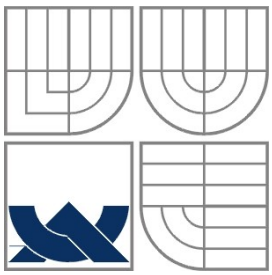
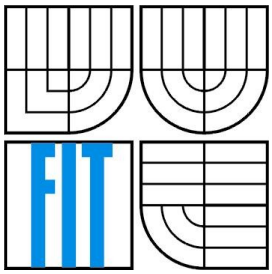


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

STRATEGICKÁ HRA S INTELIGENTNÍMI PRVKY

STRATEGY GAME WITH INTELLIGENT ELEMENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ SNÁŠEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. František Zbořil, Ph.D.

BRNO 2008

Abstrakt

Cílem práce bylo navrhnout samofungující systém, který na základě teorie her a multiagentních systémů bude schopen dosáhnout svých cílů. Prezentací tohoto systému je strategická bojová tahová hra s grafikou vykreslenou pomocí DirectX9 a implementovaná v jazyce C++. Hlavním cílem bylo navrhnout vhodné prohledávací, plánovací a rozhodovací algoritmy.

Klíčová slova

Umělá inteligence, Multiagentní systém, teorie her, strategická bojová hra, C++, objektový model, simulace, rozhodovací algoritmy, plánovací algoritmy, sjednávání závazků, vytváření koalic.

Abstract

The aim of this thesis is to design a self-functional system, which will be capable to reach its own goals by taking advance of the game theory and the multiagent systems. As a presentation of this system there is a strategical fighting game with graphics drawn by DirectX9 and implemented in C++ programming language. The main goal was to design suitable search, schedule and decision-making algorithms

Keywords

Artificial intelligence, Multiagent system, game theory, strategical fighting game, C++, object model, simulation, decision-making algorithms, scheduling algorithm, negotiation of obligations, alliance creation

Citace

Lukáš Snášel: Strategická hra s inteligentními prvky, bakalářská práce, Brno, FIT VUT v Brně, 2008

Strategická hra s inteligentními prvky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Františka Zbořila, Ph. D.
Další informace mi poskytl Bc. Honza Zelený.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Snášel
27.4.2008

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce, panu Ing. Františkovi Zbořilovi Ph. D., za odbornou pomoc při konzultacích a možnost pracovat na této bakalářské práci. Dále bych rád poděkoval mému kamarádovi Bc. Honzovi Zelenému, za poskytnutí svého grafického enginu a testování aplikace.

Obsah

1 Úvod

V současné době je plno různých samostatně fungujících systémů od základního jako je systém samotné přírody, lidstva, přes automatické systémy strojů a mechanismů (automatické řízení křižovatky pomocí semaforů a senzorů reagující na frekventovanost dopravy), počítačových her s lepší a lepší inteligencí až po samo-jezdící vozítka, které nám pomáhají v situacích, kde člověk nemůže či je pro něj situace moc nebezpečná. Rozhodl jsem se prozkoumat takový samofungující systém i z té vnitřní strany a tedy podobný systém vytvořit, i když zdaleka ne v takovém měřítku, jako výše uvedené systémy.

1.1 Cíl práce

Cílem mého snažení je vytvořit fungující inteligentní systém, který se bude sám rozhodovat dle situace a svých cílů, reagovat na okolní podněty a tím vším dosáhnout svého cíle. Jako prezentaci výsledku jsem zvolil strategickou tahovou bojovou hru, kde se dva tábory zastupující dva inteligentní podsystémy snaží dostat do svého tábora všechny vlajky na mapě. Hra spolu s grafikou je v duchu středověké doby.

Dalším cílem je poukázání vnímání systémů, her a simulací v širším měřítku jako spojení více vědních oborů.

2 O hrách obecně

Nejprve je nutné vysvětlit pojem hra a přiblížit můj pohled na význam hry obecně. Na hry se můžeme podívat hned z několika hledisek a kritérií jako je psychologie, filozofie(kultura), z hlediska informatiky a vědy. Záleží také na tom, zda ji vnímáme jako autor a tvůrce, nebo její hráč a uživatel.

2.1 Filozofie her

Co je to vlastně *hra*? Hra je něco jako iluze a náhrada skutečnosti. Nepatří sem pouze deskové, karetní, počítačové, společenské, divadelní hry či soutěže, které většinou všichni známe. Hry různého typu jsou známé už od starověku. V různých podobách se dochovaly až do dnešní doby (např Antické olympijské hry[5], které se konaly již v 7. století př. n. l.). I básně jsou jistou formou her – her se slovy (taktéž už od 7. století př. n. l., nejvíce však v období romantismu a renesance). Z dnešní moderní doby je taková simulace ať virtuální počítačová či jakákoliv jiná náhradou skutečnosti, tedy i hrou.

Hry jsou v každém oboru, ať si na něco či někoho hrajeme, či si hrajeme se slovy, s čísly, s city a pocity, s barvami, atd.

Se hrou se také pojí otázky jako: „Proč hry existují?“, „Proč je tvoříme?“ a v neposlední řadě „Proč hry využíváme ve svém životě, k čemu nám jsou?“ Dalo by se tyto otázky rozvést do větších podrobností, ale protože nejsou hlavní náplní této práce, postačí prozatím stručný výklad v následujících kapitolách.

2.2 Psychologie her

Už od pradávna lidstvo (a nejen lidstvo ale živočichové obecně), vymýšlí, tvoří a využívá hry různého zaměření. Tvoří je snad pro pobavení, protože se v životě nudí? Ne. Hlavní záměr je jiný a to zdokonalit své schopnosti a dovednosti popřípadě někoho zaujmout. Teprve později začal člověk vnímat hru jako zábavu. Staří Řekové si dokazovali sílu a své bojové dovednosti, stejnou funkci měly i *Rytířské* turnaje[6] ve středověku, kde v počátcích pro četná zranění a úmrtí se blížili spíše k samotné válce než k turnaji. Sloužily však hlavně pro zlepšení a osvojení bojových dovedností či k zaujetí bohatého pána nebo srdce vzácné paní. Rytířské turnaje a klání pomáhaly k utváření rytířské mentality a morálky - dodržení daného slova, odvaha, péče o dobrou pověst. Až později, kdy se turnaje staly bezpečnější, se zbraně začaly používat pro zábavu a z těchto turnajů se stala tradice. I při slavnostech(nejčastěji v náboženství), kde nepřevládala síla ale mysl, mezi sebou lidé soutěžili. Zde různými cenami byli odměňováni zase autoři, zpěváci, tanečníci atd.

Doba se mění a s ní i mentalita a důvody. Na rozdíl od zvířat, která využívají hry jako součást rozvoje sama sebe a svých dovedností, jako tomu bylo před tisíciletími a bude tomu i nadále, člověk využívá hry spíše k nahrazení reality(např. virtuální světy), k zisku peněz (profesionální hraní PC her, soutěže, sport, atd.). Tím více se vnímá hra jako zábava a zapomíná se na původní záměr. Horší případ je když se zapomíná i na tu zábavu (doping ve sportu). Ale i přesto mají hry další velmi pozitivní díl v dnešní době a to ve vědě.

2.3 Věda a hry

I přesto jak jsou od sebe různé vědy jakkoliv vzdálené či naopak blízké, všechny vědy a vědci mají jedno společné. Je to nadšení a zápal pro danou věc. Když se podíváme do historie, většinu objevů a vynálezů vytvořili spíše muži než ženy. Vychází to z rozdílné povahy mužů a žen a z hravé povahy dítěte, které dokáže se zaujetím a nadšením vydržet u dané věci dlouho. Postupně s věkem získává trpělivost a poznává jaký obor či zaměření ho v životě naplňuje (co ho baví). Nejen že díky vědě a touze posunout vědu zas o kousek dál a této trpělivosti, létáme pomocí raketoplánů do vesmíru a

dalekohledy zkoumáme hodně vzdálené planety a hvězdy, ale dokážeme pomocí barevných LCD displayích a mikro-technologie zkoumat a rozeznat buňky a atomy. Nebo běžnější věci jako žárovka, automobil či letadlo, rozdělení krevních skupin, zkoumání přírodních jevů či katastrof, zkoumání života jednotlivých živočichů pomocí kamer a různých technologií(zoom, noční vidění, termovize), ekologičtější výroba energie či její úspornější spotřeba, atd. Ve vědě jsme ale už tak daleko, že nám nestačí finance, schopnosti či jen možnost některý experiment provést. V takovýchto případech využíváme softwarového inženýrství konkrétně virtuální simulace k měření a provádění experimentů. Spolu se simulací se pojí ještě teorie her o které si povíme v kapitole 3.

2.4 Hra v informatice

Spojením her a softwarového inženýrství vznikaly počítačové hry a videohry(na herních konzolích)[4]. Dělí se do různých kategorií. Podle žánru na adventury, akční hry, arkády, strategie, simulátory a RPG(hry na hrdiny). Tyto hry se dají hrát na jednom počítači, po LAN síti(místní síť) a nebo po internetu(online hry). Z větší či menší části nahrazují realitu a umožňují hráči zkusit či „prožít“, co by v reálném světě nemohli. Příkladem jsou hry zaměřené na historii a její pravdivé ztvárnění (např. Civilizace), simulátory (letadel, aut, formulí, vlaků) či bojové, budovatelské a ekonomické strategie, kde teorie her hraje také významnou roli.

Dalším využitím jsou simulace, které mají ke hrám blízko. Tak jako hra nahrazuje realitu, je simulace přesněji řečeno: experimentování se zjednodušeným modelem reality. Čím přesnější výsledky vyžadujeme, tím složitější a přesnější musí model reality být. Zjednodušený model neuvažuje všechny vlivy a parametry, jedná se převážně o parametry, vzhledem k požadovanému výsledku, nedůležité a zanedbatelné.

2.5 Královská hra Šachy

A čím toto vše začalo? Jsou to šachy a jejich pomalejší předchůdce šatrandž (kterému se tehdy ovšem v Evropě říkalo rovněž šachy). Už ve středověku a renesanci byly součástí šlechtické kultury, kde byly používány pro výuku válečné strategie a nazývány „královskou hrou“. Mezi šlechtou byly považovány za jedno ze sedmera umění, které by měl ovládnout každý dospívající rytíř (vedle plavání, veršování, jízdy na koni, střelby z kuše, šermu a lovu). Nádherné šachové soupravy používané aristokracií té doby jsou většinou už ztraceny, ale některé ze zbývajících kusů, jako jsou Lewisovy šachové figurky z 12. století, vykazují vysokou uměleckou hodnotu.

Na druhou stranu však mnohde politické a náboženské autority šachy zakazovaly jako projev nemravnosti či hráčské vášně. Například Jan Hus prý šachy hrával v mládí, později se ale stal jejich

odpůrcem, neboť se domníval, že se tím promarní spousta času, který by člověk mohl strávit v trpělivé práci nebo modlitbách; rovněž mu vadilo, že šachy mohou vyvolat špatné pocity vůči protivníkovi. Hraní šachu, ale především karetních her, kritizoval také Petr Chelčický.

Odedávna se šachové prvky používají pro svou dekorativnost také ve výtvarném a užitném umění. Příkladem může být šachovaná orlice na moravském zemském znaku. O všeobecné oblibě šachů ve středověku svědčí také fakt, že už v 15. století přešel výraz šachovaný jako vzor látky do soukenické terminologie. Slovo šašek vzniklo právě jako označení muže, nosícího pestře šachované oblečení.

V osvícenství se šachy jevily především jako způsob, jak rozvíjet sebe sama. Benjamin Franklin napsal v článku The Morals of Chess (Morálka šachů, 1750):

> *Šachová hra není pouze zábavou zahalečů; její pomocí lze získat nebo posílit několik velmi cenných schopností mysli, které se pak stanou návyky použitelnými při každé příležitosti. Vždyť život je druh šachů, v němž často máme získávat body, soupeřit proti konkurentům a protivníkům a v němž nastává široká škála dobrých i špatných událostí, jež jsou do jisté míry následky lidské prozíravosti nebo jejího nedostatku. Hraním šachů se tedy můžeme naučit:*
>
> 1. *Předvídavosti, jež hledí do blízké budoucnosti a zvažuje následky, jaké může přinést nějaká akce…*
> 2. *Pozornosti, jež zkoumá celou šachovnici čili scénu akce: vzájemný poměr několika figur a jejich situaci…*
> 3. *Opatrnosti, nevykonávat tahy zbrkle…*

S podobnými nadějemi se šachům dnes učí děti ve školách celého světa a používají se v armádách jako mentální trénink pro kadety a důstojníky.

Obr. 1: Program CheesDuel[8]

Šachy jsou zajímavé i z hlediska matematiky. Už stovky let jsou známy mnohé kombinatorické a topologické problémy s nimi spojené. Ernst Zermelo je v roce 1913 použil jako oporu své teorie herních strategií, která je považována za jednu z předchůdkyň moderní *teorie her*. Nejdůležitějším matematickým problémem šachu bylo vytvořit algoritmus hrající šachy. Idea stroje hrajícího šachy vznikla už v 18. století; kolem roku 1769 se proslavil automat hrající šachy zvaný Turek, který však byl později odhalen jako podvod. První velký šachový turnaj pro počítače pořádala v září 1970 Americká Asociace pro výpočetní techniku (Association for Computing Machinery,

ACM), v němž zvítězil CHESS 3.0, šachový program z Northwestern University. Zatímco nejdříve byly šachové programy považovány za pouhé kuriozity, nyní jsou nejlepší z nich, například Rybka nebo Hydra, mimořádně silnými hráči. Už v roce 1997 prohrál Garri Kasparov, tehdy první hráč světového šachového žebříčku, zápas proti šachovému počítači firmy IBM, neformálně nazývanému Deep Blue. Z hlediska teorie umělé inteligence však jsou šachové programy(obr. 1) relativně jednoduché: v zásadě prozkoumávají obrovské počty potenciálních budoucích tahů obou hráčů a na vznikající pozice aplikují vhodnou ohodnocovací funkci (pomocí algoritmu používaný pro hraní strategických her mezi dvěma a více hráči, kde principem algoritmu je procházení stromu hry a minimalizace maximálních možných ztrát minimax)[3].

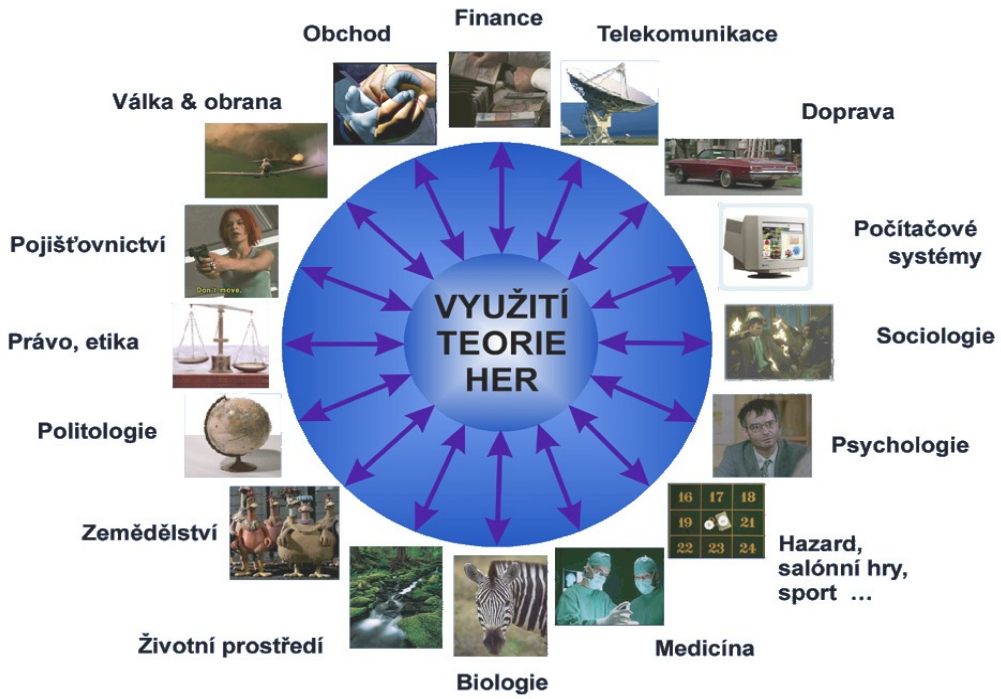
3 Teorie her

3.1 O co jde?

Každý z nás se prakticky den co den ocitá v situacích, kdy se musí rozhodnout pro vhodnou *strategii*, aby docílil – ze svého pohledu – co nejlepšího výsledku. Závisí-li tento výsledek jen na něm samém, případně na nějakých dalších vlivech, které lze s určitou pravděpodobností předvídat, ale které jsou nezávislé na rozhodnutí kohokoli jiného (například pád balvanu z pusté skály, na níž nikdo nestojí), pak je situace poměrně jasná. Méně zřejmé je rozhodování v situacích, kdy výsledek závisí ještě i na rozhodnutí nějaké další inteligentní bytosti – zde pak přichází ke slovu *teorie her*.

Náplní teorie her je jednoduše řečeno, analýza velmi širokého spektra rozhodovacích situací. Nejedná se přitom jen o to, čemu jsme zvyklí říkat *hra* – šachy, poker, hokej apod., ale především o konflikty mezi obchodními společnostmi, vojenskými jednotkami, stíhačkami, ponorkami, účastníky souboje, národy, politiky, politickými stranami, samci v říji, biologickými druhy, motoristy, uživateli počítačové sítě, majiteli téhož pozemku, milenci téže dámy, věřiteli, jejichž dlužník skončil bankrotem a zbylá částka nepokrývá všechny dluhy (obr. 2). Aby mohla být vytvořena jednotná teorie, budou se všechny tyto a mnohé další situace nazývat *hrami* a jejich účastníci *hráči*. Každá *hra* tak bude definována svými hráči, pravidly (pořadím rozhodování hráčů, možnými rozhodnutími ve všech krocích, stupněm informovanosti při rozhodování), vlivem rozhodnutí na výsledek a preferencemi jednotlivých výsledků u jednotlivých hráčů.

Cílem uvedené analýzy je jednak popis daných konfliktních situací a pochopení chování jednotlivých účastníků (to je důležité v případech, kdy má *hra* příliš komplikovaná pravidla a kdy v ní figuruje příliš mnoho účastníků), na druhé straně pak – a to je pro praxi mnohem zajímavější – podat *hráčům* návod, jakou mají zvolit strategii (ne vždy je možné určit *nejlepší* strategii, teorie her však alespoň poradí strategii *lepší*)[1].



Obr. 2: Příklady užití teorie her

3.2 Rozdělení

Teorie her se dělí dle následujících kategorií[1].

Podle rozhodování

- **racionální(inteligentní)**

  rozhodování je uvědomělé, zaměřené k určitému cíli, využívá všech objektivně dostupných informací o důsledcích voleb jednotlivých alternativ
- **neracionální(neinteligentní)**

  je lhostejné k důsledkům rozhodování; například působení prostředí(*svět*, *příroda*); alternativy neinteligentního rozhodování se obvykle nazývají *stavy*
- p **-inteligentní**

  s pravděpodobností p se chová jako inteligentní rozhodování a s pravděpodobností $1-p$ se chová jako neinteligentní

Podle počtu charakteristik

- **Monokriteriální rozhodování**
  výsledky jsou subjektem hodnoceny na základě jedné charakteristiky(kritéria)
- **Vícekriteriální rozhodování**
  výsledky jsou subjektem hodnoceny podle více charakteristik (kritérií)

Podle počtu účastníků

- Rozhodování s jedním racionálním účastníkem
- Rozhodování, jehož výsledek je ovlivněn alespoň dvěma racionálními účastníky

Individuální rozhodování podle rizika

- **Rozhodování za jistoty**
  každé rozhodnutí vede ke známému, jednoznačnému danému důsledku
- **Rozhodování za rizika**
  každé rozhodnutí vede k nějakému prvku z jisté množiny důsledků, přičemž jednotlivé důsledky nastanou se známou pravděpodobností
- **Rozhodování za neurčitosti**
  každé rozhodnutí vede k nějakému prvku z jisté množiny důsledků, přičemž jednotlivé důsledky nastanou s předem neznámou pravděpodobností

3.3 Zápis her

V teorii her jsou *hry* formálně definovanými pojmy. Hra obsahuje *hráče*, jejich možné *tahy* (nebo také *akce* či *strategie*) a *funkci* udávající zisk každého hráče v závislosti na provedených tazích. V literatuře se hry zapisují jedním ze dvou následujících způsobů[2].

3.3.1 Normální forma

Normální forma hry je většinou reprezentována *maticí*, která zobrazuje hráče, jejich možné strategie a možné zisky(Tab. 1). Obecněji může být reprezentována funkcí, která přiřazuje zisk každému hráči na základě dané kombinace tahů. U her v normální formě se předpokládá, že hráči vybírají tahy zároveň, nebo alespoň nevědí, který tah vybral protihráč. Pokud hráči mohou znát tahy protihráče, uvádí se hra většinou v extenzivní formě.

V příkladě jsou dva hráči. Úkolem prvního hráče je vybrat řádek, úkolem druhého je vybrat sloupec. Každý hráč má dvě možnosti. Zisky jsou zapsány uvnitř matice, první číslo určuje zisk pro

hráče 1, druhé určuje zisk pro hráče 2. Pokud tedy první hráč vybere *A* a druhý *X*, zisk prvního hráče je 4 a zisk druhého hráče je 3.

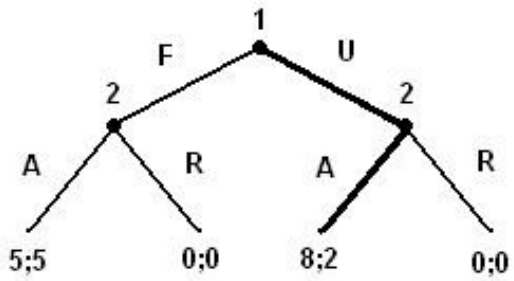
	Hráč 2 vybere X	Hráč 2 vybere Y
Hráč 1 vybere A	4,3	-1,1
Hráč 1 vybere B	0,0	3,4

Tab. 1: Hra dvou hráčů v normální formě

3.3.2 Extenzivní forma

Extenzivní forma hry bývá používána k formalizaci her, ve kterých hraje roli pořadí tahů. Hry jsou prezentovány jako *stromy* (Obr. 3). Každý uzel zde reprezentuje místo, ve kterém některý z hráčů vybírá tah. Hráč, který vybírá tah, je určen číslem napsaným u uzlu. Hrany reprezentují možné tahy hráče. Zisk pro jednotlivé hráče je specifikován v listu stromu. Extenzivní forma může zobrazit i situaci, kdy hráči vybírají tahy zároveň a také hry s neúplnou informací. Pokud hráč neví, v kterém z několika stavů je, zakreslí se okolo těchto stavů kružnice.

V příkladě na obrázku jsou dva hráči. Hráč 1 vybírá první a má na výběr buď F, nebo U. Hráč 2 vidí tah hráče 1 a poté vybere buď A, nebo R. Předpokládejme, že hráč 1 vybere U a hráč 2 vybere A. Potom zisk prvního hráče je 8 a zisk druhého hráče je 2.



Obr. 3: Hra dvou hráčů v extenzivní formě

3.4 Typy her

Hry s nulovým součtem a hry s nenulovým součtem

V případě **her s nulovým součtem** je celkový užitek pro všechny zúčastněné hráče a pro každou kombinaci strategií roven nule. Jinak řečeno, vítězný hráč získává na úkor ostatních. Příkladem takové hry je například go, šachy nebo poker.

V reálném světě se většinou vyskytují **hry s nenulovým součtem** (v podnikání, politice, příkladem může být vězňovo dilema), kdy některé výsledky přinášejí celkový čistý užitek větší nebo menší nule. Neboli zisk jednoho hráče nemusí pro jiného hráče nutně znamenat ztrátu.

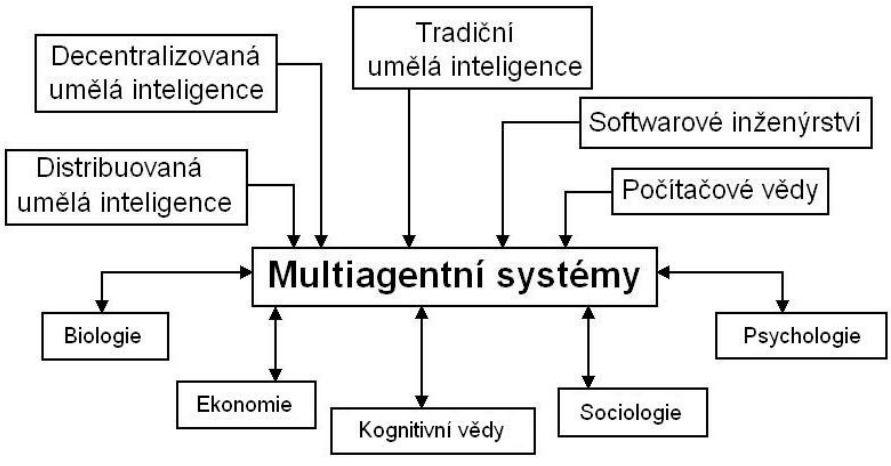
Hry s úplnými informacemi a hry s neúplnými informacemi

Ve **hrách s úplnými informacemi** má každý hráč k dispozici stejné informace týkající se hry jako všichni ostatní. Příkladem můžou být šachy. Naopak **hrou s neúplnými informacemi** je poker nebo vězňovo dilema. Hry s úplnými informacemi se v běžném životě vyskytují zřídka.

4 Multiagentní systémy

4.1 Komplexní systémy

Fyzikální, chemické, biologické a jiné systémy, jako jsou plyny, polymery, makromolekuly, buňky, neuronové sítě, kolonie mravenců, ekonomické systémy v lidské společnosti a další, se vyznačují značnou komplexností chování a projevů, které jsou důsledkem architektury a interakcí komponent, z nichž se tyto systémy skládají. Vyznačují se racionalitou, kterou chápeme jako schopnost(z hlediska času a kvality) plnění funkcí v prostředí, v němž se nacházejí. Tento pohled na racionalitu je jedním z klíčových vlastností teorie multiagentních systémů, disciplíny, která se nachází na rozhraní přírodních, technických a společenských oborů(obr. 4). Spřízněným vědním oborem je také umělý život, který se v mnohém s multiagentními systémy[7] prolíná.



Obr. 4: Interdisciplinární charakter multiagentních systémů. Šipky označují směr, velikost písma míru vlivu.

Jedním z cílů multiagentních systémů je vytvářet modely inspirované komplexními systémy tak, aby plnily svůj účel na pozadí vlastností, kterými se modelované komplexní systémy vyznačují:

- autonomie prvků
- decentralizace řízení systému
- robustnost
- adaptabilita na změny prostředí

Spolu s vlastnostmi navrhovaných systémů v softwarovém inženýrství, jako jsou např. distribuce funkcí a řízení, modularita prvků, jejich škálovatelnost nebo opětovná použitelnost, bychom mohli v budoucnosti vytvářet umělé systémy, které budou schopny konfiguraci v průběhu svého „života“ vyvíjet, učit se apod. a jejich tvorba bude více otázkou stanovení vstupních parametrů komponent a jejich vzájemných vztahů s postupnou evolucí systému, než jeho pracné vytváření od návrhu až po implementaci každého detailu systému.

Uvedené vlastnosti inspirují výzkum v multiagentních systémech v několika dílčích oblastech, z nichž pozornost přitahují zejména následující:

- architektury agentů
- formalizace teorií v MAS
- koordinace, kooperace, komunikace v MAS
- učení a evoluce v MAS
- mobilní agenti
- agentově-orientované softwarové inženýrství
- emergence, atd.

4.2 Agent posun od objektu k vyšší abstrakci

Agent jako pojem pro oblast multiagentních systémů je inspirován funkcí zástupce, který je určen pro vykonávání určitých úkolů. Měl by být samostatným do té míry, aby na člověka – operátora nebo správce systému byla delegována funkce dohledu nebo kontroly s minimálním zásahem do běhu agenta. Agent je ztělesněný v podobě hardwarového nebo softwarového zařízení, které operuje jenom ve vymezeném prostředí. Samotný vliv na prostředí souvisí s racionalitou. Akční potenciál agenta je zaměřen tak, aby stav prostředí nebo agenta přibližoval ke stanoveným cílům(uklizený prostor, nashromážděná data nebo bezkonfliktní pohyb v prostoru mohou být příklady takových stavů).

Ve srovnání s objektem, který je nejpropracovanější abstrakcí reality a zároveň komponentou modelovaného systému v objektově-orientovaném inženýrství, je agent abstraktnější a má některé důležité vlastnosti navíc, zejména:

- autonomii (nezávislost na ostatních agentech)

- existenci v prostředí
- kontinuitu senzorického a aktuačního propojení s prostředím

4.3 Architektury agentů

Z hlediska složitosti organizace vnitřních komponent agentů a jejich racionality lze agenty rozdělit do následujících skupin:

- reaktivní
- deliberativní
- sociální
- hybridní

Reaktivní agent

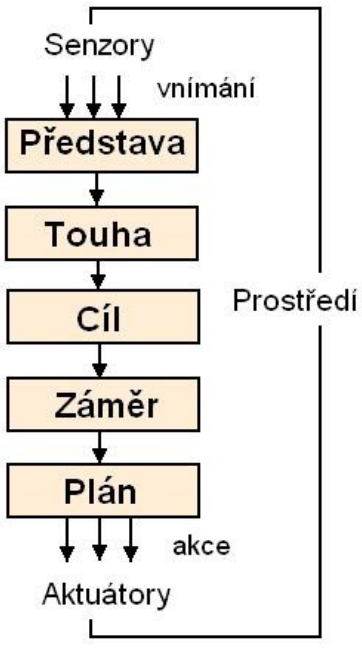
Tento typ agenta má nejjednodušší vnitřní architekturu. Je tvořen několika paralelními procesy(lépe řečeno druhy chování), které jsou spouštěny na základě podnětu z prostředí, nebo agentova vnitřního stavu, případně jejich kombinací. Agent nemá reprezentace prostředí, kromě vyrovnávací paměti, která obsahuje záznam stavu agenta. Akce jsou reakcí na stav prostředí bez možnosti plánování.

Tento typ je často zkoumán ve společenstvích s reaktivní komunikací, ve kterých se zkoumá míra nárůstu racionality chování společenstva na základě organizace a interakcí agentů navzájem a s prostředím.

Deliberativní agent

Deliberativní (uvažující) agent si uchovává symbolické reprezentace prostředí a vnitřních stavů, na jejichž základě vytváří plány pro dosahování svých cílů. Tyto plány mohou být hierarchické uspořádání a cíle jsou ohodnoceny přiřazenou prioritou pro výběr adekvátního plánu.

Reprezentativní teorií deliberativních agentů je tzv. teorie BDI (z angl. belief, desire, intention – představa, touha, záměr). Podle této teorie je postup od vnímání k plánování uzavřeným cyklem (obr. 5 napravo) a každá činnost reprezentuje tzv. mentální kategorii, které jsou důležité z hlediska intencionality a vyšší racionality chování.



Obr. 5: Vazby mezi procesy v deliberativním agentu

Sociální agent

I přesto, že komunikací se mohou vyznačovat i reaktivní agenti, za sociálního považujeme agenta, který komunikuje s jinými agenty ve vyšším komunikačním jazyce.

Hybridní agent

Architektura hybridního agenta kombinuje některé nebo všechny z uvedených architektur v jeden celek.

5 O hře Vlajky

5.1 Pravidla hry

Nejprve si zavedeme pravidla hry, podle kterých se systém bude řídit. Hra má několik jednoduchých pravidel:

- hra je pro dva hráče
- každý hráč má:
  - tábor (hexy, které vlastní)
  - prapor (vlajku)
  - jednotky (vojáky)
- hráči se po odehrání se všemi jednotkami střídají
- vyhrává ten hráč, který má ve svém táboře všechny vlajky
- pokud hráč ztratí všechny jednotky, prohrává

6 Analýza a návrh aplikace

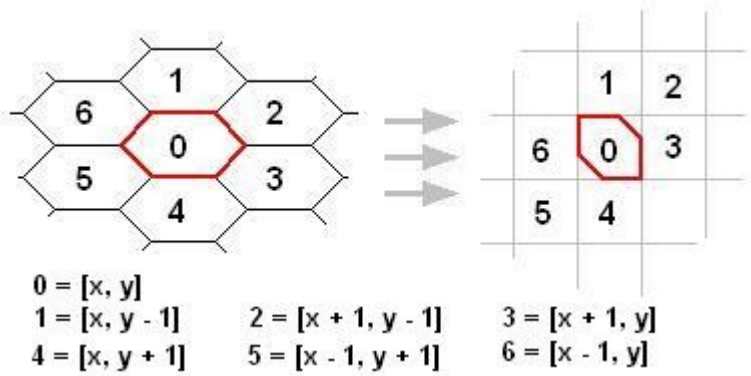
6.1 Prostředí

Prostředí do kterého je hra vložena, má základ v hexa-síti. Ta oproti čtvercové není tak jednoduchá na implementaci a tak tu máme tři problémy:

- uložení prostředí s informacemi
- vykreslení této hexa-sítě
- algoritmus na rozpoznání daného políčka(hexu)

6.1.1 Ukládání hexa-sítě

Před návrhem algoritmu pro rozpoznání pole či vykreslení sítě je třeba správně navrhnout vhodný způsob ukládání této sítě. Protože hex má, na rozdíl od čtverce 6 hran, nemůžeme tak využít dvourozměrného pole (což je čtvercová síť už sama o sobě). Musíme zachovat informace o sousedících polích, proto je nejlepším východiskem je dvourozměrné pole, jako tomu je v čtvercové síti, ale s jistými pravidly o sousedících polích. Na obrázku 6. je znázorněna transformace z hexa na čtvercovou síť s příslušnými pravidly pro určení sousedních polí.



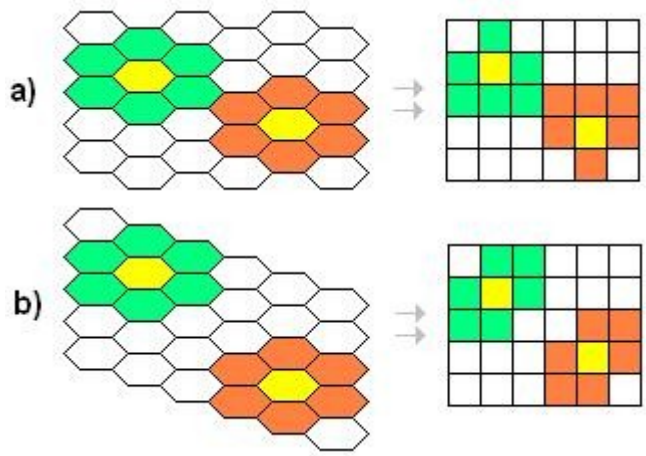
Obr. 6: Transformace hexa-sítě na čtvercovou s pravidly pro sousední políčka

6.1.2 Vykreslení hexa-sítě

Uloženou hexa-síť potřebujeme zobrazit, přičemž musíme brát v úvahu, že samotný proces vykreslování lépe řečeno posloupnost jednotlivých hexů, musí být taková, aby se grafika dobře vykreslila (podrobněji o samotném grafickém vykreslení v kapitole 6.2). Jsou dva způsoby (přístupy), jak vykreslit hexa-síť. Buď do obdélníku, nebo do kosodélníku.

Obdélníkové vykreslení, jak je vidět na obrázku 7a, je při využití celé hexa-sítě estetičtější a přehlednější, protože celkovým tvarem připomíná právě obdélník na který jsme zvyklí z čtvercové sítě. Výhoda je také v úsporném ukládání hexa-sítě do pole, pro specifické tvary sítě jako právě tvar obdélníku, který bývá nejčastější. Ale krásný vzhled a úsporné využití paměti zvyšuje obtížnost implementace tohoto způsobu vykreslení. Problémy nastanou právě zarovnáním hexů do obdélníku, kde se hexy vychýlí od své přirozené osy. Tím totiž přestává platit systém ukládání hexů do čtvercové sítě, jak je popsáno v předchozí kapitole, a dokonce se ukládají různě, jak ukazuje také obrázek 6. Posunem dvou sloupců pro výsledný obdélníkový tvar změníme vlastnosti jednoho sloupce, konkrétně sudého (indexujeme od 0). S tím je spojený i problém neintuitivní editace prostředí a proto je nevyhovující.

Druhý způsob vykreslení sítě, do kosodélníku, není na první pohled tak hezký, ale je mnohem vhodnější než-li první způsob. Ani paměťové využití není tak úsporné, protože například pro výsledný tvar obdélníku musíme nadimenzovat pole zhruba o jednu polovinu. Příčinou je seříznutí zkosení sítě. Naproti tomu je výhodná v jednoduché implementaci, která spočívá v prostém zkosení jedné osy. Systém ukládání se nemění (obr. 7b) , proto i editace mapy je celkem intuitivní.

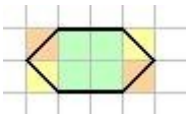


Obr. 7: Ukázka transformace dvou hexů do čtvercové sítě.

a) Obdélníkový způsob vykreslení, b) kosodélníkový způsob vykreslení.

6.1.3 Rozpoznání daného hexu

Pro funkčnost potřebujeme poznat, i na jaký hex jsme klikli či jen přes něj přejeli kursorem myši. I zde si hexa-síť promítneme a tedy i zjednodušíme na čtvercovou síť.

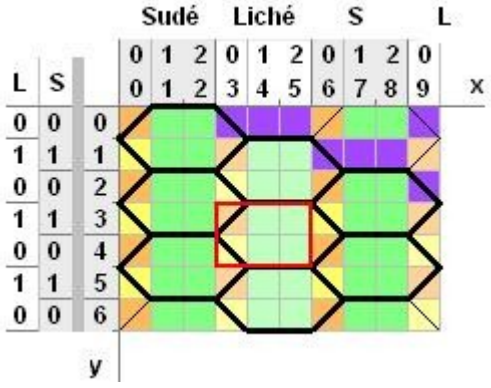


Obr. 8: Hex

Základem je dobře si navrhnout promítnutí hexa pole do čtvercové sítě, ze které se už snadno zjistí jakou pozici (x, y) daný hex má. Pro líbivý tvar hexu a efektu pohledu na mapu pod jistým úhlem je výhodné zvolit 4x2 čtverečky, jak je vidět na obrázku 8. Zelená barva znázorňuje čtverečky, které identifikují hex pouhou pozicí ve čtvercové síti, protože danému hexu náleží plná plocha čtverečku. Žlutá a oranžová barva znázorňují kolizi. Zde se musí i vypočítat poloha kursoru v daném čtverečku, zda je pod nebo nad přímkou. To nám určí, zda jsem klikli vně nebo uvnitř hexu. Platí, že oranžová plocha je *horní konflikt* a žlutá pak *dolní konflikt*, jak je lépe vidět na obrázku 9..

Žlutá a oranžová barva znázorňuje ještě jednu důležitou věc – propojení jednotlivých hexů. V těchto čtverečcích není jednoznačné, na který hex jsme klikli. Můžeme to však jednoduše zjistit podle horního či dolního konfliktu a zda jsme klikli nad přímkou či pod přímkou. Postup určení hexu [*actHexX*, *actHexY*] je následující:

1) určíme souřadnice kursoru myši v naší spuštěné aplikaci [*posX*, *posY*]
2) určíme souřadnice čtverečku [*X*, *Y*] na který jsme klikli (pomocí zbytku po dělení velikostí čtverečku – v našem případě dílek = 16 pixelů)
3) pomocí dalšího zbytku po dělení x-ové souřadnice kursoru trojkou (také jinak *posX* % 3) zjistíme, zda nastal konflikt(0)
4) po celočíselném dělení *posX* / 3 zjistíme aktuální souřadnici hexu *actHexX*
5) zmenšíme y-ovou souřadnici čtverečku o *actHexX* , protože každým sloupcem se daný sloupec posune níže o jeden čtvereček . Zjistíme zbytek po dělení tohoto výsledku. (*Y* - *actHexX)* % 2). Zjistíme zda se případně jedná o horní konflikt (0) .. nebo dolní konflikt (1)
6) teď zjistíme souřadnici *actHexY* = *Y* - *actHexX) /* 2
7) Pokud nenastal konflikt, máme již souřadnice aktuálního hexu [*actHexX*, *actHexY*]
8) jinak zjistíme kam přesně jsme na čtvereček klikli. Opět zbytek po dělení tentokráte velikostí čtverečku
9) pokud je **horní konflikt** a klikli jsme
   - **pod** přímkou **y <= x**, souřadnice aktuálního hexu jsou [*actHexX*, *actHexY*]
   - **nad** přímkou **y > x**, souřadnice tedy jsou [*actHexX* - 1, *actHexY*]
10) pokud je **dolní konflikt** a klikli jsme
   - **nad** přímku **y >= 16 – x**, souřadnice jsou [*actHexX*, *actHexY*]
   - **pod** přímku **y < 16 – x**, souřadnice jsou [*actHexX* - *1*, *actHexY* + *1*]



Obr. 9: Náčrt hexa-sítě ve čtvercové síti

Ovšem toto není vše k úplnému a správnému určení hexu. Jsou tu ještě výjimky. Políčka, která mají zápornou souřadnici (jsou mimo rozsah pole), přesto výše uvedeným výpočtem získají kladné souřadnice. Toto jednoduše ošetříme zavedením výjimek, při kterých se nebude inkrementovat souřadnice *actHexY*. Na obrázku 9 znázorněno fialovou barvou. Problémová políčka jsou:

[x, y] .. souřadnice čtverečku

x = (y+1)*3,

x = (y+1)*3 + 1,

x = (y+1)*3 + 2,

x = (y+1)*3 + 6

6.2 Grafika

Hra Vlajky je okenní aplikace s 2D grafikou vykreslovanou pomocí DirectX. Uživatelská plocha je rozdělena do tří základních částí *rám mapy, dolní menu*, *vedlejší menu* (obr. 10). Vykreslení všech těchto částí je dynamické. To znamená, že při změně velikosti okna se změní velikost rámu mapy. Dolní menu má vždy stejnou výšku a vedlejší menu stejnou šířku. Jejich pozice závisí na pozici a velikosti rámu mapy.

Obr. 10: Grafická ukázka hry Vlajky

Rám mapy

V této části se nejdříve vykreslí celá mapa. Následně rámeček mapy překryje a ohraničí prostor, v němž je mapa aktivní a reaguje na události kursoru myši. Rámeček je vykreslen dynamicky ze čtyř textur(4 rohy) a barvy vyplňující vnitřní část rámečku. Zde je barva průhledná, ale při vhodně zvolené barvě můžeme ztmavit mapu (či použít jiný barevný efekt) třeba na znázornění pozastavení hry. Pokud nevidíme celou mapu(při velké velikosti nebo je jen posunuta), můžeme pohybovat s mapou pomocí klávesových *šipek*.

Dolní menu

Dolní menu zakryje neaktivní část mapy pod rámečkem a také je v něm umístěn *inventář*. V inventáři se zobrazují objekty (chcete-li předměty), které má jednotka u sebe (tedy ve svém inventáři). Ve 4 boxech inventáře se tedy mohou zobrazit maximálně 4 předměty. Šipkami doleva a doprava, umístěné vedle boxů, se pohybujeme po celém inventáři jednotky.

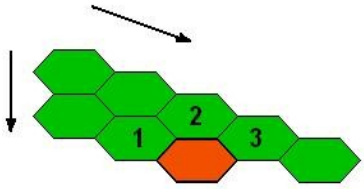
Vedlejší menu

Vedlejší menu má také funkci zakrytí momentálně nepoužívané části mapy a dále tu jsou umístěny statistiky aktuální jednotky. Ty nás informují o tom, která jednotka je označena, kolik má zdraví, útok, obrana a kolik může dát zranění. Dále tu jsou informace o aktuálním *kole*, aktuálním hráči, akci a rychlosti hry. Jelikož hra nemá žádné animace, je vhodné alespoň zobrazit, zda jednotka něco dělá nebo ne.

6.2.1 Mapa

Mapa stejně jako další komponenty (rám mapy, dolní i vedlejší menu, inventář, statistiky, info) mají svou pozici (levý horní roh) a velikost (šířku a výšku). Zde velikost mapy určuje aktivní oblast (vnitřek rámu mapy) a pozice mapy udává pomyslný střed soustavy, od které se počítá hexa-síť. Změnou středu soustavy tedy můžeme jednoduše pohybovat s mapou.

Jednotlivé hexy se vykreslují v několika průchodech po vrstvách. Nejprve se vykreslí hlína pod zemí a zem. Dále se vykreslí mobilita označené jednotky, zvýrazní se hex na který jsem klikl, dále se zvýrazní hex hex po kterém se pohybuji kursorem myši a minimální cesta. Dále pokračujeme vykreslením objektů: předměty na zemi(každý hex má svůj inventář), agenty (pohybující se objekty) a nakonec ikony *take* (vzít předmět) a *attack* (zaútočit) znázorňující, jakou akci můžeme provést.



Obr. 11: Systém vykreslení

Toto je popis vykreslení jednoho hexu. Posloupnost vykreslení všech hexů, tedy celé hexa-síťě, je taková, aby se nejprve vykreslily textury, které jsou od nás nejvzdálenější a ty jsou postupně překrývány texturami, které jsou blíže. Toto bezpečně realizuje systém vykreslení znázorněný na obrázku 11. Hexy se vykreslují zleva doprava od nejhořejšího řádku po nejspodnější. Tím zaručíme pokaždé situaci, kde hexy 1,2 a 3 se vykreslí dříve než oranžový hex. Platí že oranžový hex může překrýt hex zelený. Dává nám to možnost vykreslit vysoké či široké textury (stromy, vojáky, skály a další – fantazie a kreativitě se meze nekladou), jak je znázorněno na obrázku 12.


Obr. 12: Ukázka

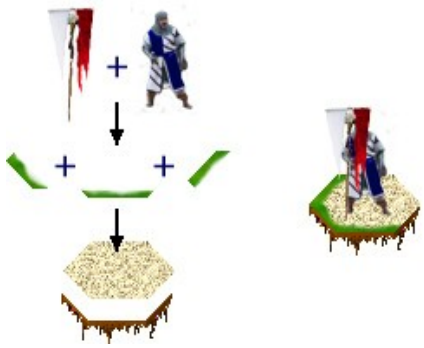
Textury

Mapa je tedy poskládána z několika vrstev a různých textur jako skládačka. Malé textury spolu s průhledností nám umožňují vytvářet další rozmanitější a různorodé textury, které bychom jinak museli pracně vytvářet. Pracné by to bylo především z důvodů obrovského počtu možností, jakých by textura mohla dosáhnout.

Les a skály

Les i *skály* můžeme vykreslit buď konkrétní texturou nebo generovat. Konkrétní texturou se to zdá být implementačně jednodušší, ale není tomu tak. Nejen, že je v první řadě pracné vytvářet velké množství textur (pokud nechceme všude stejný les), ale i samotné rozpoznávání, která textura se má vykreslit pro danou situaci. Je výhodnější vhodně vytvořit textury, které se dají poskládat do sebe – generovat nové textury za běhu programu(obr. 13). Potom se nemusíme starat, jaký objekt je na kterém povrchu.

Je tedy výhodné mít texturu jednoho stromu, nebo raději více textur různých stromů (v aplikaci jsou tři) a ty náhodně generovat do hexu. Tím se otevírá nespočet různých možností, jak vykreslit N objektů stromů na daný hex. Tímto systémem pak docílíme reálnější prezentaci lesa.



Obr. 13: Skládání textur

Tento systém platí nejen pro les ale i skály a další fakta, které detailněji popisují svět (např. generování různých keříků, květin na trávě). Tato aplikace má však jen jednu texturu skály a rozdílný efekt vidíme na ukázce na začátku kapitoly(obr. 10).

Inteligentní rozpoznání okolí

U povrchu *písek* a vysvícení mobility označené jednotky jsou přechody mezi jiným prostředím moc ostré a pro lepší vzhled potřebujeme plynulejší přechod. Opět můžeme všechny možné situace předkreslit do textur a složitě vyhodnocovat, která textura se má vykreslit, nebo texturu pro danou situaci opět vygenerovat. Stačí nám 6 textur, které nám provedou přechod z jakéhokoliv prostředí do trávy. Princip je vidět na obrázku 13. Na podobném principu je založené i vysvícení. Zatímco u přechodu mezi *pískem* a *trávou* vykreslí právě kdy spolu tyto dvě prostředí sousedí, u mobility je tomu naopak, čímž přechody spojují jednotlivé vysvícené oválky a dává to dlaždicový efekt.

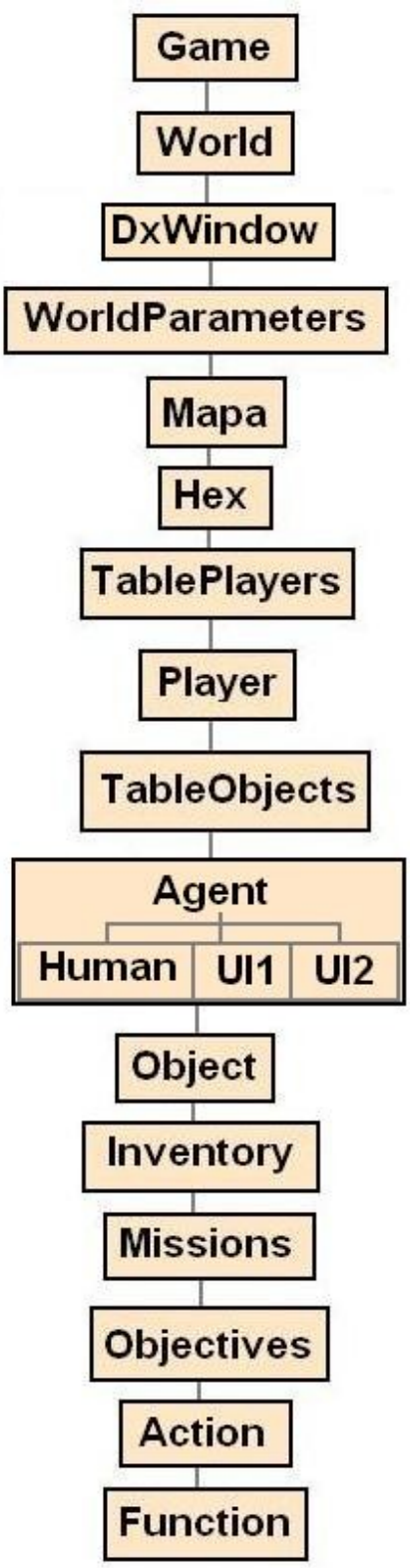
Tábor

Tábor hráčů je obyčejná *tráva* přes kterou je s určitou průhledností překreslen barevný hex (modrý, či červený)

6.3 Herní systém

6.3.1 Objektový návrh

Herní systém spočívá na objektovém návrhu. Ten dokáže práci s programem dost usnadnit, pokud je dobře navržen. Dobrý objektový návrh je většinou složitější a potřebuje tedy i mnohem více času strávený vymýšlením a implementováním. O to rychleji a snadněji se nám bude pracovat s tímto herním systémem. Při návrhu je nutné také zvážit, nejen co od něj vyžadujeme, ale i možné problémy a jak by se dali jednoduše vyřešit. Málokdy se podaří navrhnout tak dobrý objektový návrh, aby jej není potřeba měnit a i tentokrát návrh prošel několika menšími , ale i většími změnami. Většími změnami se chápe, předělání větší části návrhu, což není zrovna příjemné, ale stojí to za zjednodušení programování, urychlení výpočtu aplikace či jen úspornější ukládání v paměti. Výsledný objektový návrh ukazuje obrázek 14. Výhoda takto uspořádané struktury tříd je hlavně v tom, že přidání či odebrání další třídy je tak snadné jako přidání či odebrání prvku v lineárním seznamu, což u složitě větvené struktury jde jen velmi těžko.



Obr. 14: Objektový návrh

Game

Game je třída, reprezentující konkrétní instanci tohoto herního systému. Nastavuje událostem (kursor myši, klávesnice, window), co se má dělat. Volá příslušné funkce ve World.

World

Nejdůležitější třída reprezentující svět – zde se zpracovávají události aplikace na mapě i v menu (kursor, klávesnice,..) a také události tohoto světa, prezentující *život* v tomto světě – funkce

LiveInWorld. Víceméně se jedná o akce agentů. Dále řídí vykreslování (načítání textur, inicializace, samotné vykreslení), načtení mapy a hráčů. Má repositář s typy UI (Human, UI1,UI2)

DxWindow

Třída starající se o práci s DirectX9. Jejím úkolem je na začátku vytvořit okno pomocí WinAPI a do něho inicializovat DirectX, včetně práce s ním při změnách rozměrů okna, minimalizaci, atd.

WorldParameters

V této třídě jsou uložena veškerá data o světě (mapa, hráči, události kursoru, aktuální hex (mousemove, buttondown, atd.), aktuální hráč, aktuální objekt, atd.) Výhodou je, že jsou tak data uložena na jednom místě a tím každý, kdo si o ně požádá, dostane stejné informace. Můžeme to charakterizovat jako synchronizace dat.

Mapa

Po spuštění se načte mapa a hráči ze souboru, vygenerují se různé objekty(stromy v lese, skály, vlajky, agenti). Uchovávají aktuální stav mapy, což reprezentuje pole Hexů. Nastaví *MaxUsedX* a *MaxUsedY*, což jsou hodnoty maximálního využití pole (optimalizace pro režiji výpočtu), dále obsahuje *mapaMobility* (nenulovou hodnotou charakterizuje zda na políčko může jít a kolik potřebuje chodících bodů). Tato mapa se vypočte celá jen jednou na začátku, následně se jen aktualizuje konkrétní změna (pohyb či úmrtí agenta). A jako třída reprezentující viditelnou komponentu, má pozici levého horního rohu (střed souřadnic pro hexa-síť) a velikost mapy (šířka a výška)

Hex

Uchovává v sobě informace, které náleží danému hexu. Texturu hlíny(náhodně se vybere na začátku), texturu země, inventář hexu (předměty ležící na zemi), objekt (agent) pokud je na hexu a seznam *rozhovorů* (jinak řečeno to, co je na tomto hexu slyšet)

TablePlayers

Třída řídící seznam hráčů. Stará se o přidání nového hráče, obnovení při začátku nového kola a předání hry dalšímu hráči.

Player

Tato třída reprezentuje hráče (typ, id, UI, barva pod kterou hraje). Dále má seznam objektů *TableObjects* se všemi objekty, které vlastní. Třída mimo jiné také obstarává zabrání předmětu a nastavení UI.

TableObjects

Třída spravuje operace nad seznamem objektů (přidat, odebrat, další a předchozí pohybu-možný objekt, vygenerovat pozici na hexu).

Agent

Abstrakce objektu lépe řečeno mozek či myšlení daného objektu. Tato třída obsahuje vše, co je pro UI a hráče člověka stejné. Umí mechanické věci jako *jdi, počkej, nedělej nic, utoč, vezmi, polož, uhni.*. Dále obstarává nastavení cílů, prohledávání mapy a hledání minimální cesty.

Human

Podtřída třídy Agent reprezentující myšlení člověka, přesněji řečeno – za objekt myslí hráč sám.

UI1

Druhá podtřída třídy Agent. Tato reprezentuje umělou inteligenci reaktivního agenta. Plánuje cíle, akce, rozhoduje se, které udělá nejdřív, kontroluje se proti zacyklení atd.

UI2

Třetí podtřída třídy Agent. Tato reprezentuje umělou inteligenci kooperativního (deliberativního) agenta. Základ má jako UI1 a navíc mezi sebou agenti komunikují pomocí zpráv a spolupracují.

Object

Další z důležitých tříd. Tato třída zastupuje fyzický objekt (něco jako tělo). Uchovává informace jako počet *životů*, *obrana*, *útok*, kolik dá *zranění*, *jméno*, *id*, *texturu*, *typ*, *UI*(myšlení), *vlastníka*, na kterém hexu stojí, kolik políček může ujít, jak daleko vidí, mise(k vítězství), cíle(ke splnění misí), pole mobility, pole vzdáleností, minimální cesta (pokud někam jde), *Z* index(podle něj se určuje pořadí vykreslení na daném hexu – např. pro správné vykreslení stromů na jednom hexu), levá a pravá ruka (zda v nich drží nějaký předmět). Jelikož patří mezi třídy reprezentující viditelnou komponentu, má parametry levý horní roh, šířku a výšku pro vykreslení. Dále má funkce spravující fyzický objekt jako *polož, seber, útoč, jdi*(přesněji přemísti se), *zemři*(odhodí všechny předměty které má na zem, aktualizuje mapu a smaže se z paměti)

Zde se instance této třídy dělí na *agenty* a *předměty*. Pokud objekt může chodit (má body k chození) je považován jako agent, jinak jako předmět. Předmět může být v inventáři a být držen v ruce.

Inventory

Viditelná komponenta charakterizující seznam *předmětů*, má tedy také parametry levého horního rohu, výšky a šířky komponenty. Inventář může mít jak hex tak i jakýkoliv objekt. Tedy můžeme mít klidně předmět měšec, který je naplněn dalšími předměty. Dále má funkce spravující inventář jako přidání, odebrání, nalezení předmětu a item(vrátí předmět na daném indexu). Funkce přidání vkládá předmět do seznamu podle *Z* indexu a inventář je tedy vždy seřazený pro správné vykreslení. Dále má některé funkce z třídy vector (front, back, push_back, pop_back, size, empty a clear).

Missions

V této třídě je uložený *cíl mise* jinak řečeno úkoly k dosažení vítězství. Úkoly jsou reprezentovány jednotlivými *objectives*.

Objectives

Tato třída spravuje akce objektu. Má seznam akcí fungující jako FIFO. Umožňuje vložit a odebrat akci jak na začátek fronty tak i na konec(dle druhu akce) a vyčištění seznamu.

Action

Třída reprezentující akci. Obsahuje druh akce, cílový hex, případný cílový objekt a prioritu.

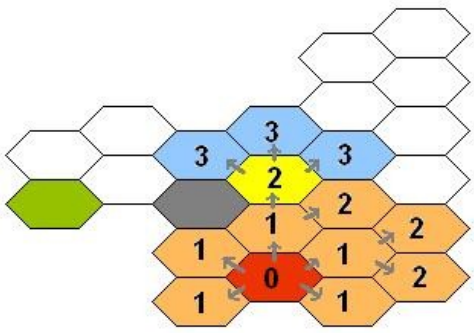
6.4 Prohledávání mapy a minimální cesta

Základem většiny her a nejen strategií je prohledání stavového prostoru. I my potřebujeme prohledávat mapu. Režie na výpočet, režie paměti či přesnost závisí na kvalitě návrhu. Tím bude i umělá inteligence reálnější a kvalitnější a aplikace rychlejší.

Od prvního až po konečný návrh hledání minimální cesty proběhlo hodně změn a rychlé a kvalitní hledání je až při 6. návrhu. Jak probíhal postup hledání a vymýšlení optimálního prohledávacího algoritmu si teď ukážeme na jednotlivých verzích.

Návrh 1. - Prohledávání s informací, kde jsem už byl, dokud nenajdu cíl

Základním principem je, že si do pomocné proměnné *poleCesta* uložíme na cílový hex souřadnice cílového hexu a na startovní hex souřadnice startovního hexu a vynulujeme *poleVzdalenosti*. Začneme prohledávat od startovního hexu tak, že prohledáme všechny sousedící hexy(je jich



Obr. 15: Prohledávání - návrh 1

maximálně 6) a pokud na něj můžeme vstoupit a ještě jsme na něm nehledali, uložíme do *poleCesta* na pozici nového prohledávaného hexu hex, ze kterého jsme se na nový dostali a do *poleVzdalenosti* na toto místo uložíme o jedničku vyšší vzdálenost než je na hexu, ze kterého jsme se na nový dostali, nový hex uložíme do fronty k prohledání. Na obrázku 15. je znázorněn výše uvedený princip: červená = start, zelená = cíl, šedá = překážka, oranžová = zde jsem už byl, žlutá = aktuálně prohledávaný, modrá = nově prohledávané hexy, šipky udávají směr cesty.

Tato rychlá metoda je vhodná pro člověka, protože kdy chce jít někam, kam mu to cestu nenajde, sám hledá kudy by to šlo alespoň přes jiného agenta. A to je zásadní problém této metody. Ostatní agenty vnímá jako zed, přes kterou nemůže jít. Člověk si poradí, ale u umělé inteligence nastane „zamrznutí“, v lepším případě, kdy je ošetřeno zacyklení, agent toto kolo zůstane stát a nikam se nepohne i když by se k cíli mohl alespoň přiblížit. To řeší druhý návrh.

Návrh 2. - Prohledávání bez ostatních agentů

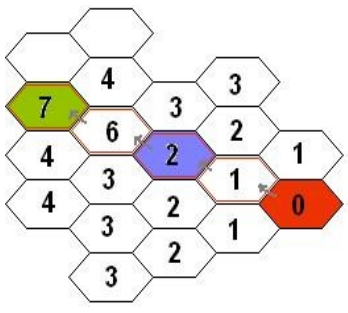
Princip je stejný jako u návrhu 1, ale neuvažujeme ostatní agenty. Tento krok nebyl vydařený, protože zprvu agenti přes sebe chodili a mazali se tak z mapy. Po té, kdy jsme ošetřili, aby agent nemohl vstoupit na políčko, kde je jiný agent se zase stávalo to, že i když měl agent cestu volnou, tak jelikož minimální cestu našel skrze jiného agenta, šel po této cestě a pak hloupě čekal až se mu uvolní cesta.

Toto není správné řešení a ani ne lepší než návrh 1.

Návrh 3. - Kombinace návrhu 1 a 2

Pokud máme dvě metody, kde každá pracuje napůl správně a každý selže v jiné části, naskýtá se nám možnost tyto metody zkombinovat a využít je pouze v případě, kdy fungují správně. Tedy nejdříve zkusíme najít minimální cestu první metodou (s agenty). Pokud cestu najdeme, je vše v pořádku a pokud ne, zkusíme metodu druhou (bez agentů). Pokud bychom ani teď nenašli cestu, evidentně bychom se na toto políčko ani nemohli dostat.

Tato metoda je už kvalitnější ovšem 2x prohledávání prostoru je dost neefektivní, což se ukázalo jako problém u větší velikosti mapy. V úvahu připadá myšlenka optimalizace pomocí limitní vzdálenosti *distance,* ve které se bude hledat. Ale i toto nepřipadá v úvahu, protože se rapidně znevýhodňuje hráč s UI oproti člověku. Najít optimální vzdálenost, kdy není ještě tak velká, aby výpočet nebyl pomalý a nebyla ani krátká, abychom vůbec dokázali cestu ve většině případů najít, je samo o sobě těžké. Řešení optimalizace nacházíme v návrhu 4.



Obr. 16: Prohledávání - návrh 4

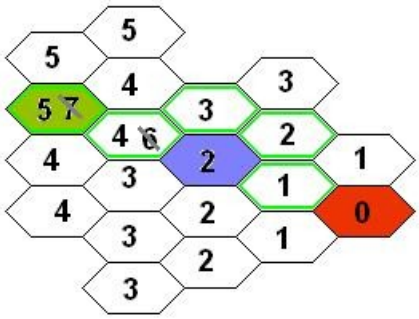
Návrh 4. - Prohledávání s postihem

Tato metoda je jednoprůchodová (počítá se tedy jen jednou) a využívá jistého *postihu*. Postihem chápeme, kolik chodících bodů spotřebujeme, pokud vstoupíme na daný hex. Například tráva(louka) nám vezme jeden *chodící bod*(MP – z angl. movement points), písek 2MP a agent má postih následovně. Pokud k němu dojdu ještě toto kolo, postih činí zbylé MP + 1, jinak 3MP.

Svého jsme ale nedosáhli. Našli jsme sice cestu jedním průchodem, když jsme museli jít přes agenta (toto obrázek 16 neukazuje), našli jsme i cestu, pokud jsme měli volný prostor a stačilo druhého agenta jen obejít. Ovšem, jak je vidět z obrázku 16. agent neobešel druhého agenta. Důvod je zřejmý, zavedli jsme různé postihy, ale to na prohledávání prakticky změnilo jen to, že bych se na to dané políčko dostal později. V podstatě jsme změnili jen číslo, které jsme zapsali do *poleVzdalenosti*. Toto vyřešil návrh 5.

Návrh 5. - Prohledávání s informací menší vzdálenosti

Už nám nestačí prohledávat jen tam, kde jsme ještě nehledali. To nám stačilo, dokud jsme počítali všude se stejným postihem (konkrétně 1MP).



Obr. 17: Prohledávání - návrh 5

Princip této metody je stejný jako u předchozího návrhu 4 ovšem s tím rozdílem, že prohledáváme nejen hex, na kterém jsme ještě nehledali, ale ten, na který se dostaneme dříve (zapíšeme do *poleVzdalenosti* na toto místo nižší vzdálenost). Tedy některé hexy prohledáváme vícekrát. Na obrázku 17 je znázorněno prohledávání mapy a nalezení minimální cesty. Jsou vidět dva vícekrát prohledávané hexy.

Konečně jsme našli metodu, která dokáže najít minimální cestu nehledě na to, zda mu agent v cestě překáží či nikoliv. Kvalita metody je ale „placena“ režií výpočtu. T0 že hexy se mohou vícekrát přepočítávat (a že v jistých případech by se jich přepočítávalo dost), způsobilo, že se **vždy prohledá celá mapa**. To je při velké velikosti mapy až katastrofální problém.

Opět se nabízí optimalizace pomocí kombinace. Tentokrát kombinace této metody do určité vzdálenosti a nad tuto vzdálenost dále prohledávat metodou 4. Z principu bychom v blízkém okolí prohledávali podrobně a od jisté vzdálenosti už né tak podrobně. Ač se zdá tato optimalizace být dobrá, pořád je pomalá (režie podmínek, zda jsem ještě v limitu apod.). Toto vše však řeší návrh číslo 6.

Návrh 6. - Optimalizace prohledávání s informací menší vzdálenosti

Předchozí metoda je pomalá, kvůli častému, a mnohdy zbytečnému, přepočítání minimální cesty. Je třeba si uvědomit: Kdy a jak často potřebujeme výpočet? Jak často se mění prostředí, kde změny ovlivňují minimální cestu? A hned tu máme jedno řešení.

Výpočet minimální cesty se provádí vždy, když s označeným objektem klikneme kam chceme jít (musí se najít minimální cesta pro její vykreslení). Dále se minimální cesta počítá také při každém zjišťování vzdálenosti daného hexu od označeného objektu (třeba při plánování). Výpočtů je za jedno kolo, kdy odehrává jeden jediný objekt, dost. Přitom pokud agent stojí, minimální cesty ani vzdálenosti se nemění. Toho se dá využít. Konečná a tedy nejlepší metoda prohledávání, která je jak kvalitní tak i rychlá, je popsána v následující kapitole.

6.4.1 Optimální metoda prohledávání

Jak je psáno v předchozí kapitole, tato metoda využívá toho, že stačí vypočítat *poleVzdalenosti* a *poleCesta* jen po každé změně (jen když je třeba) a z toho si pak už s daleko menší režií odečíst minimální cestu či jen vzdálenost k danému hexu.

Zde dostává slovo *mapaMobility*, která je uložená v Mapě. Udává nám právě postihy pro dané políčko (0 – nelze na políčko vstoupit, 1 – tráva, 2 – písek, 5 – agent). Povšimněme si, že nyní má agent postih vždy 5MP. Je to z toho důvodu, že předem daná hodnota opět sníží režii a je dokonce i vhodnější dát větší postih a to ještě každému agentovi (vyplynulo z experimentů). Vidíme, že *mapaMobility* nám zařizuje hned několik věci současně. Jednoduše kontroluje zda na políčko agent může (nemusí být režie zda není mimo mapu, zda hex není skála, zda hex není les), zároveň mi to číslo říká kolik mám na následující hexy přičíst.

Postup tedy je takový.

1. Po označení agenta si vypočítáme svou vlastní *mapuMobility*, protože hex, na kterém stojí, si změníme z 5 na 1, protože on sám si v cestě nepřekáží. Tato mapa se nezmění po celou dobu odehrávání tohoto konkrétního objektu (výjimka, pokud by někoho zabil). A po dalším označení si mapu znovu vypočítáme.
2. Na začátku si také dopředu vypočítáme *poleVzdalenosti* a *poleCesta*. Můžeme říci, že máme teď vypočítané vzdálenosti všech hexů vzhledem k sobě a cesty ke každému hexu na mapě.
3. Teď zavoláme potřebnou funkci (minimální cesta nebo jen vzdálenost) a celkem bez režie máme hned výsledek. Minimální cesta má složitost rovnu délce minimální cesty. V cyklu jdeme v poli *poleCesta* od startovního hexu po cílový. A Výpočet vzdálenosti je ještě jednodušší, má složitost rovnu přečtení hodnoty z pole.

Tento algoritmus prohledává **celou mapu kvalitně a efektivně**. Ale i přesto, pro obrovské mapy, by bylo zbytečné počítat celou mapu, pokud už ani nedává smysl prohledávat do příliš velké

vzdálenosti. Zde se konečně vyplatí optimalizace limity vzdálenosti, do které se bude prohledávat, ale v naší aplikaci toto není potřeba.

Další optimalizací se jevila možnost, seřazení hexů podle vzdálenosti příchodu na toto pole. Jinak řečeno, prohledávali bychom hexy, na které se dostaneme nejdřív. Tím odstraníme zbytečné vícenásobné přepočítávání stejných hexů. Seřazení je prostý push_back na konec fronty s tím, že pokud vkládáme hex s nižší vzdáleností, posuneme index o jeden dopředu a zkusíme vložit znovu. Zní to krásně, ale v praxi samotné seřazování trvá déle, než mapu prostě a jednoduše prohledat i s tím vícenásobným přepočítáváním některých hexů. Důvodem může být to, že hexů, které se vícekrát přepočítávají je vzhledem k velikosti mapy málo, na rozdíl od faktu, že třídění hexů je při každém prohledání nového hexu.

6.5 Umělá inteligence

Pokud se bavíme o umělé inteligenci (UI) je třeba si hlavně uvědomit, že se nejedná o vykonání akcí jako třeba *polož předmět*, *seber předmět*, *jdi na hex*, *zaútoč na nepřítele*. Toto zvládne i objekt, který ovládá člověk. UI je nahrazení lidského přičinění a v tomto případě je to *rozhodování* a *plánování*. UI se tedy stará o tu myšlenku, která „řekne“ objektu, udělej toto či udělej tamhleto.

Základem dobré UI je hlavně kvalitní prohledávání stavového prostoru, což jsme si už navrhli a v následujících kapitolách si popíšeme UI1 a UI2.

6.5.1 UI1 – reaktivní agent

Reaktivní agent prohledá mapu, naplánuje možné akce a rozhodne se pro nejvhodnější. Podobným principem funguje UI1. Postup plánování je takovýto:

1. Na začátku kola hráče se označí první agent v seznamu objektů, které hráč vlastní.
2. Tím začne kolo pro daného agenta, který si vypočítá minimální cesty (jak je popsáno v předchozí kapitole).
3. Agent si naplánuje akce
4. Rozhodne se které jsou nejvýhodnější a setřídí si seznam akcí
5. Pokouší se vykonávat naplánované akce
6. Pokud už nemá co by dělal, nebo nemá MP předá hru dalšímu objektu, pokud není další objekt, předává hru dalšímu hráči.

Plánování

1. Agent najde všechny hexy svého tábora a náhodně si zvolí jeden hex, svého tábora. Tam si naplánuje případné položení vlajky
2. Prohledá celou mapu, a uloží si zvlášť seznam vlajek a zvlášť seznam vojáků(nepřátelských agentů)
3. Naplánuje akce s vlajkami s danou prioritou. Priority jsou zvoleny tak, aby položil vlajku, když je ve svém táboře. Naopak sebrání a položení mají základní prioritu.
4. Naplánuje akce s nepřáteli. Pokud je nepřítel v jeho táboře, priorita, že na něj zaútočí, je vyšší. Pokud má nepřítel vlajku priorita se zvýší, Pokud je nepřítel v dohledu, priorita se také zvýší. Podle zranění (kolik nepříteli zbývá života) se zvýší priorita – s vidinou, že tohoto nepřítele brzy zneškodní. V případě, že se k nepříteli dostane přes jiného agenta, priorita se sníží. Podle těchto pravidel se sečtou různé zvýhodnění a akce *útok* na daného nepřítele má tedy svou prioritu.

Rozhodování – seřazení akcí

Teď, když má agent v objectives naplánované akce, tak si je agent seřadí a to takto:

1. Načte si do pomocného seznamu všechny hexy na mapě seřazené od nejbližšího
2. Zkopíruje si do pomocné proměnné seznam akcí
3. V cyklu prochází všechny tyto hexy (či dokud není pomocný seznam akcí prázdný)
    1. Pokud je daný hex cílem nějakých akcí agenta, přepočítá *hodnotu políčka* a pod tuto hodnotu uloží do struktury *map* daný hex a pod něj danou akci.
4. Takto si vytvoří strukturu *map*, která je krásně seřazená od nejvýhodnější akce
5. Nakonec si agent z této struktury zkopíruje akce v tomto seřazeném pořadí do svých objectives
6. Rozhodne se pro první akci ve frontě v objectives

Hodnota políčka

Ta se vypočítá následovně:

hodnotapole = distance + postihSkrzAgenty – sumPri – priorita

kde

distance ...	je vzdálenost daného hexu od agenta
postihSkrzAgenty ...	postih, pokud po cestě má agenty, platí:
	postih = pocet_agentů * 4

	postih = 1000 .. pokud se na daný hex ani nemůže nikdy dostat.
sumPri...	je suma priorit všech akcí, směřující na tento hex
priorita	je zvýhodnění 0 – 5 podle života nepřítele
	priorita = 5 – život_nepřítele/10

- **distance** – čím dál je cílový hex, tím menší priorita, že si vyberu tuto akci
- **postihSkrzAgenty** – čím větší počet agentů, přes který bych musel jít, tím menší priorita a menší pravděpodobnost, že si vyberu tuto akci
- **sumPri** – čím více akcí bych splnil na tomto hexu, tím je pro mne toto pole žádanější
- **priorita** – pokud je na tomto hexu nepřítel a je dost zraněný, je výhodně jít nejprve za ním a zneškodnit ho

Akce

- **WAIT**

  Pokud agent čeká a může chodit, naplánuje akce k aktuálnímu dění a vykoná je,

  pokud nemůže už chodit, předá hru dalšímu objektu, respektive dalšímu hráči.
- **IDDLE**

  Pokud agent nic nedělá, pokusí se naplánovat akce a začne je vykonávat.
- **GO**

  Pokud může agent chodit, jde na cílový hex. V případě, že narazí cestou na nepřítele, zaútočí na něj, pokud potká svého spoluhráče, poklepe mu na rameno a řekne ať uhne (nastavím mu akci SIDESTEP).

  Pokud dojde na cílené políčko, s úspěchem zruší tuto akci GO

  Pokud už nemůže chodit, nastaví první akci WAIT
- **TAKE**

  Pokud je na políčku, kde má vzít předmět, vezme jej a úspěšně zruší akci TAKE, pokud není na tomto políčku nastaví akci GO na toto políčko

  Pokud předmět na tomto políčku už není, zruším tuto akci TAKE
- **DROP**

  Pokud je na políčku, kde chce předmět položit, položí ho a úspěšně zruší akci DROP.

  Pokud není na cíleném hexu, nastaví akci GO na cílený hex.

  Pokud nemá daný předmět v ruce, najde si ho a nastaví akci TAKE na toto políčko
- **ATTACK**

  Pokud je už u nepřítele zaútočí a úspěšně zruší akci ATTACK.

Pokud je daleko od nepřítele, nastaví akci GO na cílený hex.

Pokud jsem zabil svým útokem nepřítele, podívá se zda po něm zbyly nějaké věci na zemi, pokud ano nastaví akci také na tento hex

- **SIDESTEP**

  Pokud má nějaké sousedící volné hexy, prioritně náhodně volí z těchto hexů a nastaví si akci GO na tento hex

  Pokud nemá kolem sebe volno, kam by uhnul, náhodně vybere jakým směrem půjde a nastaví si akci GO na tento hex(akce ko v příštím kole řekne ať uhne ten druhý)

  Pokud nemá kam uhnout nastaví akci GO na agenta, který řekl uhni(princip je ten, že první agent mu uhne, druhý odejde, a první se konečně dostane tam kam chtěl.)

6.5.2 UI2 – kooperativní agent

Tento agent má základ v UI1, plánování, rozhodování, provádění akcí je takřka stejné, jen s malým rozdílem. Agenti s UI1, vykonávají své akce sériově, tak jak dostanou možnost hrát. Tudíž ztrácí mnohdy své MP, protože zbytečně čekají apod.

Agenti s UI2 všichni nejprve naplánují své akce a podle situace, kdo má nejvýhodněji nebo nejrychleji hotový úkol, začíná. Převážně to je ten, kdo má k cíli nejblíže, což si řeknou pomocí akce SPEAK. Plyne z toho další rozdíl oproti UI1. Agent s UI2 než začne plánovat, nejprve naslouchá (akce LISTEN). Když zaslechne žádost o pomoc, rozkaz k úkolu či jiný rozhovor, který mu určí akci, ... zkusí si naplánovat svoje akce, a tyto nové, sdělené zprávami začlení do svých akcí. Tímto se vyřeší třeba to, že vlajku sebere ten, kdo je opravdu nejblíže. Dva či více se „dohodnou“, že zaútočí na konkrétního nepřítele, a rozvrhnou si kdo si kam stoupne (aby každý udělal co nejméně kroků) a pokud na někoho jde přesila, zavolá si o pomoc. Dotyčný, který to zaslechne a pokud je také sám, zavolá také o pomoc(tím to slyší další spolubojovníci) a sám jde vypomoci v útoku či v obraně. Agenti UI2 se také mohou dohodnout, že většina bude útočit a dva mají nepozorovaně sebrat vlajku. Toto vše UI1 nedokáže, protože každý agent s UI1 jedná za sebe, co je pro něj v danou chvíli nejlepší.

Jelikož plánování a setřídění akcí je stejné přidané akce jsou tyto:

Akce

- **WAIT**

  Pokud agent čeká a může chodit, nastaví akci LISTEN.

  Pokud nemůže už chodit, předá hru dalšímu objektu, respektive dalšímu hráči.

- **IDDLE**

Pokud agent nic nedělá, naslouchá – nastaví LISTEN

- **LISTEN**

  Vyslechne si všechny rozhovory na hexu, v kterém stojí, uloží do objectives a naplánuje k tomu i své akce

- **SPEAK**

  Zprávy, které říká se ukládají do seznamů rozhovorů příslušných hexů do určité vzdálenosti od mluvícího agenta.

Zpráva (rozhovor)

Zpráva je reprezentována stringem. Stejně jako kdyby mluvil. Ve zprávě jsou obsaženy důležité informace a to *autor* zprávy(kdo mluvil), *příjemce* zprávy(komu je zpráva určena) a *obsah* zprávy.

Příklad 1: zprávy: „`Vojak10 > ALL : ATTACK id= 15 hex= 6 3`“

Co tento příklad vlastně znamená :

Voják10 sděluje všem, kteří ho slyší, aby útočili na nepřítele s id = 15, který je na hexu (6, 3).

Příklad 2: zprávy: „`Vojak7 > Vojak6 : TAKE id= 77 hex= 0 5`“

Co tento příklad vlastně znamená :

Voják7 povídá Vojákovi6, aby vzal předmět s id = 77, který je na hexu (0, 5).

Příklad 3: zprávy: „`Vojak10 > ALL : DIE`“

Co tento příklad vlastně znamená :

Voják10 ještě přes smrtí všem řekl, že zemřel.

7 Implementace

Pro implementaci této aplikace jsme si vybral jazyk C++ a výsledná hra je vykreslována DirektX9 – třída DxWindow – díky které je aplikace výhradně pro Windows. Ale už se pracuje na multiplatformním grafickém enginu. Po implementační stránce je tu pár důležitých funkcí či tříd

LiveInWorld

Funkce která řídí chod života ve světě. Je ve třídě World a volá obdobnou funkci ze třídy Object.

ArrayINT, ArrayPONT

Jsou to třídy, které reprezentují jedno či dvourozměrné pole int(resp. POINTů) a jsou realizovány jednorozměrným polem. Obsahují takřka identické funkce spravující pole. Využívá se jako optimalizace oproti dvourozměrným polím, či vector, list, a map. Ty mají dost funkcí a právě proto jsou asi i dost pomalé.

Interval

Třída charakterizující rozsah od min, do max hodno, Má funkce pro vrácení hodnoty, nastavení a vygenerování náhodného čísla z tohoto intervalu.

CircularError

Funkce ve třídě Agent, která kontroluje agenty proti zacyklení v plánování.

Generování lesa

Zde se používají dvě důležité funkce `GeneratePositionIntHex` , která generuje pozice do hexu(pokud se netrefí zkusí znovu, limit 5000 pokusů) a `IsNearObject`, která kontroluje, zda vložené objekty, tromy, nejsou příliš blízko u sebe(v zadanám rozsahu). Tedy, aby se nestalo, že ze 4 stromů budou 3 tak usebe, že to ani hezký nebude.

Mod hry

Pro hru *hráč* x *hráč*, *hráč* x *UI* a *UI* x *UI* jsou zkompilovaný zvlášť spustitelný programy.

8 Experimenty a jejich výsledky

Do začátku vývoje této aplikace je třeba provádět různé experimenty a pokusy. Zde jsou uvedeny některé z nich, které jsou něčím zajímavé nebo jinak důležité.

8.1 Experiment 1: Útok jen z důvodu

Hned v začátcích, kdy UI uměla jen chodit, brát a položit vlajku se stávalo, že agent prošel kolem nepřítele bez povšimnutí a mířil si to za nějakou vlajkou či jiným nepřítelem s vlajkou.

Jelikož měli nastavené, získat všechny vlajky a odnést je do svého tábora a neuměli ještě utočit na nepřítele, tak *útočili jen z důvodu*, jak to navenek vypadlo. Agent šel prostě sebrat vlajku, a to že ji

držel nepřítel jenom způsobilo, že byl nepřítel zabit(tehdy byl útok dost jednoduchý a to, že při malém náznaku útoku nepřítel hned zemřel – prakticky byl jen smazán z mapy pro jednoduchost). Připomínalo mi to chování zvířat, kde také zabíjejí jen z důvodu přežití či ze strachu(až na některé výjimky predátorů).

Toto se však spravilo prvním implementováním akce ATTACK.

8.2 Experiment 2: Agenti se nemohou pohnout

Při tomto experimentu, kde stále ještě nebyla implementována akce ATTACK a prohledání bylo implementováno návrhem 1 (prohledání s agenty), se po pár tazích agenti dostali do patové situace. Oba dva mohli jít jen po volných hexech a oba dva ještě na nepřítele bez vlajky neútočili. Problematická situace nastala, kdy jednotky zatarasily úzké průchody, tím ani jeden nenašel cestu ke svému cíli a hru bylo nutno vylepšit a restartovat. Díky tomuto experimentu bylo nutné prohledávat i skrze agenty.

8.3 Experiment 3: Agent nepochopitelně odchází s vlajkami od tábora

Situace nastala tehdy, kdy hráči měli za úkol sesbírat 6 + 6 vlajek (od každého 6), které byly různě umístěny po mapě. Pozoroval jsem, zda budou chodit pro každou vlajku zvlášť, anebo seberou víc vlajek ap ak se s nimi vrátí. V tomto momentu sbírali nejprve vlajky(priorita byla tak nastavena), když se už vrátili do tábora odložit vlajky, z nepochopitelných důvodů šli zase se všemi vlajkami, které měli u sebe zase pryč pro jinou. Příčina se však našla právě v prioritách, kdy vlajka držená agentem, který je ve svém táboře, byla považována za úspěšně donesenou do tábora a tedy se nemohla nastavit akce DROP.

Další zajímavost se stala hned po té, kdy naopak agent přišel s vlajkami do tábora a vlajky položil – vzal – položil. Zapříčinilo to dvojí nastavení akce DROP na tyto vlajky. Příčina byla opět ve vlajkách vlastnící agent, který je ve svém táboře.

A poslední věc, která vedla k dalšímu zlepšení, byl samotný systém sbírání vlajek. Teď pomocí vhodně nastavených priorit, sesbírají vlajky, které jsou poblíž a jinak je jdou položit do tábora.

8.4 Experiment: 4: „Lítali jako pominutí“

Opět úkolem bylo sesbírat 12 vlajek, jenže pro velký údiv se nemohli dostat ani k jedné. Zatím co v jednom kole vyšel směrem k jedné vzdálené vlajce, tak další kolo se otočil a šel zase pro vlajku na

druhé straně mapy. Příčinou byl malý, ale dost významný detail. Při třídění akcí a ukládání do pomocné struktury *map* se ohodnocená políčka ukládala obráceně, než bylo potřeba. Takže proto ve výsledku „lítali jako pominutí“, protože vždy si naplánovali tu nejvzdálenější akci.

Jednoduchým řešením bylo ukládat opačné hodnoty *hodnoty pole*. Tím se získá z pole s největší hodnotou nejzápornější číslo a tím i jednoduché reversní pořadí ve struktuře *map*.

8.5 Experiment 5: První boje

Už u sbírání vlajek se dalo zpozorovat, že nevyhraje vždy ten samý hráč a ani ne se stejným výsldkem. O to napínavější bylo očekávání, jak dopadnou první boje. Jediný náhodný výpočet a to při náhodném zvolení hexu v našem táboře, způsobil, že výsledek nebyl vždy předem jasný. Postupně s časem bylo třeba přidat i řádek `srand((unsigned int)time(NULL));`, které nastavuje – stanovuje náhodné počáteční číslo, od kterého se dál generují náhodná čisla.

8.6 Experiment 6: Útok přes spoluhráče

V situaci, kdy nebylo dost prostoru se boj odehrával dvěmi způsoby. Bud tím, že někteří agenti zbytečně obíhali les a tou svojí nepomocí v bojích, prohráli a nebo naopak tím, že agent útočící na nepřítele, který je hned za svým spoluhráčem, místo aby oběhl les a vypomohl z druhé strany, čeká až mu spoluhráč uhne.

Toto se povedlo až na pár výjimek vyřešit kvalitním hledáním minimální cesty. (Výjimka je u při velké přesile a množství vojáků)

8.7 Experiment 7: Zátěžový test na velké mapě

První zátěžový test se zaměřoval na výkonnost prohledávání mapy a proto byla zvolena velká mapa. Ze začátku u návrhů 3 a 4 pro hledání minimální cesty, trval výpočet kolem jedné sekundy. Což bylo už po optimalizaci převodu dvourozměrných polí na jednorozměrné. Naštěstí teď kvalitní prohledávání trvá výpočet cca 0.0 – 0.3 sekundy pro velkou mapu. A to 0.3 sekundy je horní hranice, která nastala jen výjimečně.

8.8 Experiment 8: Zátěžový test s kvantem vojáků

Druhý zátěžový test se zaměřoval na vykreslování, mnoho objektů(hodně vojáků, hodně stromů) a také, jak si poradí UI s kvantem vojáků. To už máme kvalitní prohledání mapy (to nejlepší) a UI mají už implementované priority, kde hraje roli vzdálenost, druh akce, jak moc je nepřítel daleko či zraněný. Jaké však milé převapení, když neměli moc velké problémy se uspořádat. Obě dvě strany se dostali po pár tazích doprostřed mapy (přes úzká místa na každé straně), kde se nakonec střeli "na život a na smrt". Utočnící nejprve utočili na nepřátele s poměrně menším životem, což bylo dobře. Problém byl ale v tom, že občas nastala situace, kde místo aby zaútočil na nepřítele přímo před sebou, šel delší cestou (než je zdrávo) aby si jednou bouchnul do zraněného. Bylo vidět, že je potřeba změnit priority. Vedlo to tedy k zavedení výhody zaútočit na nepřítele, který je přímo před agentem(tedy nemusím nikam chodit a využiju plný počet MP)

8.9 Experiment 9: Přesila

Posledním velkým experimentem byla situace, kdy modrý hráč se brání v 6-ti vojácích a červený má blízko roztroušených asi 10 vojáků a u sebe v táboře dalších 15. Jaký byl výsledek tohoto pokusu:

Jelikož červení začínali, hned se začlo bojovat u tábora modrého hráče. Stejně to dopadlo i když v prvním kole červení čekali, ukázolo se, že je jedno kdo začíná v tak velké přesile. I mesi tím, kdy několik vojáků bojovalo, zbytek došel (jakoby na výpomoc). Tímto se vytvořil chumel vojků, kde většina nemohla nic dělat, protože se nemohla nikam pohnout. A poté, co padl poslední modrý voják, byla na řadě modrá vlajka. Jeden voják se ji zmocnil a potom nic. Nemohl se přes své spoluhráče dostat do svého tábora. On nemohl projít a oni už neměli co na práci, protože jedinou vlajku, kterou mohli vzít má spoluhráč, takže jen tak stojí. Vznikla tedy patová situace a otázka "Jak ji vyřešit?".

V úvahu byly jen dva způsoby.

Jeden, že by si vlajku mezi sebou předali. a poslední voják stojící na okraji, by měl volnou cestu a nemusel by vlajku předávat, ale donést do tábora.

A nebo druhý způsob, kde agent chce jít po své naplánované cestě (která vede přes nejmenší počet agentů) a dotyčnému spoluhráči, řekne že překáží a ať uhne. Tak jak je popsáno v kapitole 6.5.1 UI1 Reaktivní agent – akce SIDESTEP.

Řešení je zřejmé, a to akce SIDESTEP, která se dá využít třeba i v situacích, kdy nic nenesu, ale chci jen projít. Což by u prvního způsobu zkrachovalo.

9 Závěr

Závěrem je nutno říci, že jsem byl s vybraným tématem plně spokojený jelikož se o podobnou problematiku zajímám už delší dobu, jen z jiného pohledu. Tuto práci tedy považuji pro mne za velmi přínosnou, protože jsem si zkusil implementovat samostatně fungující systém a poznal tak mou oblíbenou problematiku i po této stránce, což byl také hlavní důvod, proč jsme si vybral toto téma. Využil jsme znalosti z mnoha školních projektů, od tvorby okenní aplikace, přes grafiku, umělou inteligenci a převážně objektově orientovaný přístup s C++.

Cílem této práce je vytvořit malý samostatně fungující systém, a udělat tak další krok k mému většímu cíli a to vytvořit dost obecný a pokud možno přesný multiagentní systém, který by se dal použít nejen ve hrách ale i v dalších simulačních aplikacích. Pro neuvadající nadšení o toto téma hodlám rozhodně na tomto projektu dále pokračovat. Nejprve v rámci této strategické hry, jako rozšířit o infrastrukturu, počet hráčů, editor map a podobně. Rozšířit možné akce agentů a více se tak přiblížit skutečnému životu.

Co mě ale také láká, jsou různé složité algoritmy, jako například evoluční, ale prozatím je evoluční systém dosti vzdálený a tak si toto rozšíření nechám ještě na pozdější časy. Ale určitě je to dost lákavé téma spolu s teorií her. Když se to tak vezme, tak jsem také součástí evoluce, zatím tato zajímavá vědní oblast je pro mě dost složitá, ale o to víc je pro mne motivací pokračovat ve vývoji tohoto systému.

Literatura

[1] Hykšová, M.*: Teorie her a optimální rozhodování* [citováno 30. 04. 2008]. WWW: <http://euler.fd.cvut.cz/predmety/teorie_her/hry_menu.html>

[2] *Wikipedie: Otevřená encyklopedie: Teorie her* [online]. c2008 [citováno 30. 04. 2008]. WWW: <http://cs.wikipedia.org/w/index.php?title=Teorie_her&oldid=2443817>

[3] *Wikipedie*: *Otevřená encyklopedie: Šachy* [online]. c2008 [citováno 10. 05. 2008]. WWW: <http://cs.wikipedia.org/w/index.php?title=%C5%A0achy&oldid=2501129>

[4] *Wikipedie: Otevřená encyklopedie: Počítačová hra* [online]. c2008 [citováno 10. 05. 2008]. WWW: <http://cs.wikipedia.org/w/index.php?title=Po%C4%8D%C3%ADta%C4%8Dov%C3%A1_hra&oldid=2540098>

[5] *Wikipedie: Otevřená encyklopedie: Olympijské hry* [online]. c2008 [citováno 10. 05. 2008]. WWW: <http://cs.wikipedia.org/w/index.php?title=Olympijsk%C3%A9_hry&oldid=2561133>

[6] *Wikipedie: Otevřená encyklopedie: Rytíř* [online]. c2008 [citováno 10. 05. 2008]. WWW: <http://cs.wikipedia.org/w/index.php?title=Ryt%C3%AD%C5%99&oldid=2269259>

[7] Kubík, A.*: Inteligentní agenty, tvorba aplikačního software na bázi multiagentních systémů*, 2004

[8] Monade*: Hra CheesDuel*, 2006 WWW: <http://www.monade.cz/item.php?item=8>

Seznam příloh

Příloha 1. CD se zdrojovými texty a manuálem.