VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

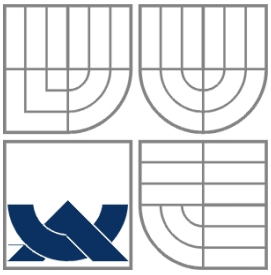
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SKRIPTOVATELNÝ BOJOVÝ SYSTÉM PRO TAHOVOU POČÍTAČOVOU HRU

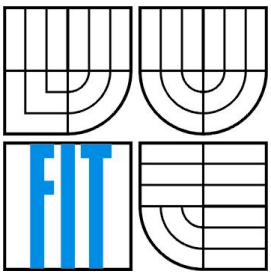
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ONDŘEJ KANICH
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SKRIPTOVATELNÝ BOJOVÝ SYSTÉM PRO TAHOVOU POČÍTAČOVOU HRU

SCRIPTABLE FIGHT SYSTEM FOR TURN-BASED COMPUTER GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ KANICH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HUSÁR

BRNO 2012

Abstrakt

Tato bakalářská práce se zabývá vytvořením soubojového systému a umělé inteligence pro něj. Cílem tohoto projektu je dát nepříliš zkušeným programátorů možnost navrhnout i implementovat vlastní umělou inteligenci. Jako platforma pro zabudování této práce je zvolen projekt DrdSim. Jako předloha pro vytvářený systém je využít rozšířený soubojový systém Dračího doupěte. Umělá inteligence je implementována pomocí speciálního jazyka platformy DSL. Využívá přístup reaktivních agentů.

Abstract

This bachelor's thesis deals with creation of fight system and artificial intelligence for it. The goal of this project is to give an opportunity for low-experienced programmers to design and implement their own artificial intelligence. The Project DrdSim is used as a platform for this thesis. Extended fight system of the Dragon's lair (Czech version of D&D) is employed as a model for created system. Artificial intelligence is implemented by DSL, special language of DrdSim platform. An approach of reactive agents is used for artificial intelligence.

Klíčová slova

umělá inteligence, počítačové hry, DrdSim, Dračí Doupě, soubojový systém

Keywords

artificial intelligence, computer games, DrdSim, Dragon's Lair, fight system

Citace

Ondřej Kanich: Skriptovatelný bojový systém pro tahovou počítačovou hru, bakalářská práce, Brno, FIT VUT v Brně, 2012

Skriptovatelný bojový systém pro tahovou počítačovou hru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Adama Husára.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Ondřej Kanich
16.5.2012

Poděkování

Chtěl bych poděkovat panu Ing. Adamovi Husárovi za to, že vedl tuto práci a panu Ing. Janovi Hranáčovi za konzultace i samotné vytvoření platformy DrdSim.

Obsah

1 Úvod

Počítačové hry jsou v dnešní době téměř v každé domácnosti. Nedílnou součástí většiny těchto her je umělá inteligence. Často pak uživatelé diskutují o tom, jak jsou protivníci v té či oné hře inteligentní nebo častěji hloupí. Většina z nich se ovšem nezajímá o to, jak umělá inteligence v jejich oblíbené hře funguje. Stačí, aby se jednou nezachovala podle jejich předpokladu, a hned je jí co vytýkat. Tato práce dává možnost si jednoduchým způsobem vytvořit vlastní umělou inteligenci. I nepříliš zkušený programátor si může návrh a implementaci takové metody vyzkoušet. A tímto způsobem získat i jistý nadhled o tom, podle jakých kritérii umělou inteligenci hodnotit.

Náplň této práce se zaměřuje na dva aspekty. Prvním z nich je návrh a tvorba soubojového systému a dále pak jeho integrace do zvolené platformy. Druhým aspektem je vývoj nástrojů pro vytváření a ukázku umělé inteligence pro tento bojový systém.

Tvorba originálního soubojového systému, který by byl vyvážený, netrpěl větším množstvím chyb a kombinací prvků, které se dají zneužít pro téměř jistě neporazitelnou postavu (hráčova představitele v tomto systému), je však nesnadná úloha. Zkušenosti jiných počítačových i deskových her ukazují, že taková činnost je otázka několika let (například u hry Dungeons & Dragons uběhly 3 roky od jejího vydání do prvního turnaje [1]).

Pro zjednodušení tohoto dlouhého procesu bude navrhovaný soubojový systém napodobovat rozšířený soubojový systém Dračího doupěte. K tomu se také váže platforma, do které bude systém zabudován. Tou je projekt DrdSim. Projekt DrdSim je charakterizován několika svými specifickými pravidly. Implementaci tak bude potřeba těmto pravidlům přizpůsobit. Pokud by se pravidla nedodržela, byl by projekt nekonzistentní.

Existuje mnoho způsobů, jak vytvořit umělou inteligenci. V této práci byla účelově zvolena jednoduchá metoda, a to proto, aby i nepříliš zkušený programátor mohl tuto inteligenci upravit či doplnit, aniž by musel studovat náročné teorie. Takový přístup je v souladu s filozofií platformy DrdSim, kde je pro podobné účely vyvinut speciální jazyk DSL. Bylo tedy vhodné na tento jazyk navázat a doplnit jej o další funkce, aby bylo možné v něm umělou inteligenci implementovat.

Ve druhé kapitole jsou popisovány informace o hře Dračí doupě a o platformě DrdSim. Třetí kapitola se zabývá různými způsoby zpracování a vytváření umělé inteligence. Ve čtvrté kapitole se věnuji návrhu a vytvoření soubojového systému a dále také návrhu a rozpracování způsobu implementace jednotlivých metod umělé inteligence a jejich zabudování do DrdSimu. Pátá kapitola pojednává o samotné implementaci, včetně testování a dosažených výsledků.

2 Dračí doupě a DrdSim

Platforma DrdSim i soubojový systém vychází z deskové hry Dračí doupě společnosti ALTAR. Pro porozumění této práci je zapotřebí znát některé jejich charakteristiky a popis. Vzhledem k tomu, že desková hra ovlivňuje nejen soubojový systém, ale i samotnou platformu, je popsána první. Důraz je kladen hlavně na soubojový systém (základní i rozšířený). Dále, hlavně pro část umělé inteligence, je důležitá vnitřní stavba a myšlenky, které stály za tvorbou platformy DrdSim. Prostor bude také pro popis jazyk DSL a charakterizaci jeho vlastností.

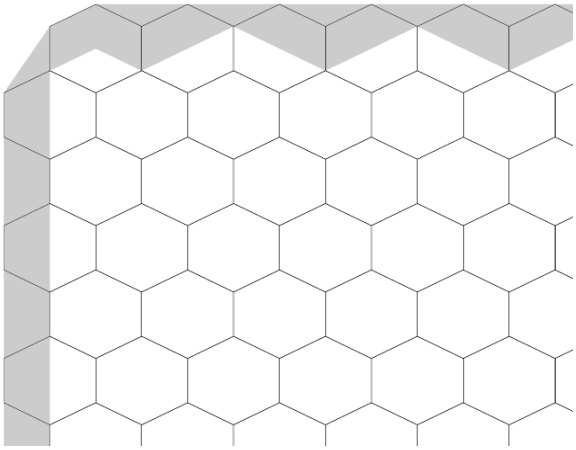
2.1 Dračí doupě

Dračí doupě je česká desková hra na hrdiny (RPG, role playing game) vytvořená firmou ALTAR. Začátkem roku 1990 chtěla skupina studentů přeložit v té době velmi úspěšnou a revoluční hru Dungeon's and Dragons, ale vlastník práv jim to nedovoluje. Proto vzniká společnost ALTAR a jejich česká verze Dračí doupě [2]. Hra modeluje středověký svět s prvky magie, ale veškeré interakce mezi hráči a světem v rámci těchto pravidel zprostředkovává prostřednictvím tzv. *pána jeskyně*. Je to obvykle jeden hráč, který řídí celý okolní svět.

Hráči, vyjma pána jeskyně, si vytvoří postavu, za kterou poté hrají. Důležité vlastnosti postav, které najdeme uvedeny v pravidlech, jsou vyjádřeny číselným stupněm. Pro představu, mezi základní vlastnosti patří síla, obratnost, odolnost, inteligence, charisma a úroveň postavy. Úroveň je nejdůležitější vlastností, protože určuje vyzrálost, stáří a moc postavy.

Pravidla Dračího doupěte jsou rozdělena podle úrovní postav na základní, pokročilá a pro experty. Soubojový systém je již v pravidlech pro začátečníky rozdělen na základní a rozšířený. Základní je vhodný pro hráče, kteří se více soustředí na hraní postav a nebo pro začátečníky. Rozšířený, který je detailnější a tedy i realističtější, avšak zabírá při hře mnoho času, je určen pro pokročilé hráče. Oba dva systémy jsou tahové, postavy se v různém pořadí střídají. Základní soubojový systém nechává rozestavení postav a jejich pozice volně. Řeší hlavně útok, obranu a iniciativu (tedy to, kdo bude začínat). Rozšířený soubojový systém přidává přesné pohyby po tzv. hexovém papíře (viz obrázek 2.1). Také určuje počet akcí za jeden tah podle typu postav. Tím zůstává poněkud méně volnosti pro fantazii hráčů. Pro představu například myšlenka lezení na strom bude dělat v pevně určeném 2D (dvoudimenzionálním) světě potíže.

Zlatým pravidlem Dračího Doupěte je však „pán jeskyně má vždy pravdu“. Tedy si může upravit všechna pravidla i hody kostkou, pokud to uzná za vhodné. Měl by tak samozřejmě činit jen ve výjimečných případech. Toto zlaté pravidlo přidává možnost, jak mohou hráči využít svých schopností takový způsobem, který nebyl původně zamýšlen, ale přesto dává smysl. [3]



Obrázek 2.1: Šesterečkovaný neboli hexový papír

2.2 DrdSim

Platforma DrdSim neboli Simulátor dračího doupěte vznikla jako diplomová práce na půdě FIT VUT v Brně. Návrh se soustředil na vytvoření aplikace, která by dovolovala jednoduchým způsobem upravovat obsah samotné hry. Pro tyto úpravy je zde jazyk DSL. Ten byl navržen tak, aby plnil i výukovou roli.

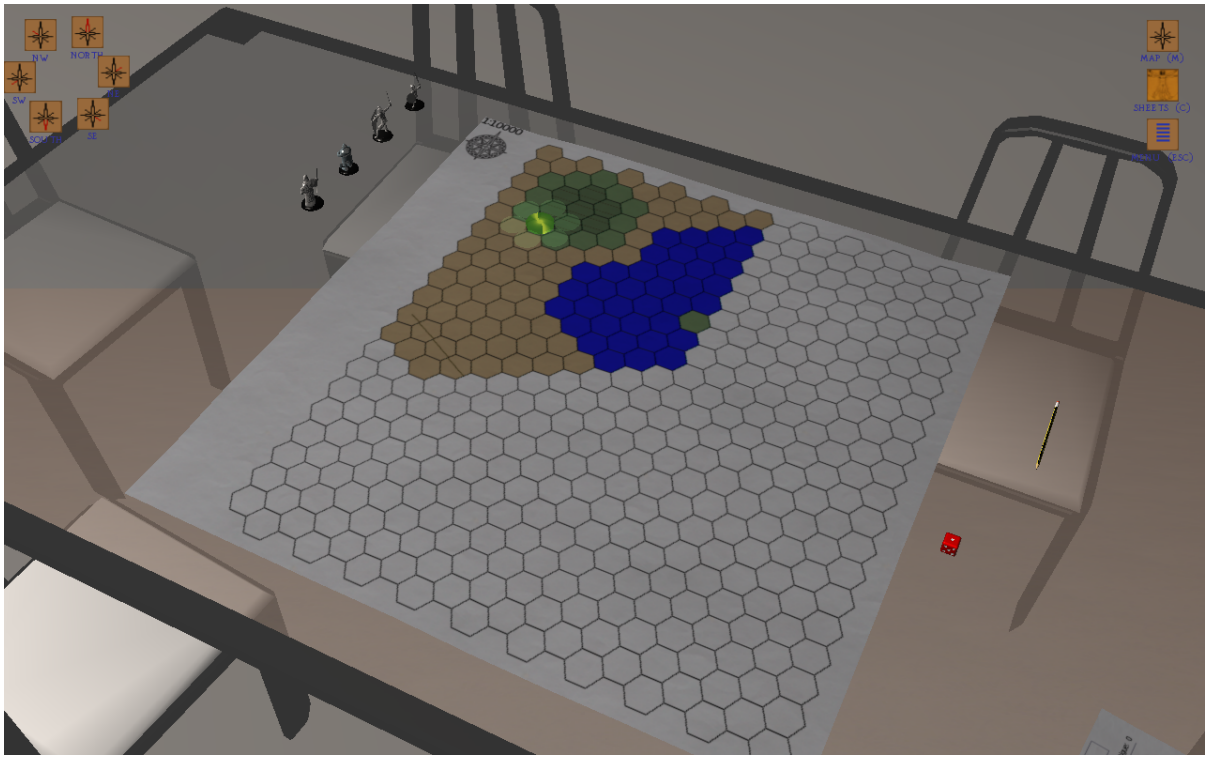
DrdSim je naprogramován v jazyce C++. Pro grafický výstup a GUI (Graphical User Interface, grafické uživatelské rozhraní) využívá 3D knihovnu OpenGL a nadstavbu Open Inventor (implementaci Coin2). Tato knihovna je sice určená spíše pro desktopové aplikace, ale platforma již obsahuje několik objektů pro její vhodné použití a případná řešení pro obvyklé požadavky. Rozhraní, mezi operačním systémem a touto knihovnou, je tvořeno za pomoci speciálního modulu SoSDLRenderArea. Díky tomu je grafický výstup Open Inventoru implementován pomocí knihovny SDL. Open Inventor využívá pro vytváření scén grafy. Název napovídá, že hra simuluje výše zmíněnou hru Dračí doupě. Platforma získala povolení od firmy ALTAR pro použití pravidel verze 1.5. Můžete si všimnout, že v citacích se objevuje verze 1.6, avšak podle [4] se nezměnilo nic, co by se dotýkalo soubojového systému tedy nedochází k porušení tohoto povolení.

Jak tedy DrdSim funguje? Po spuštění samotné hry se dostává hráč na mapu daného příběhu (mise, části tažení nebo kampaně), kde je mu ponechána absolutní volnost. Oproti Dračímu doupěti tady chybí zásadní prvek pána jeskyně. Ten je zde tvůrcem obsahu (vývojářem). Musí celou misi nebo celé tažení (sled misí) vytvořit a naprogramovat. Místo toho, aby na papír kreslil mapy jeskyní, dlouze rozmisťoval nestvůry a jejich poklady, všechny tyto prvky postupně programuje za pomocí nástrojů DrdSimu. K tomu slouží generátor map, ale hlavně jazyk DSL, ve kterém má pán jeskyně velkou volnost. Hráč pak již pouze hraje výsledný produkt. Pro pána jeskyně je tímto procesem práce

zjednodušena, ale hráčům ubírá možnosti. To, co nebylo v misi zamýšleno, prostě nejde provést. Hra se tak stává více deterministická.

Jako příklad uvedu situaci, kdy se hráči potřebují dostat přes přísně střežené místo. Pán jeskyně navrhne a implementuje několik řešení plížení, vyjednávání a souboj. Hráče však může napadnout zneškodnit jednoho strážného a využít jeho oblečení k přechodu oblasti. Tedy něco mezi řešením plížení a souboj. V deskové hře by se musel uchýlit k improvizaci. V DrdSimu tuto možnost nemá.

Jazyk DSL v sobě prakticky obsahuje jazyky dva. Ty jsou ovšem úzce propojené. Prvním z nich je deklarativní jazyk. V tomto jazyce jsou zapsány definice tříd objektu a definice jejich atributů. Lze využít dědičnosti při tvorbě třídy a jako hodnotu atributu lze zapsat celé číslo, řetězec (s možností lokalizovaného řetězce) a kód procedurálního (druhého) jazyka. Tento způsob spolupráce zjednodušuje využití obou jazyků. Druhý je procedurální jazyk. Jeho jediným stavebním kamenem je definice funkce. Ta je velmi podobná jako v jazyce C. Celkově je tato část jazyka DSL jazyku C velmi podobná. Ve zkratce se liší jen v neexistenci ukazatelů, for a do-while cyklu. Navíc je pole podobné poli v PHP, tedy adresovatelné nejen číslem, ale i řetězcem. Tímto polem, jsou také nahrazeny struktury. Klíčové slovo global, které určí proměnnou, jejíž obsah bude uložen při ukládání hry. Přesný popis obou jazyků včetně příkladů jejich použití je v příloze B. [5]



Obrázek 2.2: Ukázka herní obrazovky platformy DrdSim.

3 Metody umělé inteligence

Velkou částí mojí bakalářské práce je umělá inteligence. Tento pojem je již přes 60 let starý (poprvé s ním přišel John McCarthy v roce 1955 [6]). I přes jeho stáří není tento pojem přesně vymezen. Definice se obvykle točí kolem neschopnosti pozorovatele rozeznat počítač od člověka (například Turingův test). Je třeba si však položit otázku, zda-li je u umělé inteligence pro počítačovou hru takové chování žádoucí a nutné. Jestli by měl být každý soupeř co nejvíce podobný člověku. Což znamená, že by měl mít schopnost se učit, samostatně se rozhodovat, komunikovat a reagovat (klidně i subjektivně) na okolí. Ať už je žánr hry jakýkoliv, pokud nejde o nějaký velmi specifický, je takováto dokonalá inteligence až nežádoucí. Představme si například situaci v akční hře, kde hráč má za úkol zlikvidovat několik soupeřů. Toto je v počítačových hrách běžná situace. Doplňme, že tito soupeři se mezi sebou perfektně dorozumívají, koordinují své akce, odvádějí hráčovu pozornost do té doby, dokud jej perfektně a synchronizovaně neobklíčí. Pokud hráč v tomto případě vylepší své krytí nebo použije lepší taktiku, tak se umělá inteligence přizpůsobí. I kdyby hráč uspěl, do příště se umělá inteligence poučí. Pokud ji budou sdílet všichni nepřátelé, nemá hráč po několika pokusech žádnou šanci na vítězství. Takový přístup není možný, jelikož umělá inteligence by měla být pro hráče výzva a ne nepřekonatelná překážka. A vzhledem k tomu, že téměř veškeré využití v počítačových hrách se týká právě jisté formy souboje, řeší se tento problém téměř ve všech žánrech her. [6]

Použitá metoda umělé inteligence může být ovlivněna také charakterem hry. Teorie her formálně rozděluje hry v závislosti na různých aspektech jejich průběhu. U her v normální formě hrají hráči zároveň nebo alespoň neví, jak bude hrát jejich protivník. Extenzivní (rozšířená) forma hry neboli také dynamické hry se hráči střídají po tazích. Další dělení je na hry kooperativní a nekooperativní. Hráči spolu v případě kooperativní hry mohou komunikovat. U nekooperativní hry je tomu naopak. Existují hry s nulovým a nenulovým součtem. Hry s nulovým součtem mohou dopadnout pouze dvěma způsoby, jsou jimi výhra a prohra. Ve hrách s nenulovým součtem tomu tak není. Poslední dělení je podle míry informovanosti hráčů. Hráč ve hře s úplnými informacemi ví vše o stavu hry, stejně jako ostatní hráči. Hry s neúplnými informacemi naopak neposkytují všem hráčům stejné informace. [7]

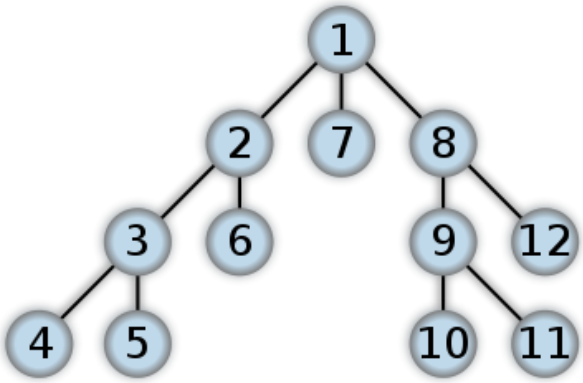
Existuje celá řada metod umělé inteligence. Jestliže vyřadíme ty, které nejsou vůbec vhodné pro použití v počítačových hrách, dostáváme se k několika základním typům. Jsou jimi metody prohledávání stavového prostoru, strojového učení a genetické algoritmy. [6]

3.1 Metody prohledávání stavového prostoru

Problém neboli řešená úloha je při použití těchto metod rozdělena do stavů a operátorů. Operátor je funkce, která mění jeden stav na druhý. Tedy problém nalezení cesty z místa A do místa B by byl

zjednodušen na stavy, což jsou jednotlivá města a operátory, které představují jednotlivé cesty mezi městy. Tímto způsobem dostaneme orientovaný graf. V grafech často stavy nazýváme uzly. Poslední fází je určení počátečního uzlu tedy města, ve kterém se nacházíme a cílového stavu tedy města, do kterého se chceme dostat. Cílových stavů může být obecně i více, ale také mohou být určeny jen například podmínkami. Teď jen tedy zbývá nalézt cestu v tomto grafu. K těmto metodám se také váže pojem expanze uzlu. Expandováním máme na mysli využití všech operátorů na daný stav. Často se také určuje maximální hloubka metody. Ta omezuje metodu pro případy, kde je stavový prostor příliš velký nebo by se metoda mohla zacyklit. V tahových hrách je tento způsob velmi snadno použitelný. Každý povolený tah je operátor a situace mezi tahy určuje stav.

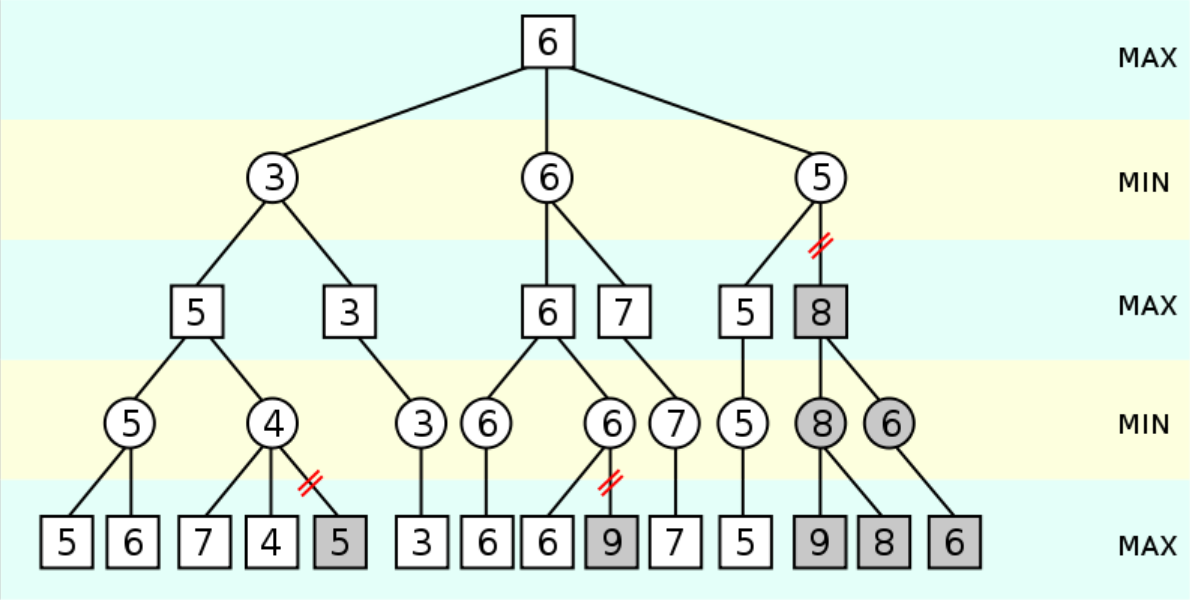
Mezi nejjednodušší algoritmy tohoto typu patří prohledávání do hloubky (DFS, depth-first search). Tato metoda prohledává, což znamená, že expanduje první uzel s využitím prvního operátoru tak dlouho, než se dostane do maximální hloubky nebo k uzlu, kdy už nejde operátor použít. Pak se vrátí o jeden stav a využije další operátor (přehledně na obrázku 3.1). Takto iteruje tak dlouho, než projde všechny stavy nebo narazí na cílový stav. Problémem tohoto přístupu je, že při počítačové hře by byl cílový stav výhra. Tento stav bude obvykle na začátku hry téměř nedosažitelný (v závislosti na složitosti hry). Tento problém se dá vyřešit ohodnocením každého stavu v maximální hloubce. Následně se upraví algoritmus tak, aby hledal cílový stav. Pokud jej nenajde, tak ukáže cestu ke stavu z nejlepším ohodnocením. Do stavu hry také zasahuje soupeř. Metodu je tedy třeba výrazně poupravit.



Obrázek 3.1. Pořadí procházení stavů DFS ([8])

Takto upravená a použitelná metoda pro hraní her se nazývá Minimax. Je použitelná v tom případě, pokud se jedná o hru dvou protihráčů, kteří se po jednotlivých tazích střídají. Graf, který při této metodě prohledáváme je tzv. *AND/OR graf*. Řekněme, že na obrázku 3.1 je znázorněn problém AND/OR grafem, kde uzel (stav) 1 je OR grafem a uzel 9 AND grafem. Aby měl uzel 1 řešení, musí být řešitelný kterýkoliv z uzlu 2, 7, 8. Naproti tomu, aby byl řešitelný AND graf, tedy uzel 9, musely by být zároveň řešitelné uzly 10 a 11. Metoda Minimax je prohledávání AND/OR grafu do hloubky, kde hráč, který je na tahu (tedy problém OR, hráč si může vybrat svůj tah) se snaží maximalizovat

zisk. Poté je na řadě soupeř (problém AND předpokládáme, že zahraje nejlepší svůj tah), který se snaží naopak zisk minimalizovat. Metoda zkoumá tahy do maximální hloubky nebo do nalezení cílového uzlu (vítězství). V této maximální hloubce pak nastává prostor pro ohodnocující funkci, která by měla velmi rychle ohodnotit daný stav a vrátit tuto hodnotu). Její pomocí se zjišťuje hodnota zisku. Vítězství, případně porážka se pak označuje plus resp. mínus nekonečnem (příklad na ilustračním obrázku 3.2). Tento algoritmus prochází mnoho uzlů zbytečně. Existuje tedy jeho optimalizovaná verze s názvem Alfa-Beta. Ta verze využívá při každém tahu tzv. alfa a beta řezů. Ty zabraňují expanzi a průzkumu uzlů, u kterých je nesmyslné je prozkoumávat (na obrázku 3.2 [9] je



Obrázek 3.2.Ukázka metody Minimax s naznačením výhodnosti Alfa-Beta řezů ([9])

ořezání červeně naznačeno). [10]

Další metoda využívající podobného principu je metoda Monte Carlo (nebo také Monte Carlo tree search). V některých hrách je počet stavů tak rozsáhlý, že prohledávání všech možností, a to i do relativně malé hloubky zabírá příliš mnoho času. Problém také nastává, pokud není možné udělat rychlou a do jisté míry přesnou ohodnocující funkci. V těchto případech je pak čas, který je nutný k zjištění výhodného tahu příliš dlouhý a je potřeba využít jinou metodu. Monte Carlo obecně funguje tak, že místo ohodnocující funkce pro každý tah potom udělá jisté množství pokusů (klidně i tisíce pokusů). Pokus spočívá v použití náhodných tahů tak dlouho, dokud hra neskončí. Úspěšnost této sady pokusů je pak vlastně ohodnocení daného tahu. Metoda je velmi dobrá ve strategii (rozhled do budoucna), ale špatná v taktice (aktuální situace). To se projeví, pokud například existuje jeden soupeřův tah, který danou situaci obrátí v jeho prospěch a v sérii náhodných pokusů na něj nedojde. Proto se Monte Carlo tree search v praxi často upravuje. Místo náhodných tahů se použijí tahy s nějakou váhou (tedy nejde o čistě náhodné tahy, ale o více či méně pravděpodobné tahy podle jejich síly). Tím stoupne síla a kvalita ohodnocení a tedy i celé metody. [11]

3.2 Strojové učení

Metody strojového učení řeší problémy za pomocí naučeného chování. Pro naučení chování se využívají tři kategorie metod. Prvním z nich je učení s učitelem. To pracuje tak, že se metodě, která se má učit, předloží vstupní a očekávaná výstupní data. Metoda se pak učí vytvořením co nejvěrohodnější aproximace funkce transformující dané vstupy na výstupy. Učení bez učitele vyhledává podobnosti ve vstupních datech. Poslední metoda je motivované neboli posilované učení. Tento způsob spočívá v tom, že metoda provede akci a prostředí ji vrátí zpětnou vazbu. Ta je pro metodu „odměnou“, která může být pozitivní i negativní. Metoda se snaží úpravou svého chování dosáhnout co největší hodnoty všech odměn.

3.2.1 Rozhodovací strom

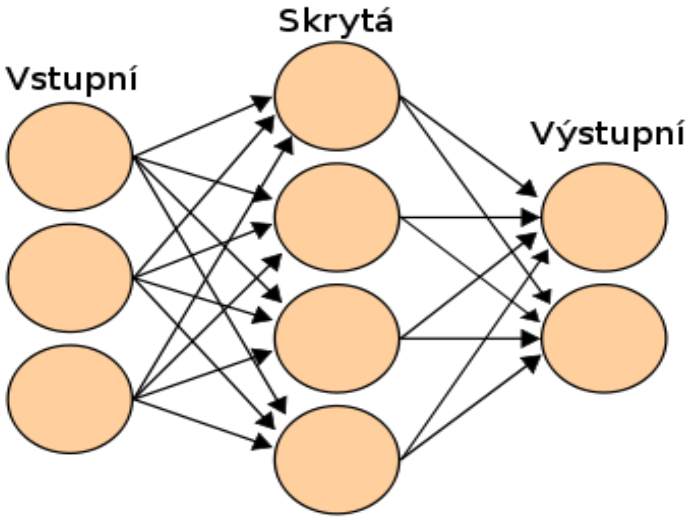
Je to jedna z nejjednodušších metod strojového učení. Pracuje na bázi učení s učitelem. Z atributů vstupních dat se vytvoří strom, který na základě konkrétních hodnot rozhodne o příslušné nejvhodnější akci.

3.2.2 Neuronové sítě

Neuronové sítě jsou celé velké odvětví metod strojového učení. Jak již název napovídá, neuronové sítě se snaží částečně kopírovat strukturu lidského mozku. Neuron byl matematicky popsán již v roce 1943 [12]. Za tu dobu se neuronové sítě dostaly do mnoha odvětví lidské činnosti. Využívají také učení s učitelem. Jejich velkou výhodou je ovšem generalizace (zevšeobecňování). Je schopna správně reagovat i na vstupy, na které nebyla trénována. I zde je jisté omezení, čím více příkladů umí neuronová síť pojmout, tím horší je její schopnost generalizace.

Jak neuronová síť pracuje? Neuron je prvek mající jeden výstup a teoreticky neomezený počet vstupních hodnot a jejich vah. Běžně se používá vícevrstvá neuronová (perceptonová, to je speciální druh neuronu) síť. Ta má obecně tři vrstvy: vstupní, skrytou a výstupní (viz obrázek 3.3 [13]). Perceptrony jednotlivých vrstev jsou propojeny jako biparitní graf. To znamená, že výstup neuronu jedné vrstvy jde na vstup všech neuronů druhé vrstvy). Počet skrytých vrstev a perceptronů v jednotlivých vrstvách je parametrem sítě. Prakticky se určuje podle zadaného problému na základě heuristik. Pro získání výsledku dáme vzor na vstupy sítě, podle jisté funkce a vah se sítí šíří signál. Z výstupní vrstvy pak přichází rovnou výsledek.

Síť se učí pomocí gradientní metody. Nastavíme váhy na náhodnou veličinu a pak počítáme kumulativní chybu. Na jejím základě upravíme váhy. Nebo se také může učit pomocí zpětného šíření (backpropagation). Ta přenáší informaci o chybě z výstupní vrstvy zpět až po vrstvu vstupní. [12]



Obrázek 3.3: Příklad vrstev neuronové sítě a propojení jednotlivých neuronů ([13])

3.3 Genetické algoritmy

Metody genetických algoritmů teoreticky patří mezi metody prohledávající stavový prostor, ale dají se využít i pro posilované učení. Z tohoto důvodu jsem je nechal jako zvláštní kapitolu. Genetické algoritmy jsou inspirované Darwinovou teorii přirozeného výběru. Většina pojmů a názvu tedy pochází z genetiky. Genetické algoritmy jsou charakteristické tím, že jednotliví jedinci jsou reprezentováni chromozomy, což jsou struktury ve tvarů řetězců). Tyto řetězce jsou tvořeny alelami (symboly, vlastnosti). Každého jedince lze ohodnotit (určit jeho kvalitu) pomocí účelové (fitness) funkce. Ta má na tyto algoritmy největší vliv, protože je to vlastně její extrém, který hledáme. Základním principem je tedy upřednostňovat kvalitnější jedince.

Na úvod je třeba inicializovat prvotní generaci. Inicializace bývá obvykle náhodná, ale měla by však pokrývat celý stavový prostor. Musí obsahovat dostatečný počet informací, jinak může metoda dojít k neuspokojivým výsledkům. Podobně jako v genetice jsou zde tři operátory nad jedinci. Jsou jimi selekce, křížení a mutace.

Selekce zajišťuje výběr lepších jedinců. Metoda však vybírá obvykle na základě pravděpodobnosti, která se určuje podle fitness funkce. Je tedy možné, aby se i slabší jedinec dostal do další generace. Ruletová selekce je přesně ta, kde fitness funkce a pravděpodobnost výběru jsou přímo úměrné. Tato metoda ale pro svou správnou funkci potřebuje velké populace. Upravenou verzí je deterministický výběr. Ten probíhá ve dvou kolech. V prvním se určí pravděpodobnost přežití

podle fitness funkce. Ta se vynásobí velikostí populace a výslednému číslu se odřízne desetinná část. Zbývající celá část určuje počet jedinců tohoto typu v další generaci. Porovnáním desetinných částí získáváme zbytek populace. Na závěr popíšu poněkud odlišnou metodu turnajové selekce. Ta probíhá tak, že se uspořádá několik turnajů, do kterých jsou náhodně přiděleni jedinci. Vítězové turnajů jsou vybráni do další generace. V této metodě můžeme zvětšením velikostí turnaje snadno vytvořit nátlak pro výběr silnějších jedinců.

Vybraní jedinci jsou pak s jistou pravděpodobností kříženi. Ti, co jsou určení pro křížení pak dostanou náhodně k sobě „partnera“. Poté se například metodou jednobodového křížení společně zkříží. Tato metoda funguje tak, že se určí křížový bod v chromozomu. Následně jsou pak vytvořeni dva potomci. Každý je tvořen dvěma polovinami svých rodičů (každý z potomků je jiný).

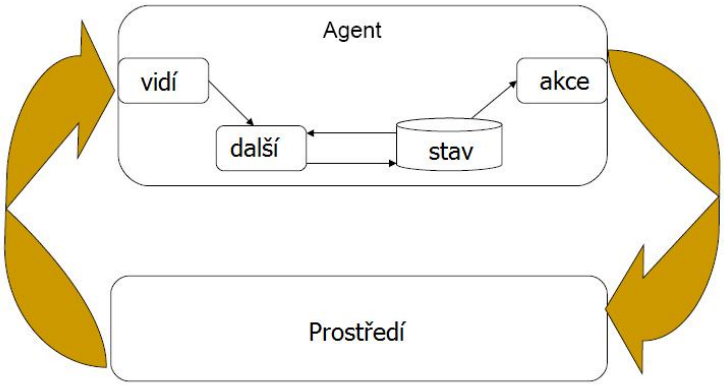
Poměrně málo vybraných jedinců je pak zmutováno. Mutace obvykle pracuje tak, že procházíme chromozom bit po bitu a náhodným pokusem s jistým prahem určíme, zda-li se má bit invertovat.

Po vytvoření dostatečně dlouhé řady generací pak dostáváme výsledky blízké maximu účelové funkce. Kolik generací bude potřeba závisí na zvoleném problému i všech parametrech genetického algoritmu (velikost populace, pravděpodobnosti křížení a mutace). Je důležité také zmínit, že vzhledem k náhodnosti procesů při každém spuštění stejného genetického algoritmu se stejným nastavením dostáváme úplně jiný průběh i výsledky. Pokud oba algoritmy nedojdou až k maximu účelové funkce, pak by byly výsledky stejné. [14][15]

3.4 Další metody

V předchozích podkapitolách jsou uvedeny obvyklé i méně obvyklé metody umělé inteligence. V realitě se však používají mnohem jednodušší metody. Často se setkáváme s úplně naskriptovanými protivníky, kteří se drží svého scénáře (skriptu) a striktně ho dodržují. Po jisté úvaze nad maximální inteligencí umělé inteligence jsem zjistil, že je propastný rozdíl mezi výše zmíněnými metodami umělé inteligence a právě skriptů. Musí přece existovat metoda, které je někde mezi těmito dvěma póly. Ukazuje se, že takový způsob existuje. Není však spojen s názvem umělá inteligence. Je to přístup, který využívají roboti. Robot je stroj, který pomocí senzorů vnímá své okolí a je schopen s tímto okolím nějak pracovat. Tedy reagovat na podněty, konat nějakou činnost. Abstrakce a zobecnění takového robota se v informatice nazývá agent. Definice agenta podle literatury [16] je: „*Agent je počítačový systém, situovaný v nějakém prostředí a schopný samostatně jednat v tomto prostředí tak, aby splnil svůj úkol.*“ Pokud pojmu každou postavu jako jistého takového agenta dostanu distribuovanou umělou inteligenci.

V tomto případě se jedná o reaktivního agenta, což je agent který koná akce na základě podnětů z okolí a svého vlastního stavu (viz obrázek 3.1). Nejvýznamnějším představitelem reaktivního přístupu je subsumpční (vzájemně potlačující) architektura. Vytvořil ji Rodney Brooks a prohlásil o ní, že je to paralelní a distributivní formalismus pro propojení senzorů a efektorů robota (tedy zobecněně i agenta). Agent má v této architektuře své chování rozdělené do vrstev. Čím nižší vrstva,



Obrázek 3.4: Reaktivní agent se stavem [15]

tím vyšší priorita vrstvy. Na každé této vrstvě je rozšířený konečný automat. Je rozšířený o časovač, po jehož vypršení se taktéž může změnit stav automatu. Tato architektura má 4 základní vlastnosti. Jsou jimi situovanost, tělesnost, inteligence a emergence (vznik složitého chování z jednodušších prvků). Situovanost znamená, že agent je nějakým způsobem ztělesněn v prostředí a je schopen svými senzory toto prostředí vnímat. Tělesnost znamená, že agentova zkušenost se světem je přímá. Je součástí interakce mezi ním a světem. Inteligence je určena střetem agenta s prostředím. Emergencí je inteligence systému, která vzniká ze sítě interakcí mezi světem a vrstev, z nichž se agent skládá. Nevýhoda toho přístupu je náročné určení exaktní akce, kterou agent zvolí, pokud je agent složitý (to znamená, že má mnoho vrstev). Každá tato vrstva obvykle reprezentuje jeden styl chování (například pokud můžeš, zaútoč). [14][15][16]

4 Návrh a inovace

S těmito informacemi je možné se pustit do návrhu celé práce. Na vývoj soubojového systému, je připojeno mnoho dílčích částí. Jde například o grafickou podobu, fázi hledání nepřítele, přechody mezi souboji atd. Také umělá inteligence, která je napojena na tento systém, potřebuje nějaký grafický výstup. Projekt by samozřejmě mohl vzniknout samostatně, ale jednalo by se o pouhou demoverzi bez většího využití. Bylo tedy nutné rozhodnout, zda práce vznikne pouze jako nějaké demo nebo využije již existující platformu, které takový systém chybí. Rozhodl jsem se tedy využít platformy DrdSim. Ta má téměř všechny tyto náležitosti hotové. Pro soubojový systém i umělou inteligenci je zapotřebí definovat soupeře. V platformě DrdSim na to jsou prostředky v jazyce DSL. V tomto jazyce je nutné vytvořit šablonu takového protivníka včetně několika konkrétních.

4.1 Soubojový systém

Ze zadání vyplývá, že soubojový systém má být tahový. Důležitý je také výběr předlohy pro tento systém nebo vytvoření originálu. Vytvářet tahový soubojový systém od nuly je náročná činnost. Výsledek takové práce není nikdy napoprvé dokonalý. To dokazují i velké herní společnosti, které mají nemalé rozpočty pro své hry a i přesto nejsou jejich systémy perfektní. Je třeba zvolit vhodnou předlohu pro vytvoření soubojového systému. Vzhledem k výše zmíněné problematické a zdlouhavé tvorbě jsou licence na tyto systémy obvykle zpoplatněné. Platforma DrdSim však již má povolení na používání pravidel Dračího doupěte (přesněji na verzi 1.5). Bylo by škoda tohoto povolení nevyužít. Jak již dříve bylo řečeno znova ujasním, že verze 1.6 se podle [4] neliší v ničem, co by se dotýkalo soubojového systému, tedy nedochází k porušení tohoto povolení.

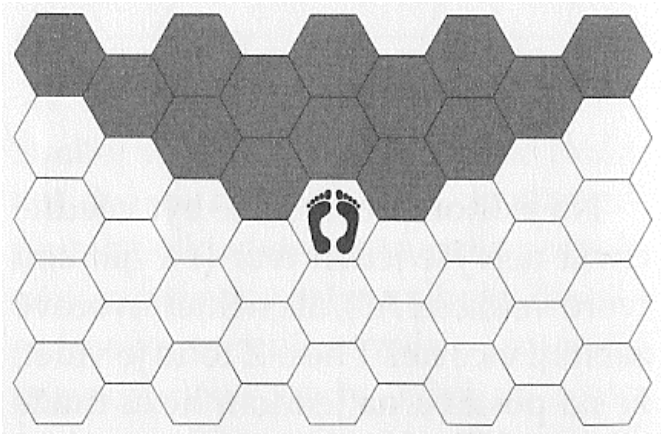
V Dračím doupěti jsou dva soubojové systémy. Oba dva využívají jistý tahový systém. Jak vyplývá z popisu v kapitole 2.1, základní systém je jednodušší a dovoluje hráčům větší volnost, zatímco rozšířený je složitější a více deterministický. Už z těchto dvou hlavních charakteristik je jasné, že pro zpracování počítačem se bude více hodit rozšířený soubojový systém. Větší časovou náročnost tohoto systému vyřeší fakt, že všechny výpočty provede počítač. Tyto výpočty v deskové hře, tedy téměř všechny musí provádět pán jeskyně a to navíc pokud možno bez kalkulačky (ta narušuje atmosféru hry) a v hlavě. Jinak by si hráči mohli snadno dopočítat přesné hodnoty jednotlivých vlastností jejich protivníka, to je samozřejmě nežádoucí. Další nevýhoda rozšířeného soubojového systému je jeho svázanost. Jak již bylo zmíněno v kapitole 2.1, zůstává v něm méně prostoru pro fantazii hráčů, tedy je méně možností a systém je více deterministický. To se ale v počítačové hře jeví spíše jako výhoda. Soubojový systém je navíc programovatelný, tedy speciální prvky si může pán jeskyně případně doplnit.

Rozšířený soubojový systém je komplexní záležitost. Většina schopností postav se týká právě souboje. A vzhledem k tomu, že pravidla Dračího doupěte mají celkově přes 500 stránek je jasné, že není možné implementovat úplně celý systém najednou. Bude vhodné využít iterativního modelu vývoje. Tedy implementovat malou část soubojového systému a následně po jejím dokončení systém dále rozšířit. Důležitou otázkou zůstává, jak velké by měly být tyto části, a co všechno by měly obsahovat. Pravidla určují 10 základních akcí. Těmi jsou

- čekání,
- útok,
- obrana,
- krok,
- otočka,
- krok a útok,
- otočka a útok,
- obrana a krok,
- jiná činnost a
- kouzlení.

Je třeba ještě doplnit jednu zásadní část a tou je iniciativa a akce.

Nyní popis jednotlivých částí. Na úvod poznámka, jak funguje souboj v rozšířeném soubojovém systému. Souboj se dělí na kola. V každém kole musí každá postava využít určitý počet akcí. Těchto akcí bývá zpravidla více a je pravděpodobné, že v jednom kole se postava dostane na tah vícekrát. Teď tedy můžeme přistoupit k popisu první části, to je mechanismus iniciativy a akcí. Ten určuje kolik akcí bude schopna daná postava použít a kdy se v daném kole dostane na řadu (kdy bude moci poprvé táhnout). S počtem akcí také souvisí, kolik akcí bude moci postava provést za jeden tah. Následuje popis samotných akcí. Čekání je akce na „zabíjení času“. Útok pracuje se 120° výsečí (viz obrázek 4.1) tedy se třemi sousedními hexy (tento počeštěný název je v pravidlech a mezi hráči zavedený) a délkou zbraně. Krok je přesun na libovolný volný sousední hex s otočkou o 120° stupňů. Otočka je otočení o libovolný úhel, kdy při všech otočkách ale musí zůstat postava čelem ke hraně hexu). Následující kombinace přesně kopírují předchozí popis s tím, že se pořád jedná o jednu akci. Kouzlení nepotřebuje dalšího vysvětlení. Mezi jiné činnosti pak patří vše od vyjednávání, přes střelecký útok po hrabání se v batohu. [3]



Obrázek 4.1: Ukázka 120° výseče na hexovém papíře [3]

S těmito znalostmi je možné nastínit návrh několika fází vývoje soubojového systému. V první fázi je potřeba zjednodušit iniciativu na hod kostkou a limit na jednu akci za kolo. Udělat akce čekání, krok a otočka. V druhé fázi zkompletujeme akce útok, krok a útok, otočka a útok. Ve třetí fázi přiřadit iniciativě počet akcí (tedy 2) a udělat fázi výběru typu akce. Výběr typu akce zatím ovlivňuje jen použitelné akce. Čtvrtou fází přidáme obranu a obranu s krokem. V tomto plánu jsou tedy vynechané pouze speciální vlastnosti povolání, jiné činnosti, zvláštní útoky a možnosti provedení více akcí na základě hodu na iniciativu. Tento systém je dle mého názoru dostatečným modelem rozšířeného soubojového systému.

4.2 Umělá inteligence

Tato podkapitola se zabývá výběrem a návrhem funkčnosti umělé inteligence. Ve kapitole 3 je popsáno několik metod, které lze využít pro implementaci umělé inteligence v tahové hře. Nyní je potřeba vybrat tu nejvhodnější. Nejprve je třeba určit kritéria výběru. Platforma DrdSim se soustředí na jednoduchost vlastní tvorby a jistý edukační charakter. Tedy využitá metoda by měla jít nějak ovlivnit a upravit tvůrcem obsahu a neměla by být příliš složitá (systém byl cílen na žáky středních škol). Tvůrce obsahu dostává k využití pouze jazyk DSL. Tedy metoda musí jít implementovat nebo ovlivnit v tomto jazyce. I přesto, že je to jazyk podobný jazyku C, nesmíme zapomínat, že neobsahuje například struktury a ukazatele. Tyto prvky jsou velmi užívané a implementace by se tak mohla velmi ztížit.

Z hlediska teorie her je soubojový systém navržen jako hra v extenzivní formě, protože se jedná o systém tahový. Jedná se o hru s nulovým součtem, ačkoliv v Dračím doupěti lze vyjednávat i uprostřed souboje, což by ukazovalo na součet nenulový. Můžeme ji zařadit mezi hry

s neúplnými informacemi, protože za ideálních podmínek má mít každý hráč dokonalé informace jen o své postavě. V Dračím doupěti je to jistě hra kooperativní. V pojetí této práce je však soubojového systém hra nekooperativní.

První podkapitola 3.1 byla metody prohledávající stavový prostor. V této části byly zmíněny dvě použitelné metody. Jsou jimi metoda Minimax a Monte Carlo tree search. Obě metody nemají úplně triviální implementace. Nevím, jestli by tvůrce obsahu nebyl spíše zaskočen než pozitivně naladěn k úpravě těchto metod. Metoda Monte Carlo je navíc jistě obtížnější na pochopení. Myšlenka, že se náhodnými tahy dojde ke statisticky lepšímu výsledku, je poněkud náročnější. Metoda by se navíc pravděpodobně vylepšila hlavně úpravou vah náhodných tahů a to si myslím, že už je celkem náročný myšlenkový pochod. Tedy z těchto dvou bych se z výše uvedených důvodů rozhodl zvolit Minimax.

Druhá podkapitola 3.2 byla věnována metodám strojového učení. Zde je poněkud problém, aby metody fungovaly jako umělá inteligence, což u metod učení bez učitele není možné. U metody učení s učitelem by to však již možné bylo. Vstupními hodnotami by byl stav souboje nebo-li výstup senzorických funkcí. Metodu rozhodovacích stromů lze použít jen obtížně. Pokud by totiž nastal stav pro který nemají řešení, metoda by pravděpodobně selhala a nic nedělala.

Lépe vypadá využití neuronových sítí, které by díky své vlastnosti generalizace fungovaly i v tomto případě. Problém ovšem nastává při uvažování, jak neuronovou síť učit. Ideální by bylo využít experta v tomto směru. Ten by určil dle svých zkušeností vhodnou množinu na které by se pak mohla síť učit. I přesto, že se každoročně koná mistrovství republiky v Dračím doupěti, nesmíme zapomínat, že systém nebude podporovat kompletně všechna pravidla. Tímto způsobem pravděpodobně experta sehnat nepůjde. Jediným závěrem je, že na aktuální systém expert neexistuje. Neuronové sítě se však využívají v kombinaci s jinými algoritmy i v posilovaném učení, které se pro tuto problematiku hodí nejlépe. Taková řešení už jsou ale velmi netriviální.

Třetí podkapitola 3.3 navrhovala využití genetických algoritmů. Tyto algoritmy jsou sice velmi účinné i v náročných situacích, ale myslím si, že soubojový systém zatím není až tak rozsáhlý, aby bylo nutno používat takto složitých metod. Metoda, která by hledala tah za pomoci těchto algoritmů by navíc jistě nemohla být v jazyce DSL. Ten totiž není příliš optimalizovaný a proces tvoření jednotlivých generací je dost výpočetně náročný.

Poslední podkapitola 3.4 byla o řešení umělé inteligence za pomoci agentů. Tento přístup stylizující jednotlivé postavy, jako reaktivní agenty využívající subsumpční architekturu se jeví jako nejjednodušší na pochopení. Ani jeho implementace není náročná a velmi jednoduše přidáním další vrstvy se rozšiřuje. Tato metoda se tedy jeví jako nejideálnější ze všech výše zmíněných.

Metodu půjde implementovat přímo v jazyce DSL. Pomocí vestavěných funkcí se potom tvůrce obsahu dostane z vlastní funkce umělé inteligence k soubojovému systému. U každé nestvůry bude uložen řetězec názvu funkce pro umělou inteligenci. Mohl by tam být sice i přímo kód, ale to by bylo nepřehledné. Tato náhrada ukazatele je mnohem elegantnější. Tedy každý typ nestvůry může mít svoji zvláštní metodu nebo ji může zdědit z nadřazené třídy. Tyto třídy jsou buď uloženy mezi globálními objekty, nebo si je může tvůrce obsahu nadefinovat sám ve složkách pro lokální dobrodružství. Funkce umělých inteligencí potom mohou být globálně mezi skripty nebo v lokálních složkách dobrodružství.

Všechny informace se do metody umělé inteligence dostanou přes vestavěné funkce. Všechny tyto funkce budou přidány do již existující třídy `interpret`. Tam jsou zatím uloženy všechny vestavěné funkce a nemá smysl schovávat ty určené pro souboj někam jinam. Je problematické určit, co vše může umělá inteligence vědět. Teoreticky lze vytáhnout ze soubojového systému všechny informace. Tedy i ty, které by postava řízená umělou inteligencí vědět neměla (například přesný počet životů nebo přesné útočné číslo). Rozhodl jsem se pro realistickou umělou inteligenci. Zde je ovšem problém, že hráč podle vybavení a svých zkušeností může určit jak silná postava může být. Toto umělá inteligence nemá. Nerad bych, aby hráč úmyslně volil vybavení tak, aby zmátl umělou inteligenci. V metodě (funkci jazyka DSL) tedy budou více či méně abstraktní vestavěné funkce pro pohyb, útok, zjištění nepřátelských pozic, atd.

Tento způsob také otvírá dveře přesnému naskriptování akcí. To již lze ale těžko považovat za metodu umělé inteligence. Za to je to způsob nejjednodušší a pro pokusy „co se stane, když...“ i nejvhodnější. Tedy možnost použití i takto jednoduchých akcí se počítá. Pro tyto účely je vhodné mít v jazyce i funkce vyšší úrovně, které dovolují volnější pokusy a zkoumání možností jazyka DSL:

Problém funkcí, které jsou na vyšší úrovni abstrakce jsou datové struktury. Ty jsou nedílnou částí procesu zobecnění. Funkce se tedy stávají jednodušší, dělají více věcí najednou, ale nezkušený uživatel může mít problém s pochopením struktury zabalené v poli. A to i přes adresování pomocí řetězců. Tato volba však bude na každém jednotlivém tvůrci obsahu k rozhodnutí.

Na závěr, jako rozšíření, jsem se rozhodl implementovat i metodu Minimax. Pokud se podíváme na metodu Minimax zblízka zjistíme, že vlastně simuluje celý soubojový systém. Přesunout celý systém do jazyka DSL by ovšem byla dosti náročný, nesystémový, špatně odladitelný a udržitelný postup, a to i v případě, že by byl zprostředkován pomocí vestavěných funkcí. Bylo tedy potřeba najít vhodné řešení této situace. Rozhodl jsem se, že metodu implementuji přímo v modulu `battle`. Ohodnocující funkce bude v jazyce DSL, aby zůstala možnost pro úpravu a tedy i seznámení se s metodou Minimax. Složitější vnitřní struktura zůstává tvůrci obsahu skryta. Metoda Minimax se tedy zavolá jako vestavěná funkce, ale všechny výpočty kromě ohodnocení proběhnou v kódu platformy, a výsledek se pak vrátí do funkce v jazyce DSL.

Metoda Minimax však předpokládá dva hráče, kteří se střídají. V rozšířeném soubojovém systému hraje ovšem několik postav za jednu i druhou stranu. V tazích se střídají podle hodu kostkou, tedy náhodně. Navíc každý z nich může využívat jinou umělou inteligenci. Například jeden může používat agenty a druhý Minimax. Tato situace je řešitelná. Podle zdroje [17] totiž Minimaxu nezáleží na pravidelném střídání protivníků. Postup popisovaný v tomto článku řeší všechny výše zmíněné problémy. Pro ilustraci uvádím následující příklad: Za sebou (pořadí může být jakékoliv) budou na tahu figurka A a B (hráče A), poté figurka Z (hráče B). Figurka A spustí Minimax. Ten do dané hloubky prochází tah figurky A, figurky B (obojí OR graf), figurky Z (AND graf), atd. Za uvedených okolností je tedy možné použít metodu Minimax i v takto upravených podmínkách. Jediným omezením zůstává, že optimalizace alfa-beta řezů je na pravidelném střídání hráčů závislá, nebude ji tedy možno použít.

5 Implementace a vyhodnocení

DrdSim je celkem rozsáhlá platforma, musel jsem nastudovat základní spojitosti celého systému. Podrobněji hlavně GUI (Graphical User Interface, grafické uživatelské rozhraní), na které jsem soubojový systém napojoval, ale i interpret a celý jazyk DSL. Nejdůležitější pro moji práci byly moduly `battle` a `interpret`. Nevyhnul jsem se však ani práci v modulu `scene` a znalostem některých funkcí z ostatních modulů.

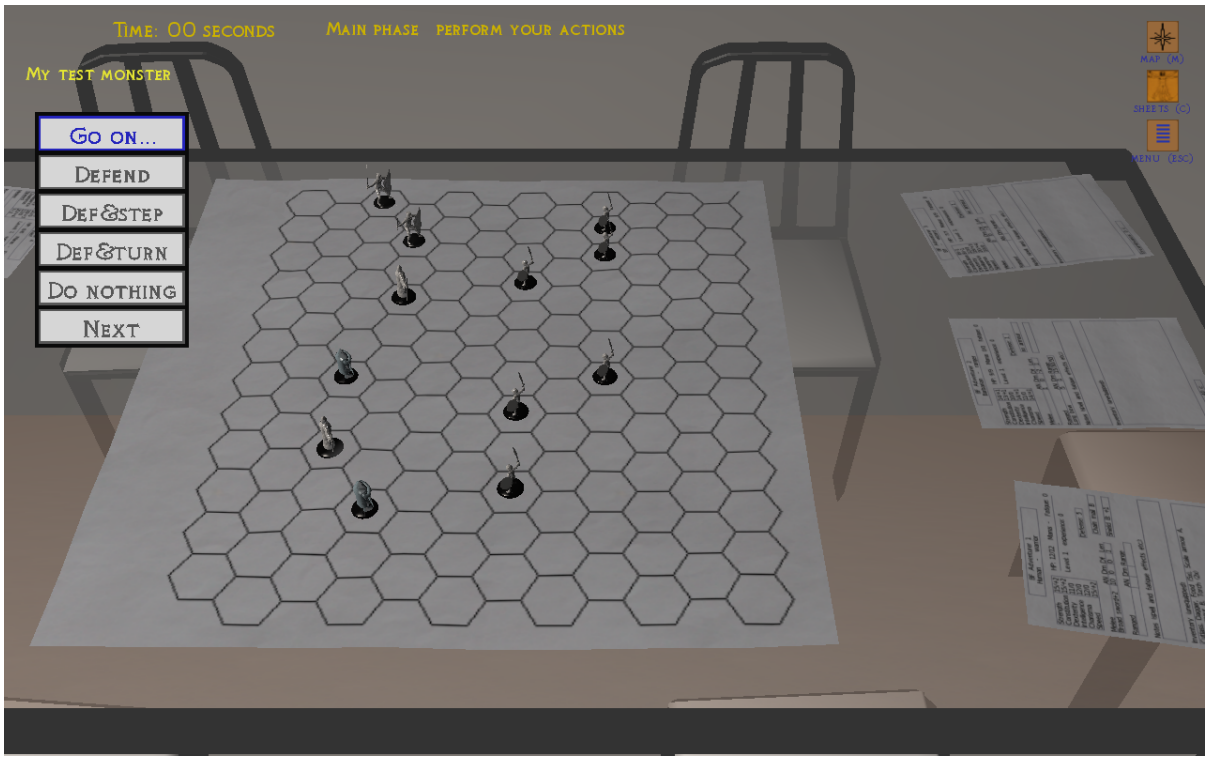
Jak jsem tedy postupoval při implementaci soubojového systému. Nabízela se možnost napojit se na třídu `battle`, kde bylo několik funkcí pro GUI souboje. Nejprve bylo potřeba vytvořit struktury pro uložení jedné postavy. Tedy vznikla třída `Pawn` a její dva potomci `EnemyPawn` a `CharacterPawn`. Toto rozdělení je nutné, protože informace o `CharacterPawn` se získávají z modulu `party`, zatímco informace o `EnemyPawn` jsou získávány ze struktur deklarativní části jazyka DSL. Tyto struktury pak s pomocí externě uložených informací mají všechny informace o dané postavě. Pro podporu skupin postav jsem využil vektoru příslušných struktur. Dále funkce pro krok, otočku a funkce pro ošetření validity pohybu. Další důležitý prvek jsou algoritmy starající se o střídání postav. Pro zjednodušení jsem využil seznam dvojic (strana a index) společně s funkcemi, které tento seznam zpracovávají a střídají podle hozené iniciativy postavy. Tímto způsobem jsem se vyhnul využití ukazatelů, což bylo jednodušší na otestování. V další fázi jsem zpracoval útok. Ten byl třeba velmi zkoordinovat s GUI. S útokem se také váže snižování životů a potencionální konec souboje. Dále jsem zapracoval strukturu `Intentions`, ta určuje, které typy akcí mohou být v daném kole použity. Společně s tím jsem převedl iniciativu na finální formát (s omezením na 2 akce za kolo). V poslední fázi zůstával poslední prvek, kterým je obrana. Po doplnění odpovídající části pro umělou inteligenci se však ukázalo, že bude potřeba pro každou akci jednotná struktura. Vznikla tedy struktura `Action`, `Attack` a `storedDefence`. Pro soubojový systém je také třeba nadefinovat soupeře. Tento soupeř se definuje v platformě DrdSim pomocí jazyka DSL. Je tedy nutné minimálně jednu třídu takového soupeře nadefinovat, aby bylo na čem celý systém testovat. Jak je vidět, testování úzce souvisí s částmi jak umělé inteligence, tak soubojového systému.

Programování umělé inteligence probíhalo hlavně v jazyce DSL. Ten jsem musel velmi důkladně prozkoumat. Napojení na platformu DrdSim probíhá pomocí vestavěných funkcí. Mnoho z nich je napojeno a využívá funkce soubojového systému, některé musely být do třídy `battle` doplněny. Vestavěné funkce zjišťují stav nebo pohybují s panáky. Pokud je tedy nazveme označením používaným u robotů,dozvíme se, že jde o senzory (zjišťují stav okolí) a efektory (tedy ty, které vyvolají danou činnost). Dostatek takových funkcí zajistil umělou inteligenci, která funguje na těchto základech: Zjisti kde jsi. Pokud můžeš, zaútoč z ideální pozice (všemi způsoby). Najdi nejbližšího nepřítele. Najdi nejvýhodnější cestu k němu. Přesuň se do co nejvíce ideální pozice. Hotová funkce je téměř ideální pro rozzuřené nebo zvířecí postavy.

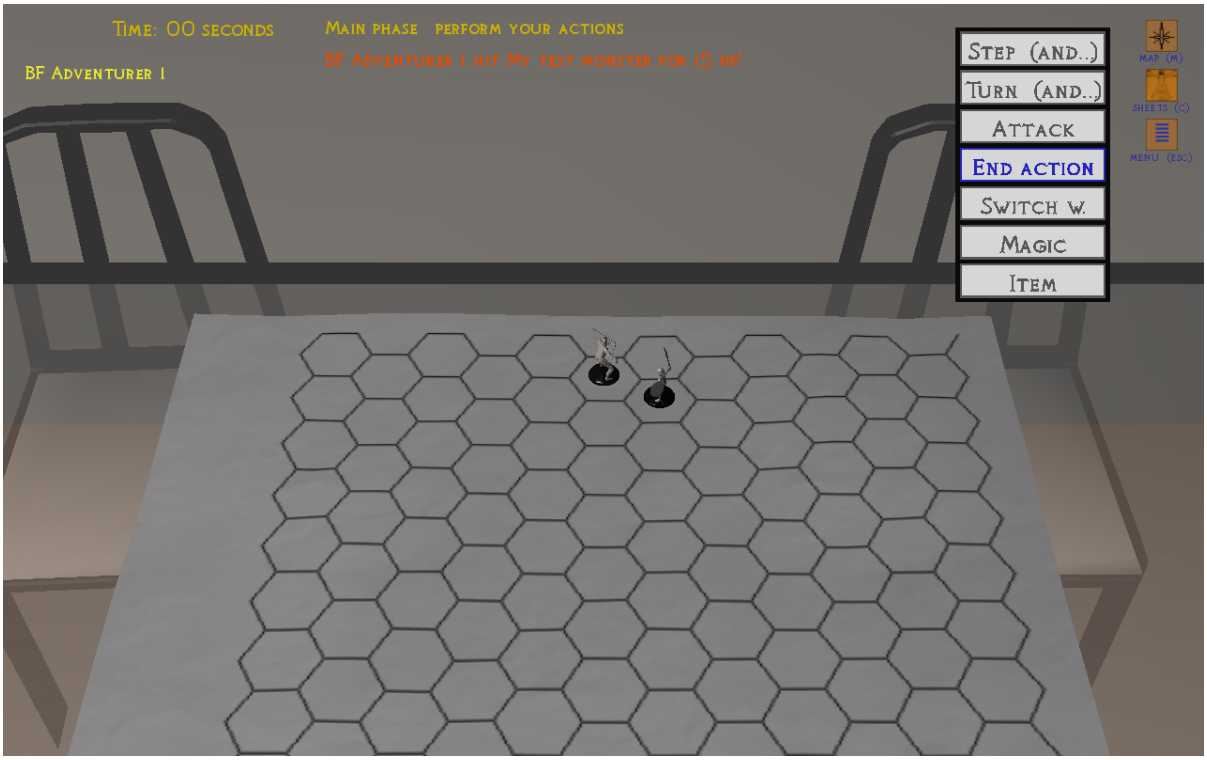
V této fázi jsem pokračoval programováním rozšíření Minimax, abych mohl na závěr oba přístupy porovnat. Vestavěná funkce Minimax je tedy výjimka, která není ani senzorická ani efektorická. Tato funkce má vrátit ideální tah pro aktuální stav hry. Pro tento účel jsem nadefinoval strukturu `State`, která uchovává stav hry. Po vytvoření struktury všech funkcí nutných k implementaci této metody jsem zjistil, že stávající implementace třídy `battle` je příliš vázaná na aktuální stav hry. Finální napojení funkcí zpracovávajících Minimax by tak znamenalo nesystémovou změnu všech funkcí soubojového systému nebo jeho kompletní přepsání. Rozšíření se mi tedy zprovoznit nepovedlo. Pokud by se s tímto rozšířením počítalo již v počátku, bylo by možné funkce psát více obecně. V této fázi je již na tak rozsáhlou změnu pozdě. Funkce tak zůstávají v kódu, ale do restrukturalizace třídy není možné je zprovoznit.

V závěru jsem implementoval ještě několik funkcí v jazyce DSL pro zjednodušení vytváření vlastní umělé inteligence. Poslední částí je programovatelnost soubojů. Ta je tvořena polem `meta` atributu `handler` bázové třídy `Monster`. Pokud se do tohoto pole s indexem `near` dá číslo jedna bude souboj začínat blízko u sebe. Popis všech vestavěných funkcí je v příloze C.

Po spuštění vše funguje tak, jak se očekává. Na políčku, kde jsou nepřátelé se spustí souboj (v cvičně uložené hře *fighttest* stačí jít jedno políčko na jihovýchod). Hráč si vybere typ akce, hra provede hod na iniciativu a souboj začíná. V soubojovém systému správně fungují všechny 4 fáze tak, jak byly navrženy v kapitole 4.1. Systém také nedovolí použít tah, který odporuje pravidlům. V použitém příkladu je celý souboj otázka tří nebo čtyř kol. Po ukončení se objeví zpráva konec souboje. Přechod zpět na mapu příběhu zatím GUI nepodporuje. Podobným problémem je i umírání postav. Pokud klesnou postavě životy na nulu, objeví se místo velikosti zásahu informace o tom, že postava umřela. Postava se nadále neúčastní souboje, ale graficky je pořád přítomna. Nelze na ni ale útočit ani šlapat. Algoritmus umělé inteligence lze jednoduše změnit, skupinku nepřátelských postav libovolně nakombinovat. Funkce pro blízký začátek souboje také funguje. I přes využití nepříliš optimalizovaného jazyka DSL je základní umělá inteligence rychlá, reaguje s neznatelným zpožděním. Na závěr uvedu několik obrázku z finálního produktu. Rád bych zdůraznil, že grafické uživatelské rozhraní nebylo předmětem mojí práce (respektive bylo jen velmi okrajově).



Obrázek 5.1: Ukázka ze soubojového systému v DrdSim.



Obrázek 5.2: Ukázka ze soubojového systému v DrdSimu.

6 Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat soubojový systém a umělou inteligenci pro něj. Tuto činnost přizpůsobit hlavnímu pravidlu platformy DrdSim. Tím je jednoduchost pro tvůrce obsahu.

Nastudoval jsem všechny potřebné materiály, co se týče Dračího doupěte a výše zmíněné platformy DrdSim (viz kapitola 2). V kapitole 3 jsem pak nastudoval několik metod umělé inteligence. Poté jsem z těchto znalostí navrhl ve 4 kapitole soubojový systém i použitelné metody umělé inteligence. Následně jsem podle předlohy rozšířeného soubojového systému Dračího doupěte implementoval soubojový systém. Tento systém je podmnožinou výše zmíněného. Úspěšně jsem všechny splnil v kapitole 4.1 navrhované fáze tohoto soubojového systému a celý tento systém napojil na platformu DrdSim.

V tomto odstavci se budu věnovat umělé inteligenci pro tento systém. Pro tvůrce herního obsahu jsem implementoval v jazyce DSL jednoduchou a snadno pochopitelnou i editovatelnou umělou inteligenci na bázi reaktivních agentů. Vypracoval jsem také jako rozšíření umělou inteligenci pomocí metody Minimax. V průběhu práce se však ukázalo, že ji nebude možné dokončit kvůli špatně navržené návaznosti na již hotový systém. Nedílnou součástí byla také implementace mnoha vestavěných funkcí. Tím jsem splnil i vše (vyjma Minimaxu) z kapitoly 5.

Dále budu diskutovat o možnostech rozšíření. Těch je, co se týče soubojového systému, případně celé platformy, velké množství. Patří mezi ně schopnosti povolání a jejich zakomponování do soubojového systému, včetně kouzel a alchymie. Pro větší názornost bude potřeba udělat animace v GUI a přidat další modely postav. Z pohledu umělé inteligence by bylo vhodné provést následující rozšíření. Především dodělat těch několik úprav nutných pro fungování metody Minimax. Pro realističtější práci metod by byly užitečné funkce pro klasifikaci a odhad soupeře podle jeho vybavení (zranění, atd.). Zajímavé by také bylo přidat do souboje překážky či speciální vlastnosti různých povrchů.

Zadání práce jsem tedy splnil. Mimo to jsem získal mnoho cenných zkušeností při spolupráci na takto rozsáhlém projektu.

Seznam použité literatury

1: Hahn, Joel A., Lawrence Mead and Ian Malcomson: Dungeons & Dragons FAQ, Wizard of the Coast [online]. 2003 [cit.2.5.2012]. Dostupný z WWW: <http://www.wizards.com/dnd/DnDArchives_FAQ.asp>.

2: Markus: Dračí doupě po 20 letech: recenze legendy, RPGForum.cz [online]. Čt, 29.10.2009 - 22:12 [cit.5.5.2012]. Dostupný z WWW: <http://rpgforum.cz/clanky/draci-doupe-po-20-letech-recenze-legendy>.

3: Klíma, Martin, Martin Kučera a kol.: Dračí doupě: Pravidla pro začátečníky, verze 1.6d. Praha: ALTAR, 2003. ISBN: 80-85979-34-9.

4: Změny v DrD Pravidlech pro začátečníky verze 1.6 oproti verzi 1.5, ALTAR [online]. 30. 08. 2004 21:09 [cit.4.5.2012]. Dostupný z WWW: <http://www.altar.cz/drd/ppz.html>.

5: Hranáč, Jan: Jazyky a překladače v počítačových hrách. Brno, 2010. diplomová práce. FIT VUT v Brně.

6: Mařík, Vladimír, Olga Štěpánková, Jiří Lažanský a kol.: Umělá inteligence (1). Praha: Academia, 1993. ISBN: 80-200-0496-3.

7: Myerson, Roger B.: Game Theory: analysis of conflict. : Harvard University Press, 1997. ISBN: 0-674-34116-3.

8: Depth-first search, Wikipedie [online]. 6 May 2012 at 17:44 [cit.3.5.2012]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Depth-first_search>.

9: Alpha-beta pruning, Wikipedie [online]. 10 April 2012 at 08:16 [cit.3.5.2012]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Alpha-beta_pruning>.

10: Serédi, Silvester: Počítačová hra s prvky umělé inteligence. Brno, 2010. bakalářská práce. FIT VUT v Brně.

11: Chaslot, Guillaume, Sander Bakkes, István Szita a Pieter Spronck. Monte-Carlo Tree Search: A New Framework for Game AI. In: Mateas, Michael a Chris Darken. Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference. Menlo Park, CA: AAAI Press, 2008, s. 216-217.

12: Mařík, Vladimír, Olga Štěpánková, Jiří Lažanský a kol.: Umělá inteligence (4). Praha: Academia, 2003. ISBN: 80-200-1044-0.

13: Artificial neural network, Wikipedie [online]. 5 May 2012 at 07:02 [cit.5.5.2012]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Artificial_neural_network>.

14: Mařík, Vladimír, Olga Štěpánková, Jiří Lažanský a kol.: Umělá inteligence (3). Praha: Academia, 2001. ISBN: 80-200-0472-6.
15: Halamíček, Jan: Real-time počítačová hra s prvky UI. Brno, 2009. diplomová práce. FIT VUT v Brně.
16: Šperka, Svatopluk: Simulace a návrh inteligentních agentů. Brno, 2009. diplomová práce. FIT VUT v Brně.
17: Melkó, Ervin a Benedek Nagy. Optimal strategy in games with chance nodes. Acta Cybernetica. 2007, roč. 18, č. 2, s. 171-192.

Příloha A Obsah CD

- Soubor "README.TXT"
  - Zde je komentář a návod k obsahu CD. Postup k testování. Je nutné si jej prvně přečíst.
- Adresář "report"
  - Obsahuje elektronickou verzi tohoto textu.
- Adresář "drdsim"
  - Obsahuje hru drdsim (zdrojové kódy i data).
  - Podadresář "html“
    - Programová dokumentace.

Přiložený Makefile funguje na Linuxu. Je třeba mít nainstalovány patřičné balíčky; na serveru Merlin patřičné balíčky v patřičné verzi nejsou, viz dále). Projekt funguje také ve Windows a to za použití kombinace *mingw+msys* funguje také. Hra vyžaduje počítač s grafickou kartou podporující shadery. Ačkoliv většina počítačů na FIT VUT v Brně toto splňuje, je možné narazit na počítač, který nevyhovuje. Je možné hru zkompilovat tak, aby neobsahovala shadery. Hru lze tedy zkompilovat i na serveru Merlin). Bližší informace možno nalézt v readme. [5]

Příloha B Podrobný popis jazyka DSL

V této příloze bude podrobně popsán jazyk DSL. Celá tato příloha se opírá o [5]. Tam také najdete plné znění a popis v souvislostech s platformou DrdSim. Jazyk DSL se skládá ze dvou částí. V podkapitolách tedy budou popsány tyto dvě části a navíc i příklady, jak s nimi pracovat.

B.1 DSL - deklarativní část

Tato část jazyka slouží k definici tříd objektů umístitelných do hry. Nejedná se o plně objektově orientovaný jazyk. Jedná se pouze o definici atributů tříd (tedy jakási obdoba struktur, ale s dědičností). Je sice pravda, že tyto objekty mohou mít obslužné funkce pro události, ale tyto funkce nemohou být volány uživatelem jako v objektově orientovaném přístupu. Třída v jazyce DSL je tedy záznam atributů, konkrétně řetězců a celých čísel. Krom toho je však do atributu možno přiřadit i procedurální kód z druhé části jazyka. Jedná se o deklarativní jazyk. Zde je pro shrnutí syntaxe definice třídy v jazyce DSL:

<id třídy> [: <id mateřské třídy>]
"{"
{<id atributu> = <hodnota atributu> ;}
"}"

Použité neterminály nejsou dále rozvinuty, protože nepotřebují další komentář ani definici. Je třeba pouze podotknout, že hodnotou atributu může být celé číslo, řetězec nebo blok kódu. Uvedení mateřské třídy není povinné. Je ale důrazně doporučeno všechny vytvářené třídy vydědit z nějaké již existující, přinejmenším alespoň z jedné z následujících tříd: Item, MapObject, nebo Spell. Uvedené tři třídy jsou jediné globální třídy, které nemají mateřskou třídu a všechny ostatní jsou odvozeny z jedné z nich. Dále je třeba se zmínit o jedné ze specialit DSL. Jedná se o lokalizované řetězce. Tyto řetězce jsou definovány ve speciálním textovém souboru v hlavním adresáři hry nebo v adresáři kampaně. Na lokalizované řetězce se dá v DSL odkazovat napsáním dolaru a názvu řetězce. Tento odkaz je zcela ekvivalentní řetězcovému literálu, na který odkaz odkazuje. Při lokalizaci pak stačí vyměnit jen lokalizační soubor. Tato lokalizace probíhá za běhu aplikace, při kompilaci DSL souborů. Odkazování na lokalizované texty je možno použít v obou částech DSL.

B.2 DSL - procedurální část

Tato část se velice podobá definici funkcí v jazyce C. Definice funkce je také hlavním a jediným stavebním prvkem této části DSL. Definice globálních proměnných, deklarace externích proměnných a funkcí - nic z toho není třeba díky pozdní vazbě. Struktury se také nepoužívají, místo toho jsou k dispozici pole podobné jako v jazyce PHP. Oproti jazyku C zmizely definice typů, vše se ověřuje při automatické interferenci až za běhu. Mnohé složitější konstrukce z jazyka C byly v DSL odstraněny. Tím se sice ztratily některé možnosti efektivního psaní kódu, ale nová očištěná syntaxe nutí psát programátora pěkně a přehledně. Zde je popis syntaxe definice funkce:

<id funkce> "(" [<id atributu> { , <id atributu> }] ")"
<blok kódu>
<blok kódu> ::= "{"
{ <příkaz> }
"}"
<příkaz> ::= <přiřazení> | <volání funkce> | <return příkaz> |
<if konstrukce> | <while cyklus>
<přiřazení> ::= <l-hodnota> = <výraz> ;
<l-hodnota> ::= [global] <id proměnné> { "[" <výraz> "]" }
<if konstrukce> ::= if "(" <výraz> ")" <blok kódu>
{ elif "(" <výraz> ")" <blok kódu> }
[else <blok kódu>]
<while cyklus> ::= while "(" <výraz> ")" <blok kódu>

Opět platí, že věci dále nerozváděné jsou řešeny tak, jak by se předpokládalo. K výrazům je třeba dodat, že podporovány jsou všechny základní matematické binární a logické operace, z unárních operací pouze znaménková a *boolean* negace. Protože se jedná o vysokoúrovňový jazyk, nejsou k dispozici žádné operátory pro bitové operace nebo pro práci s ukazateli. Můžeme si povšimnout, že je přítomen pouze cyklus *while*. V *if* konstrukci nebo *while* cyklu není možné použít jednoduchý příkaz, pouze blok kódu. V jazyce neexistuje *switch* konstrukce. Novinkou je klíčové slovo *global*. Protože proměnné (globální i lokální) jsou definovány přiřazením, je třeba rozlišit mezi vytvořením lokální proměnné a globální. Globální proměnné mají v jazyce zvláštní pozici, protože nepředstavují jen místo v paměti, ale i informaci, která je uložena na disk jako součást rozehrané hry. Klíčové slovo *global* se používá pouze při vytvoření globální proměnné (nebo při jejím přepisu). Při použití se již píše jen název této proměnné. Pokud není nalezena lokální proměnná takového jména, automaticky se bere jako globální (určováno v době kompilace). V souvislosti s poli je třeba zmínit, jak se s nimi pracuje. Pole se vytvářejí přiřazením speciálního výrazu do proměnné, tedy do prázdných hranatých závorek. V případě globálního pole je ještě třeba přidat patřičné klíčové slovo. Příklad:

```
pole = [];
global globalni_pole = [];
```

Následně je možno pracovat s polem stejným způsobem jako v PHP. Příklad:

```
lide[5] = [];
lide[5]["jmeno"] = "Jan";
lide[5]["prijmeni"] = "Novak";
```

Z uvedeného příkladu je vidět, že stejně jako v PHP jsou i zde struktury nahrazeny poli. Jak již bylo řečeno v předchozí části, i v procedurální části DSL je možno použít odkaz na lokalizovaný řetězec. Takový odkaz bude při kompilaci nahrazen řetězcovým literálem, na který odkazuje.

B.3 Programování v DSL

Zde bude jazyk DSL vysvětlen na příkladech.

Definice třídy: Definice třídy je uvedena hlavičkou, která se skládá z názvu třídy a případně z dvojtečky a názvu mateřské třídy. Tělo třídy je uzavřeno mezi složenými závorkami a obsahuje seznam atributů a jejich hodnot (název, rovnítko, hodnota). Hodnotou může být řetězec, odkaz do *stringtable* nebo číslo. Navíc je možno jako atribut uvést i další složené závorky a procedurální kód z druhé části DSL. Příklad:

```
/// Base class for triggers.
Trigger: MapObject
{
handler = {StdPrintLn("Base trigger triggerred.");}
}
/** Base class for NPCs - encounter won't implicitly
* start a fight but it can still go that way. */
NPC: Trigger
{
npc = 1;
handler = {StdPrintLn($GOODDAY);}
}
```

Definice funkce: Definice funkce je uvozena názvem funkce, kulatými závorkami a v nich případným seznamem názvů parametrů. Poté následují složené závorky a v nich tělo funkce. Příklad:

```
MojeFunkce(parametr1,parametr2)
{
// tělo funkce
}
```

Tělo funkce se skládá z příkazů a to složených, nebo jednoduchých. Jednoduché příkazy jsou ukončeny středníkem.

Jednoduchý příkaz může být přiřazení, volání funkce, nebo návratového příkazu. Přiřazení je provedeno napsáním názvu proměnné, rovnítka a výrazu určujícím novou hodnotu. Před název proměnné je možno připsat klíčové slovo *global*. Volání funkce je provedeno napsáním názvu funkce, kulatých závorek a v nich seznamu výrazů určujících argumenty volání. Návrat z funkce je proveden klíčovým slovem *return* a případně hodnotou, která se má vrátit. Výraz je složený z názvů funkcí (tentokrát bez *global*), volání funkcí, čísel, řetězců a aritmetických operátorů. Příklad:

```
global prom1 = 2*3;
prom2 = "ahoj";
MojeFunkce(prom1+1,prom2+"svete");
```

Ze složených příkazů je možno použít *if* a *while* způsobem uvedeným v příkladu:

```
n = ZiskejPocet();
if (n==7) {
// ...
}
elif (n>3) {
// ...
}
else {
// ...
}
i = 0;
while (i<n) {
// ...
i = i+1;
}
```

Práce s poli je maximálně jednoduchá. Nejprve je nutno pole vytvořit. To se provede stejným způsobem jako přiřazení, ale napravo od rovnítka je nutno napsat hranaté závorky. Poté je možno přiřazovat dovnitř pole. To je provedeno napsáním názvu pole, hranatých závorek s indexem a poté rovnítkem a výrazem. Index pole výraz, který má hodnotu buď celého čísla nebo řetězce. Uvnitř pole

je možno vytvářet další pole a zapisovat do nich přidáním dalších hranatých závorek. Stejným způsobem je možno z pole číst. Příklad:

```
pole = [];
pole[2] = [];
pole[2]["jmeno"] = "Tonda";
pole[2]["prijmeni"] = "Ucho";
StdPrintLn(pole[2]["jmeno"]+" "+pole[2]["prijmeni"]);
```

Příloha C Seznam vestavěných funkcí

Systém má v sobě vestavěno omezené množství funkcí, které umožňují ovládat hru. Mimo to existuje několik funkcí souvisejících se soubojovým systémem a umělou inteligencí. Krom toho je možno v samotném jazyce DSL nadefinovat pomocné funkce a uložit je v adresáři `scripts` v hlavním adresáři hry. Podrobnější popis může nabídnout programová dokumentace v přiloženém CD. Zde je seznam všech vestavěných funkcí:

Název funkce a typy parametrů	**Popis**
int Rand6()	Vrátí číslo od jedné do třinácti (po 6 hází znova).
int Rand10()	Vrátí číslo od jedné do deseti.
void StdPrintLn(string)	V Linuxu vypíše na konzoli ladící hlášku. Ve Windows uloží do log souboru.
string Nmr2Str(int)	Převede číslo na řetězec.
void PrintNmr(int)	Na konzoli vypíše číslo.
PlayMusic	Přehraje hudbu z daného souboru.
StopMusic	Zastaví hudbu.
PlaySound	Přehraje zvuk.
StopSounds	Vymaže zásobník zvuků.
bool IsGlobalDefined	Zjistí, zda existuje globální symbol daného jména.
UndefineGlobal	Zruší definici globálního symbolu daného jména.
int GetType(int/float/string/array)	Zjistí, jakého typu je daná proměnná.
ShowDialog	Zobrazí okno dialogu. Očekává jeho text, pole odpovědí (text + hodnota) a název řídící funkce.
ShowImageDialog	Zobrazí okno dialogu. První parametr je název obrázku. Zbytek jako u ShowDialog.
HideDialog	Schová dialog (ukáže standardní GUI).
StartCombat(array_meta)	Začne souboj s aktuálním spouštěčem.
Funkce spojené se soubojovým systémem	
MovePawn(dir)	Naplní první akci krokem v daném směru.
DblMovePawn(dir)	Naplní první akci dvojitým krokem v dané směru.
TurnPawn(dir)	Naplní první akci otočením k danému směru.

AttackPawn(ind)	Naplní první akci útokem na daného soupeře.
CreateAction()	Vytvoří akci.
ChangeAction(array_action)	Naplní akci.
array_x_y WhereAmI()	Vrátí, kde je aktuální postava.
array_x_y WhereIs(ind, who)	Vrátí, kde je daná postava.
bool CanMove(dir)	Zjistí, jestli se může postava pohnout daným směrem.
bool CanMoveFrom(array_x_y, dir)	Zjistí, jestli se může postava pohnout z daného místa daným směrem.
bool CanAttack(ind)	Zjistí, jestli se může postava na danou zaútočit.
bool CanAttackFrom(ind, array_x_y, tdir)	Zjistí, jestli se může postava z dané pozice zaútočí na danou postavu.
int CharCount()	Vrátí, kolik je soupeřů.
int Distance(array_x1_y2, array_x2_y2)	Vrátí vzdálenost mezi dvěma body.
array_x_y FakeMove(array_x_y, dir)	Vrátí, kde bude postava po daném kroku z dané pozice.
bool CanStepAttack(ind, dir)	Zjistí, jestli může použít krok a útok.
int GetMyDir()	Vrátí, kam je postava natočena.
int GetDirOf(ind, who)	Vrátí, kam je natočena daná postava.
bool IsLive(ind, who)	Zjistí, jestli postava žije.
Funkce, které obchází mechanismus počítání akcí	
CheatMovePawn(dir)	Posune postavu do daného směru.
CheatTurnPawn(dir)	Natočí postavu do daného směru.