepšení se týká grafického rozhraní. V současné době používá hra pro vlastní vykreslování GDI+, které však svým výkonem není nejsilnější, ačkoliv v současné době postačuje. Nynější návrh grafického rozhraní je do značné míry uzpůsoben tak, že používá mnoho vlastních tříd pro správu animací či udržování herních rozvržení (layoutů), takže by neměl být při určitém úsílí až tak velký problém změnit použitou knihovnu například na DirectX.

11 Reference

1. Heroes of Might and Magic. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Heroes_of_Might_and_Magic.

2. Might and Magic. *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Might_and_Magic.

3. Might Heroes - Heroes of Might and Magic 1. *Age of Heroes.* [Online] http://www.heroesofmightandmagic.com/heroes1/mightheroes.shtml.

4. Magic Heroes - Heroes of Might and Magic 1. *Age of Heroes.* [Online] http://www.heroesofmightandmagic.com/heroes1/magicheroes.shtml.

5. Primary Skills - Heroes of Might And Magic 2. *Age of Heroes.* [Online] http://www.heroesofmightandmagic.com/heroes2/primaryskills.shtml.

6. Secondary Skills - Heroes of Might And Magic 2. *Age of Heroes.* [Online] http://www.heroesofmightandmagic.com/heroes2/secondaryskills.shtml.

7. Free Heroes2 Engine. *SourceForge.* [Online] http://sourceforge.net/projects/fheroes2/.

8. *Battle For Wesnoth.* [Online] http://www.wesnoth.org/.

9. Civilization (Series). *Wikipedia, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Civilization_(series).

10. Fantasy General. *Wikipedie, the free encyclopedia.* [Online] http://en.wikipedia.org/wiki/Fantasy_General.

11. **Dill, Kevin.** Prioritizing Actions in Goal-Based RTS AI. [autor knihy] Steve Rabin. *AI Game Programming Wisdom 3.* Boston : Charles River Media, Inc., 2004.

12. **Millington, Ian.** *Artificial Intelligence for Games.* San Francisco : Morgan Kaufmann Publishers, 2006. ISBN 0-12-497782-0.

13. **Fu, Dan a Houlette, Ryan.** The Ultimate Guide to FSMs in Games. [autor knihy] Steve Rabin. *AI Game Programming Wisdom 2.* Hingham : Charles River Media, Inc., 2004.

14. **Rabin, Steve.** *AI Game Programming Wisdom 2.* Hingham : Charles River Media, Inc., 2004. ISBN 1-58450-289-4.

15. **Champandard, Alex.** *AI Game Development.* Berkeley : New Riders Publishing, 2004. ISBN 1-5927-3004-3.

16. **Freeman-Hargis, James.** Rule-Based Systems and Identification Trees. *AI Depot.* [Online] 23. 11 2002. http://ai-depot.com/Tutorial/RuleBased-Methods.html.

17. **O'Brien, John.** A Flexible Goal-Based Planning Architecture. [autor knihy] Steve Rabin. *AI Game Programming Wisdom.* Hingham : Charles River Media, Inc, 2002.

18. **Russel, Stuart a Norvig, Peter.** *Artificial Intelligence - A Modern Approach.* 2. vydání. Upper Saddle River : Prentice Hall, 2003. ISBN 0-13-080302-2.

19. **Orkin, Jeff.** Applying Goal-Oriented Action Planning to Games. [autor knihy] Steve Rabin. *AI Game Programming Wisdom 2.* Hingham : Charles River Media, Inc, 2004.

20. **Ono, Tom.** *Heroes of Might and Magic III - Oficiální příručka strategie.* Brno : Prima Publishing; Stuare, s.r.o, 1999. ISBN: 80-238-3940-3.

21. **Prokein, Reiner.** Reiner`s Tilesets. [Online] http://reinerstileset.4players.de/englisch.html.

12 Dodatky

Příloha A: Obsah přiloženého CD

- **Docs**
  - **Addendum AI.pdf**
    - Elektronický dodatek se stručným popisem, jak AI vyhodnocuje jednotlivé cíle
  - **Bachelor Thesis.pdf**
    - Bakalářská práce v elektronické podobě ve formátu pdf
  - **Manual.pdf**
    - Uživatelská dokumentace ke hře
- **Bitmaps**
  - Adresář se všemi obrázky ve hře. Program pak však používá seskupené obrázky z resource souboru, tyto obrázky představují pouze původní zdroj.
- **Program**
  - **Bin**
    - **Animations.dll**
      - Knihovna pro animace ve hře
    - **ClientApp.exe**
      - Spustitelná aplikace představující klienta
    - **Loader.exe**
      - Spustitelná aplikace pro spojení všech obrázků do resource souboru.
    - **ResLib.dll**

        - Knihovna obsahující resource soubor s obrázky
    - **ServerApp.exe**
        - Spustitelná aplikace představující server
- **Maps**
    - Adresář s XML soubory představující mapy pro hru
- **Saves**
    - Adresář s XML soubory představující uložené hry
- **Animations.xml**
    - XML soubor s definicemi dynamických obrázků ve hře
- **Connections.xml**
    - XML soubor s uloženými připojeními k serveru
- **Images.xml**
    - XML soubor s definicemi statických obrázků ve hře
- **Maps.xml**
    - XML soubor se seznamem map, které jsou pak uloženy v adresáři Maps
- **Saves.xml**
    - XML soubor se seznamem uložených her, které jsou pak v adresáři Saves
- **Units.xml**
    - XML soubor s definicemi většiny herních objektů

- **Sources**
    - Adresář se zdrojovými kódy programu

Příloha B: Uživatelská dokumentace

Uživatelská dokumentace je obsažena na přiloženém CD.

Příloha C: Dodatek k AI

Dodatek stručně popisující, jak AI vyhodnocuje cíle, na přiloženém CD.