Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Umělá inteligence v počítačových hrách

Plzeň 2008

Vondráček Tomáš

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne

Abstract

Artificial inteligence in computer games
Objective of this work is to design scenario of computer game, make a proposal of AI library and implement it into the game engine. For this, there is a need of studing varios methods. Game scenario has to include, among game play desgin, roles of several non-playable character managed by the game artificial inteligence, their complete design and properties. Also has to contains describe of all posible actions, reactions and states. Game AI library has to fit into the game engine and must be designed to cover all tasks of proposed game.

Obsah

1 Úvod **1**

2 Teoretická část **2**
2.1 Návrh scénáře počítačové hry . . . 2
2.1.1 Obecný popis hry . . . 2
2.1.2 Platforma . . . 2
2.1.3 Příběh . . . 2
2.1.4 Situace a prostředí . . . 3
2.1.5 Průběh hry . . . 3
2.1.6 Popis postav . . . 3
2.1.7 Vlastnosti postav . . . 4
2.1.8 Design kouzel . . . 5
2.2 Studované metody umělé inteligence . . . 5
2.2.1 Konečný automat (Finite State Machine) . . . 5
2.2.2 Dynamické programování (plánování) . . . 6
2.2.3 Hledání cesty (Path finding) . . . 8
2.2.4 Skriptování . . . 9

3 Realizační část **12**
3.1 Návrh knihovny umělé inteligence . . . 12
3.2 Propojení s herním enginem . . . 13
3.3 Implementace vybraných metod . . . 17
3.3.1 Konečný stavový automat . . . 17
3.3.2 Použití skriptů . . . 19
3.3.3 A* algoritmus . . . 22
3.3.4 Výběr další akce (Action selection) . . . 23
3.4 Řešení viditelnosti . . . 24
3.5 Výsledky testů a pozorování . . . 25

4 Závěr **30**

1 Úvod

Důvodem vzniku této práce je potřeba herního scénáře a umělé inteligence v počítačové hře, kterou vyvíjíme s dalšími kolegy. Náplní práce je návrh herního scénáře s důrazem na popis protivníků, jejich vlastností a možných akcí, u kterých je také nutné nadefinovat vlastnosti, váhy pro rozhodování, reakce a důvody. To vše s ohledem na vyváženost a hratelnost. Dále je obsahem práce popis metod pro chování a rozhodování počítačem ovládaných bytostí, které by byly vhodné pro navržený scénář. V realizační části pak následuje popis implementace knihovny umělé inteligence včetně popisu implementace vybraných metod a začlenění do herního enginu. Závěr páce tvoří výsledky, tedy příklady chování, posloupnosti akcí v určitých situacích, do kterých se ve hře postavy mohou dostat.
Výsledkem této práce by měla být funkční knihovna umělé inteligence, která pokryje veškeré potřeby, které vyplývají z navrženého scénáře. Zároveň by knihovna měla být plně zakomponována do herního enginu.

2 Teoretická část

Zde bude popsán navržený scénář počítačové hry a jednotlivé studované metody umělé inteligence, které připadají v úvahu pro použití v knihovně umělé inteligence. Umělá inteligence v počítačových hrách má skrze iluzi inteligentního chování zlepšit hratelnost hry.

2.1 Návrh scénáře počítačové hry

Pro návrh hry je směrodatná hratelnost a zajímavost pro hráče. Zároveň je důležité, aby veškeré parametry byly vyladěny a nedocházelo pak k nevyváženosti a hra nebyla například příliš snadná, tedy nudná, nebo naopak příliš těžká a frustrující. Nikoho nebaví soupeřit pořád dokola s jedním protivníkem, kterého stále ne a ne porazit.

2.1.1 Obecný popis hry

Hra je 3D akční RPG, odehrávající se ve fantasy světě v době kdy světu vládla magie. Hlavní hrdina, kouzelník, má za úkol objasnit záhadu zmizení náboženského vůdce vesnice. Musí najít tajemnou knížku, kterou měl duchovní u sebe, a musí tak učinit dřív, než se k tomu dostanou nebezpečné nepřátelské klany. Hráč bude vnímat svět z pohledu první osoby. Celá hra je pouze venkovní.

2.1.2 Platforma

Hra je určena pro osobní počítače s operačním systémem Microsoft Windows XP a vyšší. Ke svému běhu vyžaduje nainstalovaný .NetFramework 2.0 a DirectX 9.0.c.

2.1.3 Příběh

V zemi, kde se hra odehrává, není žádná vláda ani král, jednotlivé vesnice a města řídí místní duchovní. Nejrozšířenější a nejvíce hlásané náboženství je Wicca a tak všichni žijí ve zdánlivém míru. Ovšem ne všichni žijí ve vesnicích zapojeni do společnosti a vytvářejí různé nezávislé tlupy a klany sídlící v lesích, které žijí podle svých pravidel nesouhlasíce se všeobecným náboženstvím. Ve vesnici, kde hra začíná, má místní duchovní mimo jiné velmi vzácnou knihu, ve které jsou popsány počátky Wiccy a tajné rituály, které nikdo nezná. Duchovní se pravidelně vydává každý týden sám na meditace do hlubokých lesů. S sebou si vždy bere onu knihu. Jednoho dne se však nevrátí a ve vesnici je shon. Zde začíná hra. V té době se připlete do vesnice hrdina, který se nakonec vydá duchovního hledat. Spolu s ním ho hledají i neznámé klany, které se snaží získat knihu, která by jim mohla pomoci oslabit pozici hlavního náboženství a dodat jim moc, když by poznali podstatu Wiccy. Pokud se hrdinovi nepodaří úkol splnit, hra končí, pokud však zjistí, kde má hledat a najde vůdce dříve než všichni ostatní. Hrdina zjistí, že vůdce přešel na temnou stranu a přichytí ho při rituálu s dalšími lidmi. Bohužel je hrdina odhalen a všichni, kdož rituál prováděli, zmizí neznámo kam. Hrdina se vydává na cestu, aby vyzjistil co se děje. Zde hra končí.
Nutno podotknout, že toto má být pouze malé demo z celé verze, úvod.

2.1.4 Situace a prostředí

Design herního prostředí je dlouhodobější záležitost. I když už je herní engine hotový, je třeba vytvořit modely a představovaný svět poskládat, následně vyladit. Idea vypadá následovně.
Hráč se nachází ve fantasy světě. Krajina je obdobná té skutečné, je pokryta lesy, loukami, na východě jsou skály a hory. Nacházejí se zde některá zvířata, ale i různá magická stvoření, jejich původ hráč nezná. Po většinu doby je zde však silně zamračeno a podivná tma. Postavy jsou vybaveny zapálenými pochodněmi a svítí si tak na cestu. Na západě mapy se nachází vesnice obydlená lidmi, Ebertan, která čítá přibližně 50 lidí. Ve vesnici jsou postaveny budovy ze dřeva a kamení. Sídlí zde různá povolání, jako jsou obchodníci, kováři apod. Hlavou vesnice je její duchovní, který sídlí ve velkém klášteře. V lesích se ukrývají různé tlupy.

2.1.5 Průběh hry

Hra začíná v krajině, kousek od vesnice. Když hrdina přijde do vesnice, dozví se od vesničanů, že se jejich duchovní nevrátil z jeho pravidelného výletu do lesů. Chvíli na to v hostinci místní vyhlásí, že hledají někoho, kdo by jim pomohl jejich vůdce najít. Když se v tom dozví hrdinův původ, požádají jej o pomoc. Ten souhlasí. V dalším rozhovoru s vesničany se dozví, že v krajině jsou rozmístěny 4 obelisky, ve kterých má být zašifrována poloha posvátného místa, kde se smí provádět rituály popsané v knize. Ve vesnici jsou zároveň příslušníci klanů sídlících v lesích, aby vyzvěděli, co se stalo. Kniha by jim mohla dodat moc a oslabit pozici hlavního náboženství, proto po ní touží. Hrdina se vydá směrem k obeliskům, aby zjistil, kde hledat. U každého z nich je jeho strážce, kterého musí porazit, aby se dostal k tabulce na obelisku, kde je vytesán kus mapy. Když takto obejde všechny obelisky, zjistí polohu duchovního vůdce, která je za úzkou soutěskou a z mapy po vyluštění hádanky vyčte kód, který je potřeba pro přístup k duchovnímu vůdci. Vyslanci klanů mají stále ten samý úkol a stejné podmínky pro jeho splnění a tak nastává konflikt, který se dá řešit silou, ale i chytrostí. Hráč nemusí nutně všechny porazit, stačí, aby jako první došel na místo, kde se nachází duchovní. To je za těsnou soutěskou. Když překoná nástrahy cesty, dorazí jako první na určené místo, najde duchovního při rituály, ten se lekne a zmizí, hrdina se vydává na cestu za objasněním. . .

2.1.6 Popis postav

Hlavní postava je Nathaniel, mladý, dospělý, lidský muž, učeň slavného kouzelníka Elemaara, který je všude uznáván pro své schopnosti a moudrost. Již dokončil své učení a vydal se na pouť světem. Zvládá několik základních kouzel na útok a na léčení. Vesnice, kde děj začíná, se jmenuje Ebertan, její duchovní vůdce je Matheel. Ten je od malička veden k pravidlům náboženství a silně jej prosazuje. Ve hře má jen dějovou úlohu a nijak aktivní nebude. Dále zde budou vystupovat vesničané, kteří hráče přiblíží ději a kouzelníci lesních klanů, kde každý klan vyšle jednoho, aby se pokusil získat knihu. S těmi bude hráč konfrontován v průběhu hry, jelikož mají všichni stejný cíl. Tudíž, všichni jdou proti sobě.

2.1.7 Vlastnosti postav

Hráči (hrdina i proti němu stojící NPC) budou mít definované atributy, které jsou víceméně běžné v žánru RPG her:

- hit body - vyjadřují zdraví postavy, ubývají se zásahem nepřítele a doplňují se buď kouzlem nebo velmi pozvolna samy od sebe. Typické hodnoty se pohybují mezi 1000 až 3000.
- mana - pro kouzlení většiny kouzel je třeba mana, čím více many, tím více kouzel může postava kouzlit. S každým vykouzleným kouzlem se odečte příslušné množství many. Může se doplnit speciálním kouzlem nebo podobně jako hit body se sama od sebe pozvolně doplňuje. Typické hodnoty jsou v rozmezí 300 až 1000.
- energie - Každá postava má do začátku 100 bodů energie a ty ubývají s každou akcí, kterou NPC vykoná. Může se doplnit, pokud je NPC v odpočinkovém stavu (viz kapitola 3.3.1).
- typ - značí zařazení postavy a určuje, jakým způsobem se bude NPC chovat.
- level - úroveň postavy, určuje, jak je dobrá. Ovlivňuje zranění, které může postava způsobit. Je v řádu jednotek.
- priority - platí pouze pro NPC. Číselnými hodnotami určuje některé vlastnosti postavy. Může se uplatnit při rozhodování. Obsahuje priority pro přežití, zabití nepřátel a jiné.

Typ NPC může být:

- Villager - vesničan, který nebojuje, jen podá případně informace hráči a dokresluje děj.
- Enemy - nepřítel, který soupeří se všemi ostatními. Jsou to hlavní postavy, které jsou ovládány AI
- Boss - hlavní nepřítel, jeho nalezení je částečně cíl hry.
- Guard - stráž, uvidí-li nepřítele, tak jej napadnou.
- Beast - zvíře či jiná příšera, bez nějakých úkolů.

Atributy postav by se dále mohly rozšířit například o sílu, agilitu, inteligenci a tím různoroději ovlivňovat sílu kouzel, odolání útoku a jiné. Souboje spočívají v kouzlení útočných a uzdravujících kouzel. Poškození, které kouzlo, způsobí je ovlivněno atributem útok útočníka a obrana u zasaženého.

2.1.8 Design kouzel

- Útočné kouzlo, *fireball*. Má velkou účinnost, stojí více many a trvá více času ho vykouzlit.
- Lehké útočné kouzlo, *fire drop*. Má malou účinnost, ale je velice levné na manu, energii i čas.
- Uzdravovací kouzlo, *heal*. Vyléčí daný počet hit bodů vlastní postavě.
- Kouzlo doplnění many, *Moon prayer*. Doplnění určitý počet many kouzelníkovi, nepotřebuje manu, ale hodně energie

Každé kouzlo má nadefinovanou svoji:

- pravděpodobnost úspěchu zakouzlení
- poškození nepřítele - poškození, které nepřítel utrpí
- cenu many, která se kouzelníkovi odečte
- cenu energie
- dobu trvání, kdy kouzelník nemůže dělat nic jiného
- doplnění zdraví, které se kouzelníkovi přičte
- doplnění své many, pokud kouzlí kouzlo na doplnění many

Systém by měl umožňovat snadnou a rychlou změnu konfigurace kouzel tak, aby se daly přizpůsobit momentálním potřebám hry.

2.2 Studované metody umělé intelignece

2.2.1 Konečný automat (Finite State Machine)

Konečné automaty slouží k reprezentaci stavů herních postav. Formálně můžeme stavový konečný automat definovat jako uspořádanou čtveřici (S, Σ, σ, s) , kde:

- S je konečná množina stavů.
- Σ je konečná množina vstupních symbolů, nazývaná abeceda.
- σ je tzv. přechodová funkce (též přechodová tabulka).
- s je počáteční stav.

Pro naše potřeby bude množina stavů S například „Jdi", „Bojuj", „Uteč pryč", „Stůj", „Obnov" a jiné. Σ je množina všech možných akcí a podnětů, které se vztahují k dané postavě, například „Nový úkol," „Nepřítel zabit," „Došla mana," „Dochází zdraví" a spousty jiných, které si návrhář vymyslí. V přechodové funkci σ je potom zapsáno, jak

na kterou akci automat zareaguje a do jakého stavu přejde. Dá se snadno reprezentovat přechodovou tabulkou. Automaty můžeme rozdělit na deterministické a nedeterministické. Determinstický má v přechodové tabulce ke každé akci nanejvýš jeden stav, do kterého může přejít. Naproti tomu u nedeterministického může být pro každou akci celá množina dalších možných stavů. Mezi nimi se pak může přejít na základě pravděpodobností nebo i jiných kritérií. Tím se ovšem trochu vytrácí nedeterminismus, protože vlastně určujeme pravidla pro přechod na konkrétní stavy. Nedeterministický automat lze převést na deterministický. Konečné automaty jsou často používaná technika. Rozdělují komplexní chování do menších a jednodušších celků. Dobře se synchronizují s okolními událostmi, např. zvuky. Ale se stoupající složitostí automatu se ztrácí přehlednost a snadná rozšiřitelnost.

2.2.2 Dynamické programování (plánování)

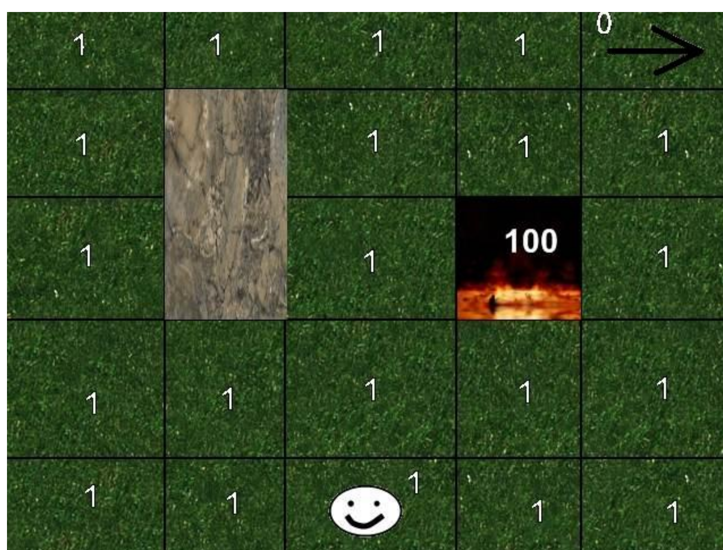
Dynamické programování je způsob, jak se vypořádat s náhodou u rozhodování umělé inteligence. Algoritmus hledá nejlepší cestu mezi dvěma body. Obecně lze použít k hledání cesty nebo i k řešení soubojů, podobně jako výše. Zde ovšem hraje velkou roli, že přechody mezi různými stavy nejsou deterministické a na základě jedné akce se můžeme dostat do několika různých stavů, které mají svojí pravděpodobnost. Vezměme si ukázkový příklad na hledání cesty mezi dvěma body. Na malé mapce zobrazené na obrázku 1 je naše postavička (bílý obličej), která se chce dostat ven (šipka v pravo nahoře). Možné akce jsou jdi na sever, jdi na jih, jdi na západ, jdi na východ. Přičemž pro akci „jdi na sever“ platí, že postavička s pravděpodobností 50% půjde na sever, s pravděpodobností 25% půjde na západ a s pravděpodobností 25% půjde na východ. Pro akci jdi na západ platí, že s pravděpodobností 50% půjde na západ a s pravděpodobnostmi 25% půjde na sever nebo na jih. Pro zbylé akce platí obdobné. Na mapce je skála, kam se jít nemůže, a také políčko s ohněm, kde hrozí naší postavičce nebezpečí. Každé políčko mapky má své začáteční ohodnocení, přičemž cíl má ohodnocení nula, políčko s nebezpečím má velké ohodnocení a ostatní začínají na jedničce. Tím jsou dány vstupy algoritmu. Algoritmus funguje na iteračním principu, s každou iterací projede všechna políčka a spočíta jejich ohodnocení. Čím více iterací, tím lepší výsledek. Nastínění algoritmu v pseudo kódu:

```
Inicializuj všechny buňky v poli V na vstupní hodnoty
Pro každou buňku v mapě
    Pro každou možnou akci v aktuální buňce
        Spočti ohodnocení akce U
        Vyber nejmenší z ohodnocení Us
        Nastav ohodnocení buňky na V = Us + ohodnocení buňky
```

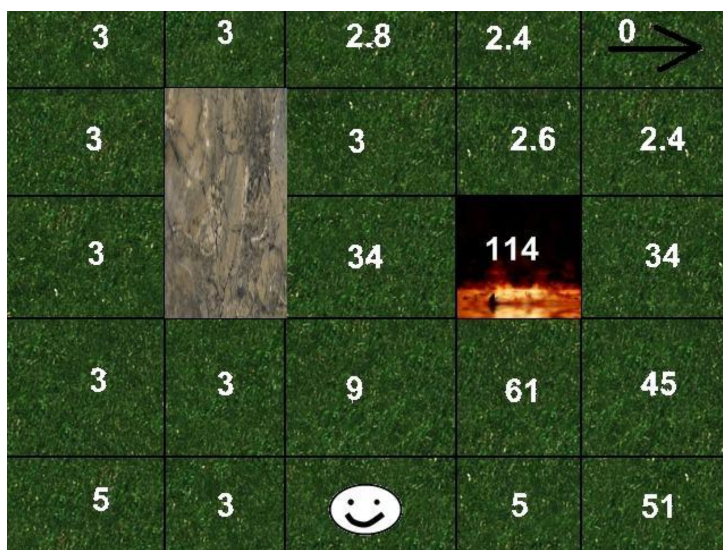
K bodu *Spočti ohodnocení akce U.* Ohodnocení jednotlivých políček je stanoveno dle políček, na které se může přejít na základě akce, a jejích pravděpodobností. Tedy máme-li akci jdi na sever potom ohodnocení políčka bude:

```
hodnota políčka na severu * 0.5 + hodnota políčka na západě * 0.25
+ hodnota políčka na východě * 0.25
```

Ohodnocení se tak nasčítavá a políčka kolem onoho nebezpečného tak získávají vyšší



Obrázek 1: Dynamické pogramování, mapa, počáteční stav



Obrázek 2: Dynamické pogramování, mapa po několika iteracích

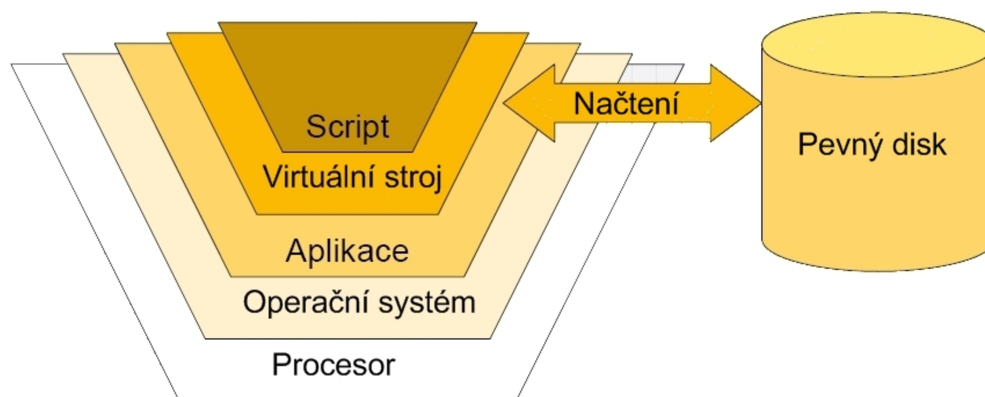
a vyší hodnotu.
Máme-li takto ohodnocenou mapu, jak je ukázáno na obrázku 2, můžeme vybrat nejlepší akci pro každé políčko:

```
Inicializuj pole akcí A
Pro každou buňku v mapě
    Pro každou možnou akci v aktuální buňce
        Spočti ohodnocení akce U na základě pole ohodnocení V
        Ulož akci s nejmenším ohodnocením do pole akcí A
```

Pole nejlepších akcí A je stejně velké jako pole buněk V, tedy každé buňce bude příslušet jedna vybraná akce. Akční hodnota U je vyvozena na základě ohodnocení buněk, na které by se mohlo přejít na základě akce.
Další možné užití této techniky, zřejmě i užitečnější, je v soubojích. Představme si situaci, kde kouzelník ovládananý umělou inteligenci je v souboji s hráčem. Jeho cílem je smrt hráče. Má k dispozici útočné kouzlo a uzdravovací kouzlo. Každé kouzlo má definovanou svoji pravděpodobnost úspěchu. Definujme stav souboje, tak aby pokryl vše co se týká souboje, tedy jako vektor (hp NPC, hp protivnik, mana NPC, mana protivnik), cílový stav potom je `(hp větší než 0, 0,-,-)`. Pomlčky znamenají, že na těchto hodnotách nezáleží. Jednotlivé akce spojují stavy dohromady, tedy například ze stavu `(200, 550, 100, 120)` se akcí uzdravovací kouzlo dostaneme do stavu `(350, 550, 50, 120)` s pravděpodobností `p`, která je definována v dané akci. Stavový prostor se potom skládá ze všech možných stavů, které mohou nastat. Tím jsme nadefinovali vše potřebné a můžeme použít výše uvedený algoritmus.
Z popsaného plyne, že na větších mapách, popřípadě s většími stavovými prostory, je náročnost algoritmu velmi vysoká. Je tedy třeba uvážit některé optimalizace.Například jde-li o mapy, počítat cestu po částech na menších mapách. Dále je třeba si uvědomit, že algoritmus produkuje již nějaké výsledky po pár iteracích, dokonce i po první iteraci. Teorie říká, že ke správnému řešení konverguje algoritmus vždycky a obecně je třeba přibližně tolik iterací, kolik je třeba kroků do cíle. Ale my nepotřebujeme vždycky nejlepší výsledky. Je také možné počítat výsledky za běhu. Tedy již po několika málo iteracích udělat první krok a s každym dalším krokem vylepšovat výsledky. I tak jsou nároky značné jak na procesorový čas, tak na pamět. Podrobnější informace včetně příkladu implementace v [GPG404]

2.2.3 Hledání cesty (Path finding)

Hledání cesty zjednodušeně znamená jak se dostat z místa A do místa B a pokud možno se vyhnout překážkám a slepým uličkám. Nejznámějším a asi i nejpoužívanějším algoritmem je **A***.
Algoritmus začíná s bodem A a všchny jeho možné následovníky (sousedy) si uloží do `open` seznamu. Ti jsou pak ohodnoceni. Nejlepší z nich je vybrán, přesunut do `closed` seznamu a jeho následovníci opět dáni do open seznamu a ohodnoceni. To se opakuje, dokud se nedojde až do cíle. Ohodnocení se skládá ze dvou částí: g a h. V g se nasčítává délka cesty od začátku do políčka, v kterém zrovna algoritmus je, a h, také zvaná heuristická část, je ohodnocení ze současného políčka do cílového.



Obrázek 3: Ilustrace virtuálního stroje pro spouštění skriptů (obrázek převzat z přednášek předmětu KIV/PH)

Další algoritmus path findingu je Dijkstrův algoritmus, který je obdobný A*, ale nepoužívá heuristiku, takže je obecně pomalejší. Pokud u A* vynecháme h složku ohodnocení, máme vlastně Dijkstrův algoritmus.
Naivní variantou hledání cesty je prohledávání stromu do hloubky nebo do šířky.
Pěkné vysvětlení a úpravy A* algoritmu jsou v [GPG100]. Rozšířené možnosti pathfindigu jsou v [GAI05]

2.2.4 Skriptování

Jsou to metody reprezentace procedurálních znalostí nebo jinými slovy, reprezentují stereotypní události. Dovolují herním designérům vytvořit a řídit jednak běžné i unikátní situace a držet hráče na příběhové linii. Jako skriptovací jazyky se používají Lua, Python, Perl, Ruby, C, C# a mnoho jiných. Také se někdy používá vlastní jazyk, který si vývojáři přizpůsobí své hře. Potom je však nutné jazyk navrhnout a vytvořit překladač, popřípadě interpreter.
Skripty mohou být velmi komplexní části kódu, nebo malé jednoduché funkce, záleží na vývojářích, jak je použijí.
Skripty ve hrách můžeme rozdělit na dva typy:

- Herně-technické - skripty lokací, postav (rozmístění, vlastnosti), pastí atd.
- AI skripty postav - umožňují například řídit chování v boji, kouzlení, dialogy, interakce s hráčem a jiné.

Skripty jsou často interpretované. Běží ve virtuálním stroji jakoby na jiném procesoru a v jiné paměti. Princip je ukázán na obrázku 3. Jsou používané pro jejich možnost měnit vlastnosti hry a postav bez kompilace celé aplikace. Často je nepíší programátoři, ale designéři. Z toho vyplývá požadavek na jednoduchost jazyka a snadnost naučení. Dále je pro využití ve hrách důležitá rychlost provádění. Nevýhodou skritpovacích jazyků je náročnost na ladění a hledání chyb. Možnosti krokování, sledování proměnných a jiné samozřejmosti, jsou zde většinou nemožné. Dále, pokud má skripty psát programátor, který je zvyklý na moderní integrovaná vývojová prostředí, jež velkou měrou

zjednodušují a zpříjemňují práci, tak ve většině případů může na tyto vymoženosti zapomenout.

Lua jako skriptovací jazyk [Lua]
Lua je ve hrách hojně využívaný jazyk. Je implementován jako knihovna napsaná v ANSI C. Je poměrně minimalistický a tím i nenáročný na systémové prostředky. I proto se hodí do počítačových her. Syntaxe je mix mezi jazyky C a Pascal, proměnné jsou netypované. Rozlišuje tyto datové typy:

- čísla - nezáleží zda celá nebo s desetinou čárkou (implicitně s dvojitou přesností plovoucí čárky)
- boolean
- řetězce - uzavřeny v uvozovkách popř '[[' a ']]'
- funkce
- thread
- tabulky - asociativní pole (indexované podle klíčů)
- uživatelská data - speciální typ odpovídající ukazatelům
- nil - neinicializovaná proměnná.

Dále obsahuje běžné operátory, podmínky a cykly.
Lua podporuje jak klasické procedurální, tak objektové programování. To se realizuje pomocí tabulek (a metatabulek). Ty totiž mohou obsahovat jak například čísla, tak i reference na funkce a uživatelská data. Tabulky jsou jediný, zato mocný nástroj pro kolekce. Obsahuje také garbage collector, tedy nástroj pro úklid paměti (stejně jako jazyky z rodiny .NET). Vzhledem ke svoji minimalističnosti je jednoduchá na naučení. Lua je volně ke stažení.

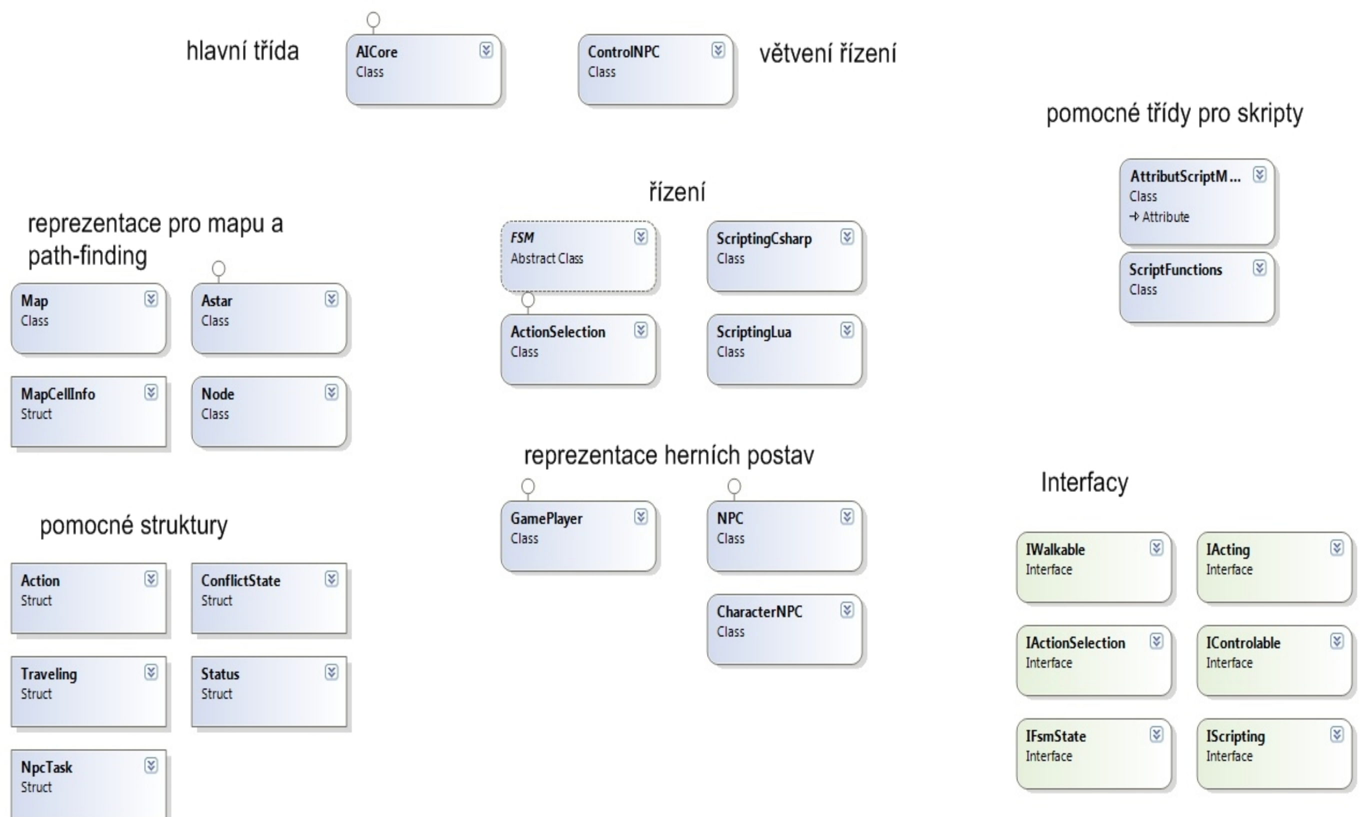
Python jako skriptovací jazyk [Pyth]
Python je interpretovaný, objektově orientovaný programovací jazyk. Oproti jazyku Lua je více mocný, ale také náročnější na systémové prostředky. Je navržen tak, aby v se v něm mohly psát plnohodnotné aplikace i s uživatelským rozhraním. Je vyvíjen jako open-source projekt. Obsahuje třídy, má silnou typovou kontrolu. Proměnné se chápou jako pojmenované odkazy na objekt, funkce se též chovají jako objekt, dokud nejsou zavolány. Do složených datových struktur (pole) se ukládají odkazy na objekty. Mezi hlavní výhody patří jednoduchost jazyka. Python vychází, mimo jiných, z jazyka ABC, který je výukový. I proto je vhodný také pro začátečníky. Existuje několik implementací. Původní je CPython psaný v jazyce C, dále existuje Jython (java) IronPython (.NET).

C# jako skriptovací jazyk [cs]
C# je kompilovaný, objektově orientovaný jazyk z rodiny .NET navržený společností Microsoft. Syntaxe vychází z jazyka C++, je ovlivněn především jazyky Java a Delphi. Jazyk je navržen pro obecné použití s důrazem na jednoduchost a produktivitu. Má silnou typovou kontrolu (od verze 3.0 přibyl i netypový objekt), garbage collector. Protože se jako všechny jazyky z rodiny .NET překládá do mezikódu - Common Intermediate Language, není tak rychlý jako C, případně C++. Jako skriptovací jazyk není příliš známý, avšak ve spojení s aplikací napsanou v jazyku z .NETu je toto docela jednoduše možné.
Další informace a odkazy v [WikiScript] a [GAI05]

3 Realizační část

3.1 Návrh knihovny umělé inteligence

Celá knihovna je napsaná v jazyce C#, využívá tedy služeb Microsoft .NET Frameworku, který je nutný k běhu ve verzi 2.0. Také se drží principů objektového programování. Pro fungování knihovny je třeba mít informace o mapě, na které se hra odehravá, reprezentaci postav, které mají být ovládány. Dále jsou třeba třídy pro řízení a také reprezentaci vybraných metod umělé inteligence. Na zjednodušeném UML diagramu jsou zobrazeny hlavní třídy, struktury a interfacy (obrázek 4). Popis hlavních tříd:



Obrázek 4: zjednodušný UML diagram knihovny s komentáři

- `AICore` - Hlavní třída, která inicializuje a spouští všechny ostatní. Přes tuto třídu lze přidávat NPC postavy. Obsahuje funkci `Update()`, která je volána z hlavní herní smyčky, v pravidelných intervalech. Ta pak dále volá Update třídy ControlNPC.

- `ControlNPC` - Zde se rozděluje řízení podle typu NPC. Volá se Update pro každé NPC.

- `NPC` - Reprezentuje NPC postavu. Implementuje interface `IActioning`, který je použit dále při rozhodování v soubojích. Obsahuje struktury `Status` pro uložení

informací o aktuálním stavu jako jsou hit body, mana, počet nepřátel v dosahu a jiné, `CharacterNPC`, který říká, jaké jsou inicializační vlastnosti postavy (hit body, mana, level) a také jaké jsou jeho priority, které reprezentuje struktura `Priorities`. Ty zahrnují prioritu pro přežití, zabíjení a jiné. Dají se použít při rozhodovaní, jaké akce použít nebo při řešení do jakého stavu z množiny možných se přejde. Je zde také struktura `NpcTask`, ve které jsou uloženy cíle postavy.

- `GamePlayer` - Reprezentuje hráče, myšleno člověka, který sedí u počítače a hraje. Též implementuje interface `IActioning`. Tato třída je tu proto, aby NPC mohly interagovat s hráčem, především ve smyslu soubojů. Udřžuje také svůj `Status`.

- `Map` - Třída pro reprezentaci mapy. Používá návrhový vzor *Singleton*. Vychází z výškové mapy. Je udržována jako 2D pole struktur `MapCellInfo`, kde každá představuje jedno políčko (buňku). Tato struktura obsahuje informace o výšce, obsazení jinými postavami nebo překážkou. Na každé buňce může být jen jedna postava, pokud v ní už není překážka. Poskytuje NPC postavám informace o jejich okolí a také slouží jako stavový prostor pro path finding.

- `FSM` - Abstraktní třída sloužící pro definici konečných stavových automatů. Viz bod *Konečný automat*

- `Astar` - Třída obsahující A* algoritmus pro hledání nejlepší cesty. Jako uzly, které prohledává, slouží třída `Node`, které obsahují informaci o pozici, heuristice a také z `MapCellInfo`. Více v bodu *A* algoritmus*.

- `ActionSelection` - Tato třída má na starosti výběr akce v soubojích. Výběru akce je věnován vlastní bod dále. Využívá struktury `ConflictState` pro uchování informace o současném stavu konfliktu. Více v bodu *Výběr další akce*

- `ScriptingCsharp` - Tato třída implementuje podporu načítání skriptů v jazyce C#, jejich překlad za běhu a volání funkcí z těchto skriptů. Využívá jmeného prostoru `System.Reflection` a `System.CodeDom.Compiler`

- `ScriptingLua` - Obdobné jako výše, slouží pro načítání a spouštění skriptů v jazyce Lua.

- `ScriptFunctions` - Tato třída je jakousi branou pro skripty do knihovny umělé inteligence. Obsahuje funkce, které jsou určené pro volání ze skriptů.

3.2 Propojení s herním enginem

Pro inicializaci knihovny je třeba předat v konstruktoru pouze třídu implementující interface `IWalkable`, tedy třídu pro terén. Tento interface obsahuje funkce pro zjištění informací o mapě, překážkách, velikosti, také funkce pro převod 3D pozice do 2D mřížky, kterou používá knihovna, a obráceně. Tím získá knihovna kompletní informaci o mapě. Pro každé přidané NPC se načítají tyto informace:

- Jméno - slouží k načtení grafického modelu a k identifikaci NPC.

- Ovládací mechanizmus - buď jméno konečného automatu nebo jméno skriptiu, kterým se bude postava řídit.
- Pozice - startovní pozice. Konkrétně 2D pozice v mřížce.
- Jméno charakteru - slouží k načtení správného charakteru (viz níže). Více NPC může mít stejný charakter.
- Akce - jméno souboru akcí, které NPC může používat.

Konfigurační XML pak může mít následující podobu:

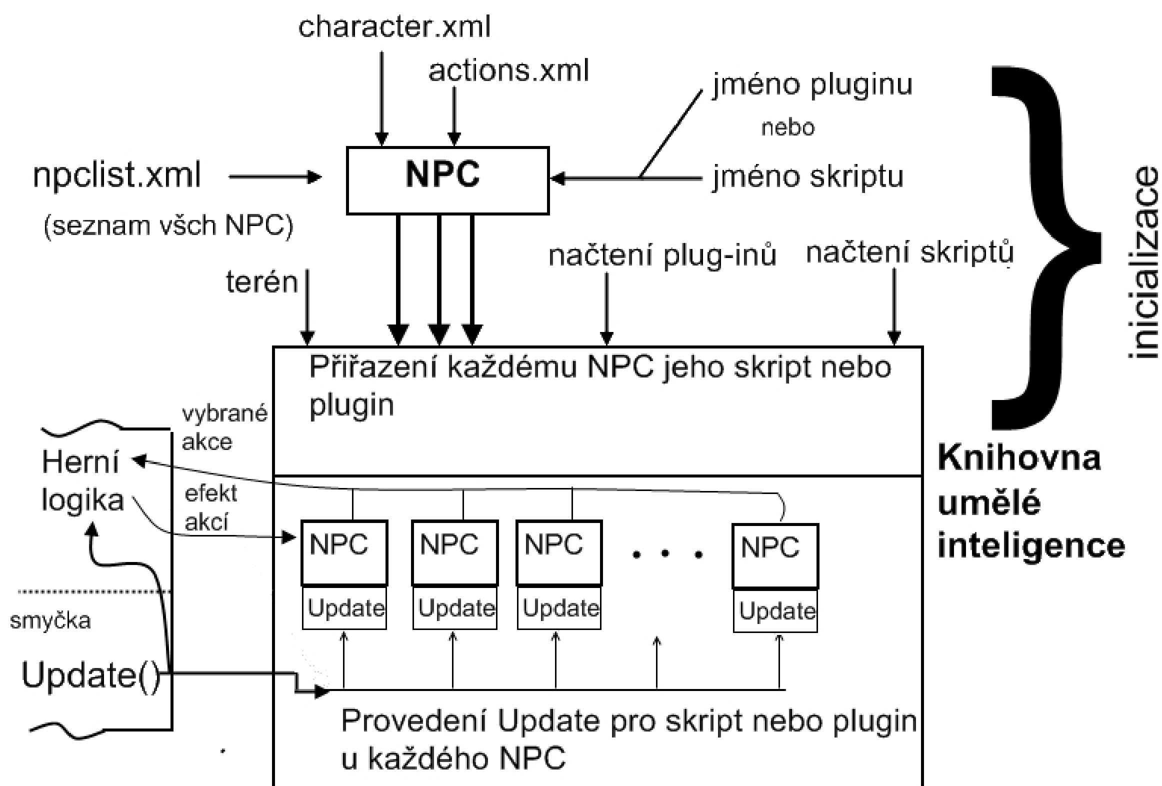
```
  <NPC>
    <Name>Ton</Name>
<!-- Pokud chceme použít plug-in: -->
    <FSM>BasicFSM</FSM>
<!-- nebo -->
<!-- pokud chceme použít skript: -->
<Script>jmeno_skriptu.lua</Script>
    <Position>
      <X>8</X>
      <Z>98</Z>
    </Position>
    <Character>Ton</Character>
    <Actions>AttackSpells</Actions>
  </NPC>
```

Dále při přidávání NPC se musí do knihovny předat objekt implementující interface `IControlable`, který představuje grafickou reprezentaci postav. Obsahuje funkce pro změnu pozice a směru, funkce pro vykonání efektů kouzel, smrti apod. Pokud si NPC vybere nějakou akci, pošle informace o této akci do herní logiky přes tento interface. Pokud daná akce má nějaké další efekty, pošlou se tyto výsledné efekty z herní logiky do knihovny umělé inteligence. Typickým příkladem je vystřelení útočného fireballu. NPC se rozhodne tento vystřelit, pošle zprávu a nutné informace do herní logiky. Ta zajistí vytvoření efektu ohně a vystřelení směrem k nepříteli. Po nějakém čase může tento fireball zasáhnout jiné NPC (nejspíš to, na které byl mířený, ale klidně jakékoliv jiné), potom se pošle do knihovny stejná zpráva jako předtím posílalo NPC do herní logiky, jen s tím, že došlo k zásahu. V knihovně se pro zasažené NPC aktualizuje jeho status (odečtou se hitbody).
Ke každému NPC je připojen jeho charakter, který definuje základní vstupní informace o NPC jako zdraví (hp), sílu, level a další. Je definovaný XML souborem a má podobu následujícího příkladu.

```
<NPC>
  <Name>Ton</Name>
  <Side>2</Side>
  <Type>3</Type>
  <Attributes hp="870"level="6" mana="250"
       power="5" defense="8"></Attributes>
```

```
  <Priors live="70" kill="60"></Priors>
</NPC>
```

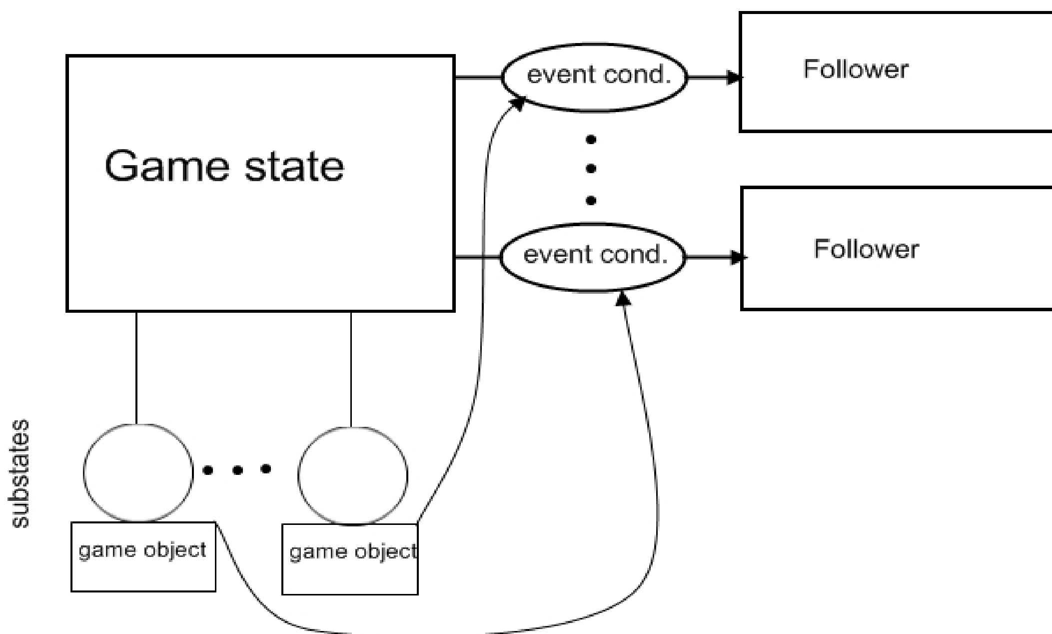
Schéma inicializace a běhu knihovny je na obrázku 5.



Obrázek 5: Schéma knihovny umělé inteligence

Každá správná hra by měla mít svůj děj nebo alespoň nějaké naznačení. Ta naše jej má tak jak je popsáno na začátku této práce. Každopádně v ději hrají svoji roli i počítačem ovládané postavy. Proto je nutné ještě další spojení s herním enginem. K tomu, abych mohl toto další propojení představit je třeba si ukázat jak je děj reprezentován v herním enginu.

Hra má definované své herní stavy a funguje v ní dějový konečný automat. Herní stav je ilustrován na obrázku 6. K němu je ovšem třeba další popis. Herní stav má svoje podstavy. Dále má své následovníky, další herní stavy. Podstavy jsou vždy svázané s jedním objektem ve hře, ať už je to NPC, dveře, socha nebo cokoliv jiného, a s jednou akcí nebo jinak, událostí (například označení, zabití, příchod, ...). Ve výsledku se potom zkoumá, od jakého objektu přišla jaká událost a co to znamená pro automat, jestli přejde do jiného stavu (podstavu) a jakého nebo se neděje nic. Pro každého následníka musí minimálně jeden podstav odpovídat podmínce přechodu. Pokud se pak automat dostane do takového podstavu, přejde se do odpovídajícího následníka. Takto nadefi-



Obrázek 6: Znázornění stavu hry

novaný děj je uložený v XML souboru a dá se opět libovolně konfigurovat.
Příklad jednoho stavu:

```
<GameState>
  <Name>Obelisc</Name>
  <Followers>
    <Follower condition="obeliscsdone">Forest</Follower>
  </Followers>
  <Substates order="true">
    <Substate object="obelisc1">obelisc1</Substate>
    <Substate object="obelisc2">obelisc2</Substate>
    <Substate object="obelisc3">obelisc3</Substate>
    <Substate object="obelisc4">obeliscsdone</Substate>
                  <!--tento podstastav je přechodový-->
  </Substates>
</GameState>
```

Když se herní stav změní, je tento přes funkci `Update()` předán do knihovny AI. V ní se zavolá skript, který dostane informaci o stavu a jeho výstupem je definice úkolu pro NPC postavy. Tyto úkoly se pak ukládají do fronty pro každé NPC a to je postupně vykonává. Úkol zastupuje struktura `NpcTask`, která obsahuje položky:

- seznam lokací - tedy bodů, kam musí NPC dojít
- parametr určující zda záleží na pořadí seznamu lokací
- čas, který NPC má na to, aby úkol splnil. Možnost uplatnit časový limit byla přidána z důvodu zabranění situacím, kdy by mohly NPC při potkání nepřátelského

NPC zvolit útěk a to opakovaně, a nikdy by se nedostaly k plnění cíle. Takhle je možnost uplatnit zbývající čas jako kritérium při rozhodování o chování.

Těmito body se tedy dá definovat úkol pro NPC. O implementaci skriptů je sekce „Použití skriptů“.

3.3 Implementace vybraných metod

Pro path-finding jsem vybral klasickou metodu A* algoritmu. Pro řízení NPC jsem porovnal dvě metody, konečné automaty a použití skriptů. Porovnávání se týkalo jednoduchosti a efektivnosti imlementace a také rychlosti provedení v programu. Také je v knihovně implementována třída pro Action selection, tu využívají konečné automaty, aby v jednom konkrétním stavu vybraly další akci, kterou bude NPC dělat. Pro tyto potřeby se nabízela metoda Dynamického programování, jež je zmíněna v sekci 2.2.3, ale ta nebyla použita vzhledem k tomu, že pro potřeby navrženého scénáře plně postačí jednodušší varianta bez plánování. Tím ušetříme výkon a paměť.

3.3.1 Konečný stavový automat

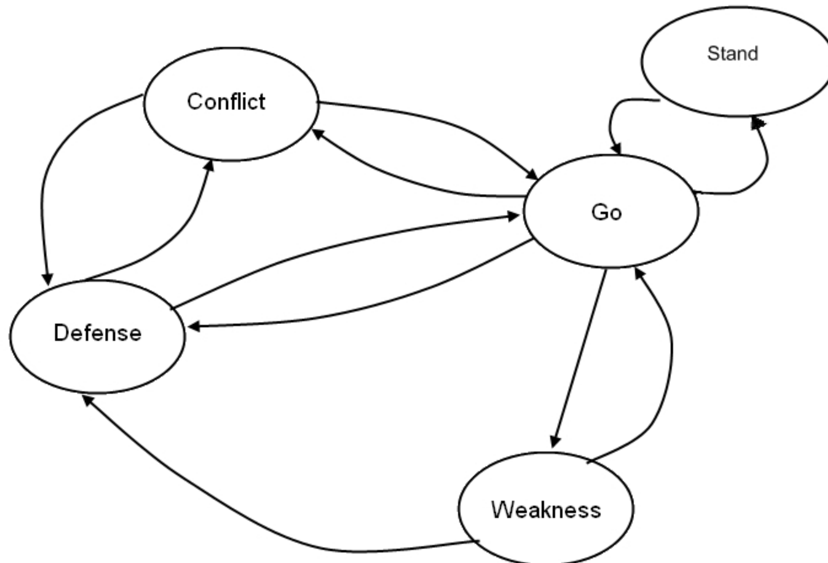
Konečný automat a jeho stavy určují, jak se postava chová. Zde je proto nanejvýš vhodné, aby šlo snadno měnit nastavení automatu, případně automat sám. Proto jsem zvolil architekturu **plug-in**ů. V knihovně je definován interface `IFsmState` pro stavy automatu a abstraktní třída `FSM`, která deklaruje abstraktní funkce:

- `Update()` - měla by sloužit pro kontrolu situace a případně změnit stav automatu.
- `React(AI.AiEvents action)` - Umožní přímo zareagovat na akci přímo.
- `CheckState(AI.IActionSelection actionSelection)` - V této funkci by mělo být zajištěno chování postavy podle stavu ve kterém se nachází. Tedy například nachází-li se v bojovém stavu, potom výběr kouzla které se bude kouzlit apod.

Z této třídy pak musí vycházet všechny plug-iny, které mají být konečnými automaty. Z hlavního Updatu AI je voláno pro každou postavu `Update()` automatu následovaný `CheckState(AI.IActionSelection actionSelection)`, pokud tedy postava má připojený nějaký automat. Využití interfacu `AI.IActionSelection` přináší výhodu zapojení jakéhokoli mechanizmu výběru akce. Plug-iny jsou do programu načítány za běhu z předem dané složky v podobě dll knihovny. K načtení a spuštění se používá jmenný prostor `System.Reflection`, který obsahuje vše potřebné. Z toho všeho vyplývá, že chování postav je plně modifikovatelné z vnějšku programu. To je velmi výhodné z hlediska modifikovatelnosti hry, tak jak je zrovna potřeba nebo jak si herní návrhář zrovna vymyslí. Hlavní výhoda pro programátora je ve snadnosti ladění.

Základní konkrétní navržený automat, který je jako plug-in obsažen již v základu, má následující podobu.

Je načten z XML souboru, kde je uloženo jméno stavu a možní následníci stavu. Jednotlivé stavy vymezují množinu akcí (kouzel), které má postava k dispozici. Mezi stavy se přechází na základě podnětů z mapy, zbývajícího času na úkol a charakteru postavy. V tomto charakteru jsou nastaveny priority například pro přežití, zabíjení a jiné. Všechny možné přechody jsou znázorněny na obrázku 7. Možné stavy jsou:



Obrázek 7: Konečný automat

- *Conflict* - Bojový, útočný mód. Rozhodují se na základě vlastního stavu (HP, mana) a zároveň stavu a síle nepřátel. Možnost útěku je znevýhodněna. Do tohoto stavu se může přejít po zjištění přítomnosti nepřítele, případně nepřátel. V tomto stavu se používají kouzla tak jak jsou původně definována.
- *Defense* - Bojový, obranný mód, kde NPC mají k dispozici lehké útočné kouzlo, kouzlo pro doplnění many a silné uzdravovací kouzlo, zároveň může utéct z boje. Rozhoduje se na základě stejných podkladů jako výše. Do tohoto stavu se může přejít po zjištění přesily nepřátel, nebo pokud má velmi málo hit bodů. Při aplikaci kouzel se zvýší účinnost uzdravovacích kouzel a naopak sníží účinnost útočných kouzel
- `RunAway` - NPC se dá na útěk. Vybírá se, pokud stojí proti přesile nebo je již příliš slabé. Směr kam utíkat se určí podle rozmístění nepřátel a podle cíle NPC. Tedy NPC se snaží utíkat pryč od nepřátel a pokud možno ke svému cíl..
- *Weakness* - Odpočinkový stav, NPC musí nabrat energii, protože jí má nedostatek. V tomto stavu zastaví cestu za cílem a začne odpočívat. Nejsou k dispozici žádné akce, jen se prostě odpočívá.

- *Go* - Stav kdy je NPC samo a jde za svým dějovým cílem. Během cesty se nerozhoduje o žádných akcích, jen se kontrolují možné přechody do jiných stavů (například jestli nevidí nepřítele).
- *Stand* - Počáteční stav, kdy NPC nemá žádné úkoly a nemůže útočit. Může pouze přejít do stavu Go ve chvíli kdy obdrží nějaký úkol.

Popsaný automat je určen pro hlavní protihráče. Dále knihovna obsahuje další 2 automaty, které je možné načíst. Ty jsou zjednodušené oproti automatu výše a jsou určené například pro stráže nebo divoká zvířata.

3.3.2 Použití skriptů

Alternativa k použití plug-inů je použití skriptů.

Řízení NPC Pro skripty, které by řídily NPC, jsem vybral jazyk Lua. Důvodem je převším jeho jednoduchost a výkon, který je v počítačových hrách vždy klíčový. Lua je psaná v jazyce C, takže ji nelze použít přímo. Pro napojení na C++ se používá LuaBind [Luabind], pro napojení na jazyky rodiny .NET potom LuaInterface [LuaInterface]. Ten je použit v knihovně. Umožňuje jednoduše přistoupit na funkce ze skriptu, proměnné a registrovat funkce pro skript. Ukázka inicializace a provedení skriptu:

```
LuaInterface.Lua lua = new LuaInterface.Lua();
lua.DoFile(path);
//registrace funkci, ktere muze skript pouzivat
RegisterFunctions(scriptFunctions);
//pristup k funkcim skriptu
object[] results = lua.GetFunction(name).Call(par);
//pristup ke globalnim promennym ze skriptu pres indexer
int i = (int) lua[variableName];
```

Psaní skriptů v jazyce Lua je celkem snadno naučitelné a rychle přejde do krve. Obzvlášť ze začátku se může velmi hodit výborný návod [LuaTutor]. Přes LuaInterface lze předat do skriptu celé struktury a přistupovat k jednotlivým proměnným, což usnadňuje práci, protože můžeme využít již nadefinované struktury pro status a jiné. Registrace funkcí, které lze ze skriptu volat se provadí:

```
lua.RegisterFunction(methodName, target, info);
```

Zde je `target` objekt třídy, ve které se funkce nachází a `info` je objekt třídy `MethodInfo`, který popisuje registrovanou funkci. Zde se stále různě pletou pojmy funkce a metody. Abych toto ujasnil, z pozice lua skriptu se jedná o funkce, ale z pozice knihovny, to jsou metody. Mechanismus registrování metod je automatizován pomocí atributů. V knihovně je deklarovaná třída `AttributScriptMethod`, která slouží jako atribut. Metody, které se mají registrovat do skriptu se pak jednoduše označí tímto atributem. Při načítaní se pak vezme celá třída a všechny její metody označené tímto atributem jsou zaregistrovány. Třída, která má tento úkol je `ScriptFunctions`.
Knihovna obsahuje v základu 3 lua skripty pro ovládání postav, jsou určeny pro hlavní

nepřátele, kteří soupeří s hráčem, pro finálního nepřítele (tzv. bosse) a poslední je obecný pro různá monstra.
Skripty mají následující podobu. Rozdělují se na tzv. stavy, které jsou v jazyku lua reprezentované tabulkou. každý stav má 3 funkce, které se volají pří vstupu do skriptu, pro vykonání akce ve stavu a pro vystoupení ze stavu. To má výhodu v tom, že se mohou při přechodech mezi stavy volat funkce na mluvení, zvukové efekty a jiné. Ke každému stavu je připojena jedna akce. Při vykonání stavu (volá se z funkce Update) se buď přejde do jiného stavu nebo se vykoná připojená akce. Tento postup je inspirován konečnými automaty.
Zjednodušený, okomentovaný příklad:

```
--definice tabulek
State_stand = {};
State_attack = {};
State_run = {};
State_heal = {};
--a další

--inicializace
actualState = State_stand

--tato funkce je volána z programu
function Update()
    actualState.Execute()
end
......
--příklad jednoho stavu
State_attack["Execute"] = function()
    --zavolaní registrované funkce
    local status = GetStatus();

    if status.enemySeen  == 0 then
        ToStand(); --přechod do jiného stavu
    elseif status.hp < npc.character.hp / 4 and not alreadyRunned then
        ToRun(); --přechod do jiného stavu, utek pri malo zdravi
    else
        --zautocit
        Spell("Fireball");
    end
end

State_attack["Enter"] = function()
    --Zavolani regitrované funkce pro mluveni
    Talk("Jdu po Tobe!")
    --pridani logovací inforamce
    Log("LUA: state attack... enter");
end
```

```
State_attack["Exit"] = function()
    --pridani logovací inforamce
    Log("LUA: state attack... exiting");
end
....
--přechodová funkce
function ToStand()
    actualState.Exit();
    actualState = State_stand;
    actualState.Enter();
end
....
```

Napojení na děj Pro napojení na herní děj jsem použil skripty v jazyce C#. Výhoda tohoto výběru je přítomnost kompilátoru v .NET Frameworku, známost syntaxe, možnost využít reflexe k snadnému přístupu k funkcím skriptu a po načtení zkompilovaného modulu stejný výkon jako ostatní kód.

```
Microsoft.CSharp.CSharpCodeProvider provider =
    new Microsoft.CSharp.CSharpCodeProvider();
System.CodeDom.Compiler.CompilerResults results =
    provider.CompileAssemblyFromFile(
    this.parameters,new string[]{path});
```

V parametrech se pak předají parametry kompilátoru (zda přeložit do paměti, přidání ladicí informace,...) včetně reference na knihovnu umělé inteligence. Metoda skriptu se pak spouští opět jednoduše.

```
object result = t.GetMethod(methodName).Invoke(null, pars);
```

Zde `t` je odkaz na typ načtený ze skriptu a `pars`, jsou parametry, které se předají funkci.
Nebo můžeme z načteného a přeloženého skriptu rovnou vytvořit instanci. To má několik výhod. Za prvé rychlost provedení kódu je naprosto stejná jako jakýkoliv jiný kód v knihovně. A za druhé v konstruktoru můžeme předat libovolný objekt (například objekt třídy ScriptFunctions) a tím umožnit skriptu volat metody pro řízení.

```
IScripting script =
    (IScripting)Activator.CreateInstance(type, constructorPars);
```

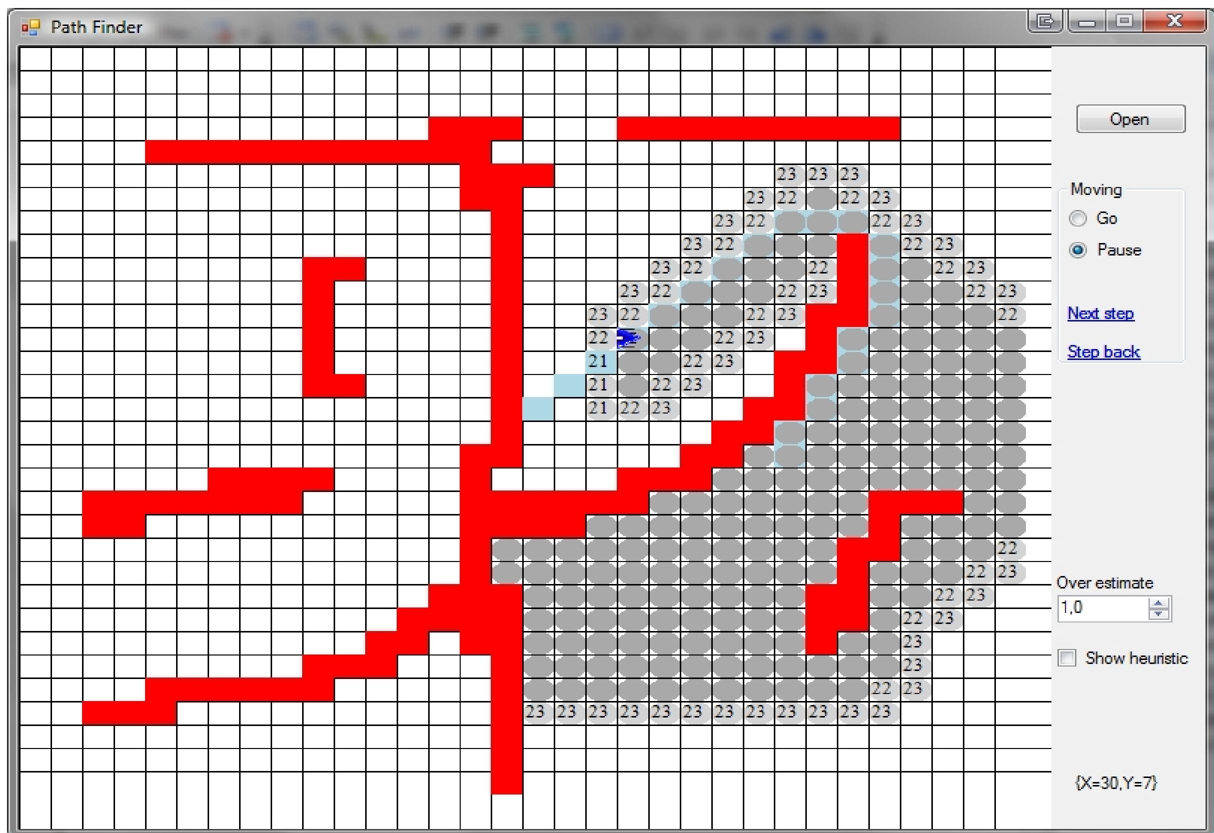
Zde `type` je načtený typ ze skriptu neboli třída, `constructorPars` je pole objektů, které chceme jako parametry předat konstruktoru. Tím získáme instanci interfacu IScripting, který v mém konkrétním případě obsahuje pouze jednu metodu:

```
object[] Update(params object[] pars);
```

Vytvořený skript má v metodě Update vytvoření úkolu pro NPC podle herního stavu, který se předá jako parametr.

3.3.3 A* algoritmus

Pro potřeby hledání cesty je asi nejlepší řešení použití A* algoritmu. A* jako stavový prostor používá 2D mapu (mřížku). Uzly jsou jednotlivá políčka mapy. Jeho nastínění:



Obrázek 8: Zobrazení hledání nejlepší cesty.

```
Inicializuj OPEN a CLOSED seznam
Vlož počáteční uzel do OPEN
Dokud není OPEN prázdný
    Vyber nejlepší uzel z OPEN
    Nastav ho jako CURRENT a odstraň ho z OPEN
    Zkontroluj jestli CURRENT není cíl
    Pro každého následníka CURRENT
        Nastav G jako CURRENT.G + cena kroku
        Jestliže je následník již v OPEN a má horší G
            ignoruj ho a pokračuj
        Jestliže je následník  již v CLOSED a má horší G
            ignoruj ho a pokračuj
        Nastav rodiče následníka na CURRENT
        Spočítej heuristiku H pro následníka
        Dej následníka do OPEN seznamu

    Dej CURRENT do CLOSED
```

Z hlediska výkonu je důležitá nejen velikost stavového prostoru, ale také především použité datové struktury. V `OPEN` seznamu se nachází jen omezené množství uzlů, potřebujeme jej mít seřazený podle ohodnocení a také jej potřebujeme rychle prohledat na uzly se stejnou pozicí. Zde se tedy jeví jako nejlepší použití prioritní fronty. Naproti tomu v CLOSED seznamu bývá velmi velké množství uzlů, obzvláště pokud jsou v mapě překážky. Nepotřebujeme jej nijak řadit, zato rychle prohledávat na výskyt uzlů se stejnou pozicí. Zde jsem zvolil jako nejjednodušší a nejrychlejší řešení vytvořit si pole o velikosti mapy (maximální počet políček, který může být v CLOSED listu). Pole na začátku obsahuje samé null hodnoty. Políčka vkládáme na index `pozice.Y * Šířka mapy + pozice.X`, kde pozice je 2D pozice políčka v mapě. Pokud chceme zjistit výskyt políčka v seznamu, pak pouze vezmeme jeho pozici, vytvoříme index a ověříme, zda v poli na tomto indexu je null. Odebírání z listu je jednoduše přiřazení null na daný index.

Velmi užitečné informace k nalezení na [Amit]

3.3.4 Výběr další akce (Action selection)

Nastane-li konflikt mezi NPC a hráčem nebo NPC samotnými, potom je třeba vybírat akce, které budou NPC vykonávat. Každé NPC má definované svoje akce, které má k dispozici. Mezi akce se řadí převážně kouzla, které může NPC používat. Prvky, které akci definují, jsou popsány v kapitole Design kouzel. Jejich nastavení je načítáno z konfiguračního XML souboru, to kvůli snadnému vyladění akcí a jednoduché změně, bez nutnosti zasahovat do kódu.
Implementovaná varianta výběru další akce vychází pouze z aktuální situace a stavu. Podle něj se pak NPC rozhodne. Zde ještě navíc je třeba uvažovat, že některé akce mají svou procentuální šanci na úspěch a jsou další dva možné efekty dopadu akce. Algoritmus tedy probíhá tak, že ze vlastního statusu a statusu soupeře (struktura `Status`) se definuje stav konfliktu (strukura `ConflictState`). Dále se na ConflictState aplikují jednotlivé akce a vyvodí se všechny tři možné efekty - úspěšné provedení akce, neúspěšné bez efektu a neúspěšné se zraněním postavy, která akci prováděla. Tím se vytvoří nové ConflictState struktury, a to pro každou akci tři. K nim se přiřadí pravděpodobnosti podle toho, jak jsou nadefinované u akce. Ještě se zjistí, jestli stavy jsou platné, pokud například hitbody nepřesáhly maximum nebo mana neklesla pod nulu. Vznikne nám seznam párů ConflictState - pravděpodobnost (v kódu struktura `StateProbPair`). Připomeňme, že každá postava má mimo jiné definované své hit body, manu a energii. Z těchto parametrů pak vychází při ohodnocování. Konkrétně bere v potaz:

- vlastní hit body - položka *myHpRating*, je zde pro uzdravovací kouzla, čím méně zdraví NPC má, tím spíše se pokusí vyléčit.

- hit body protivníka - položka *enemyHpRating*, zde je účel jasný, čím méně hit bodů protivník má po zásahu kouzlem, tím lépe.

- vlastní mana - položka *myManaRating*, znevýhodňuje drahá kouzla, ale především propaguje kouzlo na doplnění many, pokud ta dochází.

- vlastní energie - položka *myManaRating*, pokud postavě dochází energie, volí méně náročná kouzla. Lépe by se uplatnila v případě většího množství kouzel.
- čas nutný k provedení - položka *timeRating*, čím delší kouzlo, tím hůře. Ovšem tato položka zatím u tak málo akcí nemá většího vlivu.

Ohodnocení se pak sestaví jednoduše podle vzorce:

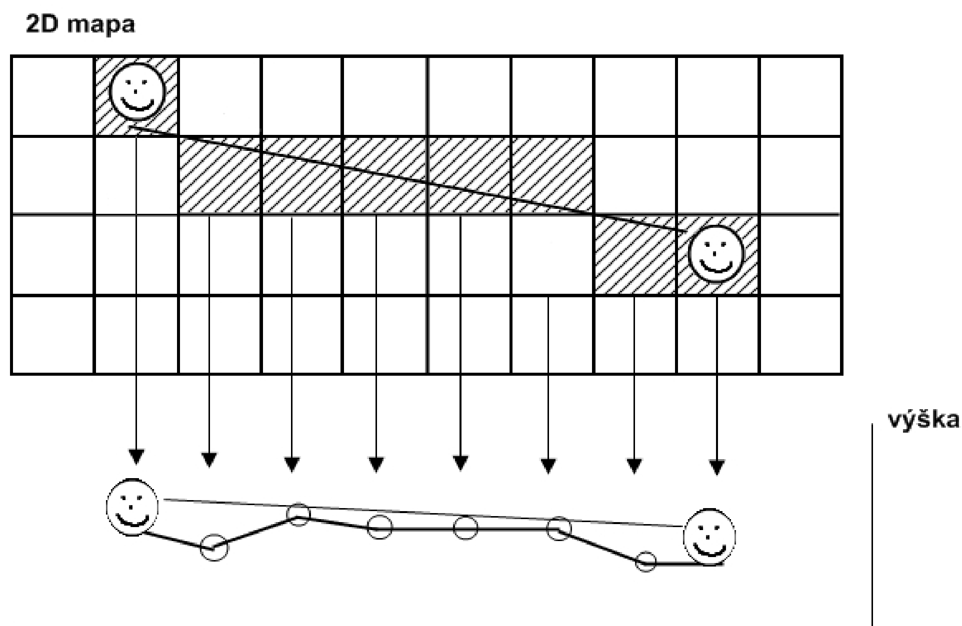
$$\begin{aligned} *sum &= myHpRating + enemyHpRating + energyRating + manaRating + timeRating \\ sum &= sum \cdot probability \end{aligned} \tag{1}$$

Toto se provede postupně pro všechny páry ve vytvořeném seznamu a vybere se ten, který má nejmenší ohodnocení. U tohoto postupu je velmi důležité vyvážení jednotlivých akcí, jejich účinků a ztrát, které způsobí.

3.4 Řešení viditelnosti

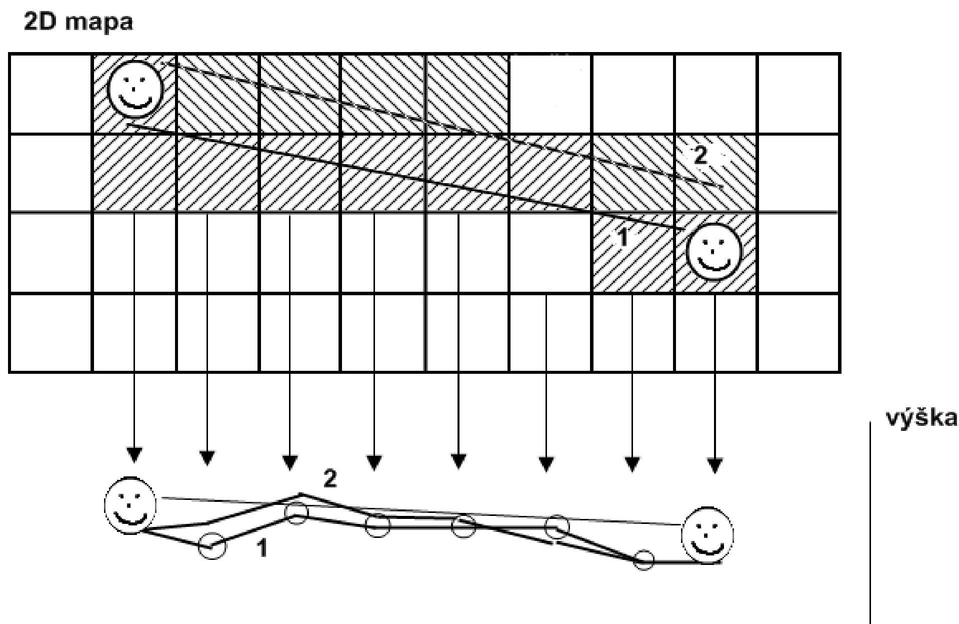
Ve hře by působilo velmi nepřirozeně, kdyby NPC útočily na hráče i zpoza kopců. Nejen proto je řešení viditelnosti důležité. Vzhledem k tomu, že máme k dispozici 2D mapu, tedy vlastně mřížku, nabízí se možnost využití Bresenhamova algoritmu [WikiBres].

Tím získáme buňky v mapě, které jsou mezi postavami. Z každé buňky odečteme



Obrázek 9: Řešení viditelnosti

výšku. Tím získáme posloupnost výšek mezi postavami. Dále mezi nimi natáhneme úsečku, která jde „z očí do očí“. Pro každý bod posloupnosti výšek zkontrolujeme,



Obrázek 10: Řešení viditelnosti 2

jestli jeho výška neni větší než výška úsečky v daném místě. Pokud ano, postavy se nevidí, pokud ne, zkotrolujeme další bod. K porovnávání výšek použijeme vzorec:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} = \frac{z - z_1}{z_2 - z_1} \tag{2}$$

Zde známe x a z, to jsou souřadnice v mřížce, a hledáme y. Ilustrace je na obrázku 9. Jedinou nevýhodou tohoto postupu je, že nezohledňuje stavby ani jiné objekty v mapě, pouze samotný terén. To by šlo vyřešit zavedením do mapy informace o případné stavbě a ve výškové mapě by potom stavby vypadaly jako malé kopce. Ještě je třeba zmínit, že postavy mají v rukou pochodeň se zapáleným ohněm a viditelnost postavy je vlastně viditelnost světla, které má větší zabraný prostor. Zde by bylo vhodné mezi nimi spočítat přímky dvě, tak aby se pokryla větší plocha, jež osvětluje světlo. To je ukázáno na obrázku 10.

3.5 Výsledky testů a pozorování

Porovnání plug-inů a skriptů Nejprve porovnání z hlediska výkonu:

plug-in	0,1	ms
c# skript	0,15	ms
lua skript	1,48	ms

V tabulce jsou změřeny doby provádění plug-inu a skriptů, vždy pro jedno NPC. Složitost prováděných operací je podobná. Čísla nelze brát absolutně, protože jsou de-

gradována během aplikace v profileru. Co ale lze brát jsou poměry. Zde negativně překvapila lua. I když to pravděpodbně bude spíše vina LuaInterface, který je psaný v C# a v C++NET. Dále pokud uvážíme efektivnost použití oněch technik, pak je třeba říci, že skripty v jazyce lua jsou snadno a rychle implementovatelné a v porovnání s plug-iny si pro stejnou funkčnost vystačíme i s o něco méně řádky kódu. Při psaní plug-inů se programátor nemusí starat o to, které všechny struktury má k dispozici, jednoduše všechny, které potřebuje. Plug-in se tak může snadno stát ještě mocnějším. Ve výsledku bych vyhodnotil použití plug-inů jako vhodné pro masovější záležitosti, tedy pro větší množství NPC, kdežto skripty v jazyce lua se velmi hodí, chceme-li dodat nějakou výjimečnou funkčnost výjimečným postavám, jako například finální nepřítel (boss). Takových postav jsou řádově jednotky. U postav, kterých jsou desítky a více už by bylo výkonové zatížení větší a zde by se proto lépe uplatnily plug-iny.

Výkon knihovny V tabulce jsou zobrazeny zprůměrované časy, jak dlouho trvají jednotlivé akce v rámci jednoho provedení(snímku) a poměry zatížení počítače:
Pro procesor Mobile AMD Sempron 3100+

	doba provádění	zatížení
grafika	234,57 ms	95,68%
AI	10,64 ms	4,32%

Pro procesor Intel Core2Duo E6600

	doba provádění	zatížení
grafika	38,297 ms	98,68%
AI	0,72 ms	1,84%

Při měření bylo ve hře 32 NPC, z toho 4 byly řízeny lua skriptem, zbytek plug-inem. Absolutní čísla opět nejde brát vážně vzhledem k běhu v profileru. Celkově se ukazuje, že zatížení výpočetního systému je nízké. To je způsobeno i náročností grafického enginu, který je navíc psán pomocí Managed DirectX, které výkon dále brzdí. Dále je ve hře jen málo postav (celkem 5), které by například využívaly A* algoritmus pro delší cesty. Stráže a divoká zvířata jej nepotřebují. Například strategické hry, kde se pohybuje velké množství jednotek po velkých mapách a na velkých vzdálenostech jsou v tomto směru mnohem náročnější.

Chování NPC Testování chování bylo realizováno vizuálním pozorováním mnou a dalšíma dvěma kolegama za běhu hry. Následující text se týká hlavních nepřátel.

- Pohyb - algoritmus hledání cesty nachází dobré cesty, NPC nekličkují ani zbytečně nezacházejí. V boji se NPC snaží držet blízkou vzdálenost k protivníkovi. Zároveň uhýbají do stran a tím ztěžují zaměřování střel. Ne, že by reagovali na střely, ale jejich pohyb působí živě.

- Boj NPC proti NPC - když bojují dvě NPC proti sobě, velmi záleží na jejich vstupních atributech level, síla a obrana, které ovlivňují účinky kouzel. Silnější NPC většinou vyhrává. Svoji roli zde hraje ale i náhoda a uhýbaní střelám. Ze

začatku NPC na protivníka útočí, pokud má málo zdraví, uzdravuje se, tak aby nebylo ohrožené. Záleží ovšem i na stavu protivníka. Když ten má již velmi málo zdraví volí útok. Když má NPC málo zdraví a ještě k tomu je protivník silnější (vyšší level) zvolí obranný mód (viz kapitola Konečný stavový automat). Když má NPC velmi málo zdraví a protivník je silnější, dá se na útěk pryč od nepřítele. Když se navíc do boje připojí další NPC potom si každé NPC vybírá jako svůj cíl nejslabší z možných.

- Boj NPC proti hráč. Vzhledem k současné pomalosti vystřelených fireballů (hlavní útočné kouzlo) má hráč výhodu v tom, že při rychlých reakcí může uhýbat střelám od protivníků. Ovšem při tom je těžší mířit, obzváště, když se NPC pohybují, a navíc pokud se NPC drží blízko je uhýbání velmi těžké. Boje proto nejsou nudné a je třeba od hráče snaha. Pro NPC není důležité proti komu bojují a reagují stejně proti hráči jako proti jiným NPC. Tedy uzdravují se, případně utíkají, pokud mají málo zdraví, doplňují manu, když jí mají nedostatek a jinak útočí.

Příklad interakcí Následující příklad ukazuje souboj mezi NPC „Mikel", „Ton" a „Vala". Výpis akcí tak jak šli za sebou:

NPC	akce	cena	soupeř	
Ton	Fireball	692	Mikel	Ze začátku po sobě NPC střílí
Mikel	Fireball	678	Ton	opakovaně
Ton	Fireball	692	Mikel	
Mikel	Fireball	678	Ton	
Ton	Heal	626	Mikel	Dochází zdraví, rohodl se pro uzdravení
Mikel	Fireball	548	Ton	
Vala	Fireball	418	Ton	Do boje se přidalo 3. NPC
Mikel	Fireball	418	Ton	A opět po sobě střílí
Ton	Fireball	692	Mikel	
Vala	Fireball	418	Ton	
Mikel	Fireball	418	Ton	
Ton	Heal	573	Mikel	uzdravení
Vala	Fireball	496	Ton	
Mikel	Fireball	496	Ton	
Ton	útěk		Mikel	nemá šanci, tak se rozhodl pro útěk
Vala	Fireball	289	Ton	
Mikel	Fireball	289	Ton	
Ton	umřel		Mikel	nestihl utéct
Mikel	Fireball	525	Vala	teď se zbylé 2 NPC postaví proti sobě
Vala	Moon prayer	588	Mikel	NPC má nedostatek many - doplňí
Mikel	Moon prayer	523	Mikel	
Vala	Heal	682	Mikel	NPC má nedostatek zdraví - vyléčení
Mikel	Fireball	603	Vala	
Vala	Fireball	692	Mikel	
Mikel	Heal	591	Vala	
Vala	Fireball	514	Mikel	
Mikel	Fireball	603	Vala	
Vala	Fireball	258	Mikel	
Mikel	Heal	143	Vala	
Mikel	útěk		Vala	NPC se dá na útěk

Nastavení paramterů kouzel Pro výsledné nastavení jsem použil metodu poukusů a omylů, tedy zkoušel jsem jak se v různých konfiguracích budou NPC chovat. Výsledné nastavení:

kouzlo	efekt kouzla	cena many	potřebný čas[s]	cena energie
fireball	-395hp protivníka	18	1,8	3
fire drop	-50hp protivníka	12	1,8	2
heal	+200hp pro sebe	60	2	4
moon prayer	+190many pro sebe	0	3,5	10

U nastavování konfigurace bylo důležité, aby například kouzlo „moon prayer“ NPC nevolilo pokaždé když má méně než maximum many, ale až když jí má nedostatek, proto musí být znevýhodněno v ostatních atributech jako je čas a energie. Obdobně uzdravovací kouzlo „heal“, které stojí více many. U něj navíc je třeba, aby přidané hit body byly menší než síla útočného kouzla, aby se NPC nemohlo uzdravo-

vat do nekonečna (v kombinaci s doplňováním many). U útočných kouzel je zase důležité, aby nezabili protivníka na 2 rány, tím by se boje staly jednoduché a aby nezastínily svoji silou (výhodností) ostatní kouzla. Kouzlo „fire drop“ je obranné kouzlo, proto je tak slabé.

Nastavení paramterů NPC Tím je myšleno nastavení jejich hit bodů, levelu, many, síly a obrany. Cílem bylo pro stupňovat obtížnost pro hráče, tak aby obyčejná zvířata byla jednoduchá, stráže již těžší, hlavní protivníci rozhodně těžcí na poražení a finální boss už velmi těžkým oříškem. Jednotlivá NPC stejného typu se mezi sebou trošku liší, některá jsou silnější než ostatní, ale rozdíly jsou malé.

4 Závěr

Výsledkem této práce je prostudování několika metod umělé inteligence a především funkční a zdokumentovaná knihovna umělé inteligence vytvořená na platformě Microsoft .NET, která je zakomponovaná do herního enginu. Implementuje metody pro řízení postav jako jsou konečný stavový automat, výběr akcí, path finding a podpora skriptů. Zároveň umožňuje značnou modifikovatelnost mimo program díky skriptům, architektuře plug-inů a konfigurovatelnosti parametrů akcí a postav z konfiguračních XML souborů. V práci byly porovnány metody skriptů a plug-inů, výsledkem je pojednání o jejich výhodách a nevýhodách a vhodném uplatnění a také o možných jazycích pro skripty.
Z pohledu uživatele knihovny stačí napsat dějový skript, pomocí XML souborů nastavit, jaké postavy ve hře budou s jakými atributy a jaké akce budou mít k dispozici. Pokud by chtěl víc, může si napsat plug-in pro stavový automat případně skript, kterým může naprosto změnit chování NPC postav. Nebo může použít nastavení tak jak je připravené. Jako nedostatek bych viděl absenci umělé inteligence pro nebojová NPC, tedy například vesničané. Zde je opravdu velký prostor pro zlepšení ve směru interakce s hráčem, třeba pomocí dialogů. Epizodická paměť by byla velmi vhodným doplněním a určitě by hru velmi obohatila. Další věcí, která v knihovně chybí je týmová spolupráce mezi počítačem řízenými postavami, popřípadě s hráčem. Navržený scénář s touto možností sice nepočítá, ale do budoucna by také byla vhodným vylepšením. Na druhou stranu, tyto 2 nedostatky by dost možná vydaly na další samostatnou práci.
Celkově vzato, knihovna plní svůj účel, je snadno modifikovatelná a je i dále do budoucna rozšiřitelná, což je nejdůležitější.

Přehled zkratek a použitého značení

NPC - Non playable charakter, nehratelná postava neboli počítačem ovládaná bytost. V textu často označeno pouze jako postava.
RPG - Role playing game, hra na hrdiny. Je to žánr her, kde hráč hraje za jednu postavu, hrdinu. Důležitým prvkem je zlepšování se s plněním úkolů a také sociální prvek.
Heuristika - přeneseně „umění objevovat". Použití v algoritmech umělé inteligence, kde nelze použít klasické postupy. Při jejím využití není zaručen nejlepší výsledek, ovšem většinou se dojde relativně rychle k „dobrému" výsledku.
Engine - Framework, ktereý zaobaluje nízkoúrovňovou funkcionalitu pro práci se zdroji, grafikou, zvukem, hrou,...
XML - Extensible Markup Language. Je meziplatformní, softwarově a hardwarově nezávislý a standardizovaný nástroj pro přenos informace.
Profiler - Nástroj pro ladění výkonu aplikací, dokáže odhalit kritická místa v programu a jiné. *Virtuální stroj - software, který vytváří virtualizované prostředí, ve kterém koncový uživatel může provozovat software na abstraktním stroji.*

Reference

[GPG404] KIRMSE, Andrew. Game programming Gems 4
Charles River Media, ©2004. ISBN: 1-58450-295-9

[GPG302] TREGLIA, Dante. Game programming Gems 3
Charles River Media, ©2002. ISBN: 1-58450-223-9

[GPG100] DELOURA, Mark. Game programming Gems 1
Charles River Media, ©2000. ISBN: 1-58450-049-2

[GAI05] BUCKLAND, Mat. **Programming Game AI by Example**
Wordware Publishing, ©2005. ISBN: 1-55622-078-2

[WikiScript] Wikipedia: Scripting
$http://en.wikipedia.org/wiki/Scripts_(artificial_intelligence)$*[22.6.2008]*

[WikiBres] Wikipedia: Bresenham's line algorithm
$http://en.wikipedia.org/wiki/Bresenham's_line_algorithm$*[22.6.2008]*

[Amit] Amit: Amits A* pages
$http://theory.stanford.edu/\ amitp/GameProgramming/$*[22.6.2008]*

[XNA] Microsoft Corporation: Xna Creators Club
$http://creators.xna.com/$

[Luabind] Rasterbar software: Rasterbar products
$http://www.rasterbar.com/products/luabind.html$*[22.6.2008]*

[LuaInterface] LuaForge: LuaInterface
$http://luaforge.net/projects/luainterface/$*[22.6.2008]*

[LuaTutor] Lua users: Lua users tutorial
$http://lua-users.org/wiki/TutorialDirectory$*[22.6.2008]*

[Lua] Departamento de Informática, PUC-Rio: The Programming Language Lua
$http://www.lua.org/$*[22.6.2008]*

[Pyth] Python Software Foundation: Python Programming Language
$http://www.python.org/$*[22.6.2008]*

[cs] Microsoft: Visual C# Developer Center
$http://msdn.microsoft.com/en-us/vcsharp/default.aspx$*[22.6.2008]*