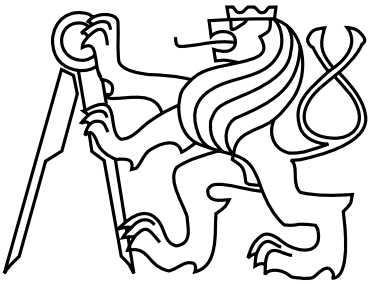
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Framework pro tvorbu RPG her

Bc. Jan Beneš

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

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující magisterský

Obor: Výpočetní technika

3. ledna 2010

Poděkování

Poděkování patří vedoucímu práce, Ing. Petru Felkelovi, za průběžné konzultace a cenné rady, rodičům za podporu v hektických dnech psaní práce a také Janu Abrahamčíkovi a Janu Hinkovi, kteří se účastnili drobných projeků, které práci předcházely a poskytly cenné zkušenosti.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 4. 1. 2010 .....

Abstract

The thesis describes design and implementation of a specialized game creation system created for making computer role-playing games. The main goal is to design an open system capable of creating a wide range of diversified games.
The text describes overall design of the system, including specific format of configuration and data files and continues with description of design and implementation of two main parts.
The first one is a game engine, which encapsulates functionality for managing events, behaviour of monsters, user interface and - most of all - a RPG system, created as a module with extensive configuration options of chararcter development.
Second part is formed by a set of tools for the game authors.

Abstrakt

Práce se zabývá návrhem a realizací specifického frameworku/GCS (Game Creation System), zaměřeného na žánr RPG her. Důraz je kladen na otevřenost a možnost vytvářet široké spektrum rozdílných her.
V textu je popsán návrh systému jako celku, včetně formátu používaných konfiguračních a datových souborů, a dále návrh a implementace jeho dvou hlavních částí.
První je herní engine, zahrnující funkcionalitu zpracovávající události v herním světě, chování nepřátel, grafické rozhraní a především RPG systém, realizovaný jako modul s rozsáhlými možnostmi nastavení pravidel pro vývoj postav.
Druhou částí je uživatelské rozhraní pro autory her.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Podobně stará jako počítačové hry samy je i chuť hráčů zapojit se kreativně do vytváření jejich podoby. Proto je mnoho her distribuováno i s editory pro úpravy a doplňování rozličných částí obsahu. Mnoho uživatelů však by však rádo šlo ještě o krok dál a vytvořilo si celou hru podle vlastního nápadu a s vlastními pravidly. Kromě - poněkud náročné - možnosti naučit se některý z programovacích jazyků je dnes možná i cesta přes některý ze specializovaných GCS[1] programů jako Game Maker[2] nebo Game Editor[1], které poskytují intuitivní uživatelské rozhraní pro tvorbu her s 2D grafikou.

Pro žánr RPG[2], který je velmi specifickým typem her, vyznačujícím se především velmi rozsáhlými a komplikovanými pravidly, se však jejich použití stává zbytečně složitým, protože vyžaduje implementovat veškerou vnitřní logiku, často s poněkud omezenými prostředky. Proto vznikají i úzce specializované projekty, které autorům poskytují hotovou kostru hry, grafické nástroje na tvorbu obsahu i výchozí sadu dat.

1.1 Motivace

Pokud by se hypotetický autor rozhodl realizovat svůj nápad na RPG hru pomocí některého z těchto projektů a vybíral systém na základě následujících kritérií:

- software, který je zdarma a pokud možno open-source
- není po technické stránce zcela zastaralý
- všechny důležité aspekty herního systému/pravidel jdou změnit
- dostupný hráčům na všech hlavních platformách (Windows, Linux, Mac OS X)
- umožňuje vytvářet hry v češtině,

pak by zjistil, že žádný z programů, které jsou k dispozici, ani zdaleka nevyhovuje.

Na trhu je rozšířená velice oblíbená komerční aplikace RPG Maker[8] z Japonska, etalon v oboru, která ovšem - kromě toho, že není zdarma - staví na pevně daných pravidlech s

[1] Game Creation System
[2] Role-playing games

možností ovlivňovat pouze hodnoty některých parametrů hry. Dále narazíme na řadu systémů na RPG Maker úzce navázaných, především přes využívání stejného formátu dat a mnoho systémů v různé fázi vývoje, většinou po technické stránce velmi omezených (překvapivý počet různých projektů disponuje grafikou v rozlišení 320x240).

Cílem této práce je tedy vytvořit čistě navržený systém splňující daná kritéria a s ambicí poskytnout po stránce vnitřní logiky a pravidel systém srovnatelný spíše s komerčně úspěšnými RPG hrami, než se zmiňovanými projekty (především RPG Maker ale bude ještě podrobně rozebrán v kapitole 2).

Kapitola 2

Popis problému, specifikace cíle

2.1 RPG hry - úvod, definice pojmů

Počítačové RPG hry jsou přímými potomky společenských her na hrdiny (například Dungeons & Dragons), přičemž zachovávají mnohé principy a často i matematický aparát - v některých případech například simulované házení kostkou pro určení úspěchu či neúspěchu nějaké činnosti. V typické RPG hře hráč prochází s postavou nebo skupinu postav (Avatarů) rozsáhlým herním světem, až na vyjímky ve fantasy nebo sci-fi prostředí, plní úkoly zadané mu jeho obyvateli (NPC[1]) a bojuje s nepřáteli. Postavy jsou odměňovány zkušenostmi a lepším vybavením, postupně se zdokonalují ve svých dovednostech.

2.1.1 Vývoj postavy

Právě vývoj a postupné zdokonalování postav je stěžejním prvkem, odlišujícím RPG hry od ostatních žánrů. Proto si můžeme definovat pojem **RPG systém** jako sadu hodnot a pravidel popisující vlastnosti a schopnosti postavy a možnosti jejich vývoje, zahrnuje také vzorce pro výpočet míry úspěšnosti prováděných akcí. Vždy jde o několik navzájem propojených částí, nejdůležitějšími jsou:

Vlastnosti - množina číselných hodnot popisující základní fyzické i psychické vlastnosti postavy ("síla", "intelekt", ...) a její šance v soubojích. Některé hodnoty mohou být funkcí jiných hodnot. Každá vlastnost má obvykle neměnnou výchozí hodnotu a hodnotu aktuální, která se naopak v závislosti na událostech ve hře mění velmi často.

Dovednosti v rámci RPG systému popisují, do jaké míry postava ovládá jednotlivé činnosti implementované v rámci vnitřní logiky hry (zde se mezi sebou jednotlivé herní tituly výrazně liší, obvykle však jde o ovládání různých zbraní a druhů vybavení, magii a zvláštní schopnosti a také běžné činnosti jako běh, smlouvání v obchodech nebo opravy poškozených předmětů). Jak je postava zběhlá v dané dovednosti je opět dáno číselnou hodnotou, která se může zlepšovat například aktivním používáním dané dovednosti. Dovednosti mohou také měnit hodnoty vlastností či být provázané s jinými dovednostmi.

[1] Non-player Character

Zkušenosti a úroveň - celkové zkušenosti postavy jsou obvykle vyjádřeny jednou hodnotou, ke které se přičítají bonusy za vítězství v boji nebo úspěšné splnění úkolu. Funkcí zkušeností je úroveň postavy, hodnota obvykle v řádu maximálně desítek. Dosažení vyšší úrovně znamená celkové posílení postavy, ať už automaticky nebo pomocí několika bodů, které hráči rozdělí mezi jednotlivé vlastnosti a/nebo dovednosti.

V počáteční fázi hry je obvykle postup po úrovních relativně rychlý, s postupem času se zpomaluje.

Předměty - systém předmětů je tvořen jednak databází jednotlivých druhů vybavení, jednak pravidly, podle kterých předměty ovlivňují hodnoty vlastností.

2.1.2 Západní a asijský styl RPG her

V rámci práce má význam uvést i odlišnosti mezi RPG hrami vzniklými v Americe nebo Evropě a asijskými hrami, především z Japonska. Systém RPG Maker, který budu často používat jako referenci, je typickým představitelem asijského přístupu, zatímco systém vytvářený v rámci práce spadá mezi RPG hry západního druhu.

Asijské RPG hry jsou většinou zaměřené na příběh postav ovládaných hráčem, postavy jsou proto na začátku hry dány scénářem, stejně tak většina jejich rozhodnutí v průběhu hry. Velkou část herního času tvoří dialogy.

Při cestování herním světem jsou postavy čas od času náhodně konfrontovány se skupinou nepřátel. Souboj obvykle probíhá na kola, ve vyhrazeném prostředí.

Typickou platformou asijských RPG her jsou herní konzole.

Západní RPG hry obvykle dávají hráči možnost vybrat si postavy a ovlivnit jejich vlastnosti. Stejně tak rozhodnutí, ovlivňující další průběh hry, jsou plně v rukou hráče.

Souboje s nepřáteli probíhají v reálném čase, přímo v lokaci, kterou postavy procházejí.

Typickou platformou západních RPG her jsou osobní počítače.

2.2 Typický uživatel

Pro další použití v textu si definujme dvě uživatelské role:

Autor hry je uživatel, který pomocí systému vytváří obsah hry. Počítá se u něj s důkladnou znalostí žánru RPG, je možné předpokládat zkušenosti s vytvářením obsahu do různých her, pravděpodobně (ale ne nezbytně) znalost nějakého značkovacího nebo skriptovacího jazyka. Může jít o jednu osobu nebo malou skupinu lidí, tvorbou her se zabývá ve svém volném čase.

Hráč je uživatel, který využívá engine[1] systému pro hraní některé z vytvořených her. Do této role se bude pravidelně stavět také každý autor. Hráč nemusí mít žádné povědomí o struktuře projektů a způsobu jejich vytváření.

[1] Programové jádro hry, obvykle zpracovává grafiku, umělou inteligenci, vnitřní logiku.

2.3 Struktura systému

Návrh a implementace projektu se bude týkat tří hlavních částí:

- **Struktura a formát dat** - prvním krokem je navrhnout, jak budou popsány jednotlivé části her, ať už z pohledu datových struktur popisujících herní svět a pravidla nebo formátu datových souborů. Podle zadání je hlavním kritériem otevřenost, jednak aby si autor mohl detailně nastudovat a pochopit, jak systém funguje uvnitř, jednak pro usnadnění vytváření externích nástrojů.

  Důležitá je také perspektiva autorů. Účelem celého systému je autorům usnadnit práci v dlouhém a náročném procesu zahrnujícím tvorbu pravidel, světa i dat, proto by datové struktury i formát souborů měly dobře popsané, zohledňující možnost různých implementací jednotlivých modulů a především jednoduché. Stejné podmínky platí i pro formát dat - jeden z cílů je podpořit kreativitu autorů a tvorbu originálních dat, ne pouze sestavovat prostředí z množiny dat poskytnutých ve standardní instalaci systému. Důležité bude například rozhodování o formátu dat a možnosti animací v rámci grafického zobrazení. Použít 3D grafiku? Modely nebo pouze billboarding?

  Dalším typickým příkladem problému, který bude nutné vyřešit, je struktura dat prostředí herního světa - bude celý herní svět jednou datovou strukturou nebo bude rozdělen na části ("Mapy")? Bude potřeba nějak zohlednit rozdíly mezi interiéry a exteriéry? Jak budou data uložena - v textovém nebo binárním souboru? Jak budou propojena s dalšími daty a moduly? Bude struktura herního světa dostatečně jednoduchá, aby se celé prostředí dalo vytvořit v rozumném čase?

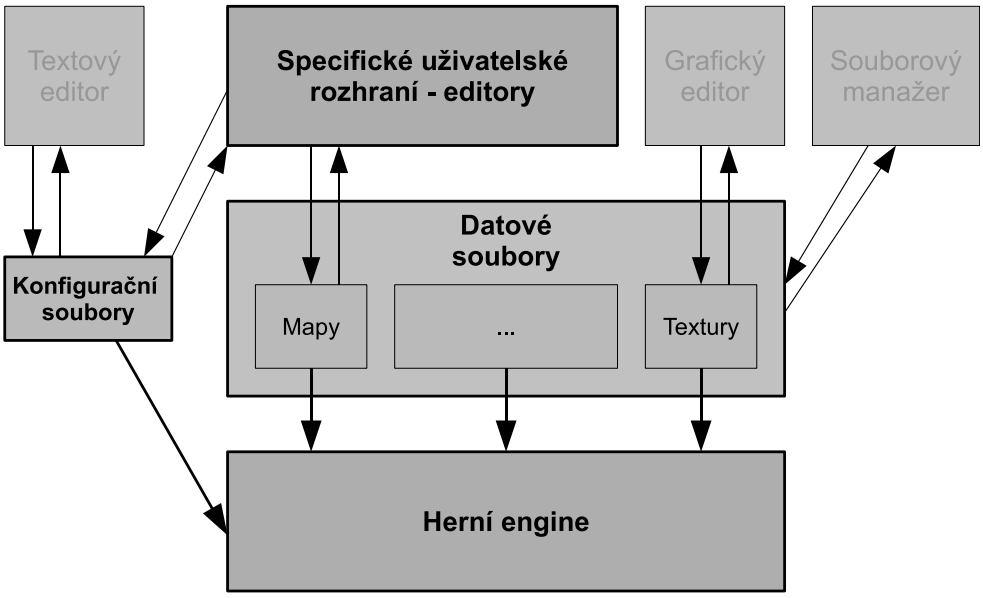
  Důležitá je také hierarchie adresářů v rámci projektu - měla by být pevně daná a logicky strukturovaná, každý herní projekt bude obsahovat mnoho souborů s vnitřními daty a nastavením a zároveň množství multimediálních, především obrazových dat. Návrh se musí zabývat i instalací dalších her a jejich správou v rámci systému.

- **Herní engine** - nejrozsáhlejší částí implementace bude vlastní jádro hry, sestávající z mnoha modulů: RPG systém a vnitřní logika pro postavy, nepřátele, předměty a dovednosti, dále zpracování událostí, grafika a uživatelské rozhraní, umělá inteligence a hledání cest, systém pro načítání dat a jistě také nástroje pro ladění a analýzu problémů. Po dokončení práce bude jádro obsahovat alespoň základní funkční verze všech modulů, vzhledem k rozsahu zadání bude největší důraz kladen na možnosti RPG systému.

  V rámci návrhu bude nutné zvolit technologii pro implementaci celého systému a pečlivě zvolit způsob řešení jednotlivých částí.

- **Uživatelské rozhraní pro autory** - v závislosti na návrhu datových struktur a formátu souborů bude možné určit, pro jaké činnosti v rámci tvorby projektů bude potřebné nebo výhodné vytvořit specifické nástroje s grafickým rozhraním a pro které bude výhodnější použít běžné aplikace jako textový a grafický editor.

  Nejlepším příkladem, kde je prakticky nutné vytvořit grafické rozhraní, je opět herní prostředí. Modelování terénu a umisťování objektů do prostředí může být velmi usnadněno poskytnutím editoru s několika nástroji pro typické úkony. Důraz bude kladen na účelnost a použitelnost.



Obrázek 2.1: Struktura projektu

Kapitola 3

Struktura práce

Práce je typickou implementační prací, čemuž bude odpovídat i obsah následujících kapitol:

V kapitole 4 se budu věnovat existujícím systémům, jednak z open-source scény, jednak podrobně rozeberu komerční aplikaci RPG Maker, která bude dobrým měřítkem pro vlastní práci, protože dobře ukazuje možnosti i omezení tohoto typu aplikací.

V kapitole 5 - "Analýza a návrh systému jako celku" - bude popsán návrh z globálního pohledu, představeno několik principů, kterými se další práce bude řídit a budou navrženy součásti používané celým systémem, především bude rozebrána volba formátu souborů pro uložení konfigurace a dat vytvářených projektů.

V kapitole 6 - "Návrh a realizace dílčích součástí a modulů" bude detailně rozebráno postupné řešení dílčích problémů v rámci jednotlivých částí systému. Jedná se především o jádro enginu, jeho jednotlivé moduly a také uživatelské rozhraní pro autory her.

Popsanými, navrženými a realizovanými moduly enginu budou především RPG systém, modul zapouzdřující herní svět/prostředí, modul zpracovávající chování postav a také grafické rozhraní.

Kapitoly 5 a 6 nejsou striktně rozdělené na témata "návrh" a "realizace"- vzhledem k rozsahu práce a také tomu, že se jedná o velké množství relativně nezávislých problémů, bude postupováno po jednotlivých částech a pro každou součást systému bude popsán návrh i realizace. Podaří se tak snad vyhnout situaci, kdy v návrhu bude popsán konkrétní modul a definovány pojmy a až o desítky stran dále v další kapitole budou použity v popisu realizace.

V sedmé kapitole, zabývající se testováním implementovaného řešení, předvedu a okomentuji možnosti implementovaného systému na dvojici co nejrozdílnějších her. Hlavní rozdíl bude v nastavení RPG systému, účelem je ukázat jednak hru co nejjednodušší, jednak hru využívající maxima možností implementovaných modulů.

V osmé kapitole bude implementovaný systém podroben detailnímu srovnání s aplikací RPG Maker.

V kapitole devět zhodnotím, zda se podařilo splnit zadání a shrnu výsledky a zkušenosti získané prací na projektu. Vzhledem k velmi rozsáhlému zadání také doplním řadu návrhů na možné rozšíření.

Výsledky pak budou ještě naposledy shrnuty v závěru.

Kapitola 4

Analýza existujících systémů

Předmětem kapitoly 4 je rozbor dostupných systémů a jejich zhodnocení z hlediska zvolených kritérií.

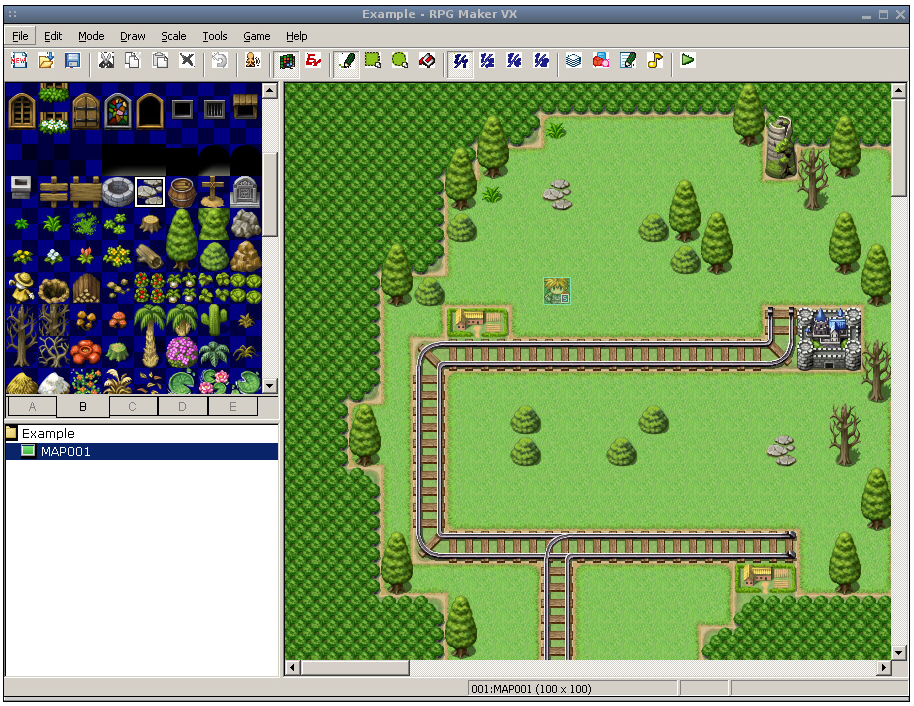
4.1 RPG Maker

Japonská série RPG Maker má poměrně dlouhou historii, na platformě PC existuje od roku 1997, kdy vyšla verze RPG Maker 95. Software je cílený především na japonské uživatele, se zpožděním jsou vydávány anglické verze. Verzí vyšlo již několik, postupně přidávají nové funkce, hlavní principy ale stále zůstávají zachovány. Jedná se o shareware, cena licence je 20$.

Podle zprávy společnosti Enterbrain, která program vytváří, se jenom do roku 2005 prodalo přes 20 milionů kopií, jde tedy o velmi úspěšný projekt. Po osobních zkušenostech s jeho používáním zkusím vyjmenovat možné důvody:

- jednoduché, i pro laiky srozumitelné a intuitivní rozhraní editoru
- množství dat součástí výchozí instalace
- pevně daný, ale pečlivě vyvážený RPG systém
- na straně enginu možnost okamžitého testování, velice rychlý běh a žádné prodlevy při načítání
- pixelartová 2D grafika v manga stylu

RPG systém pracuje podle typických "východních" pravidel, postava získává zkušenosti, díky kterým roste její úroveň, což je hodnota, od které jsou odvozeny hodnoty všech vlastností. Systém umožňuje zadat křivku/vzorec, podle kterého výpočet hodnot probíhá, seznam vlastností a jejich význam je pevně daný. Při startu hry je také pevně dáno, s jakou/jakými postavami hráč začíná, podle scénáře se družina může rozrůstat o další členy.



Obrázek 4.1: RPG Maker - editor

Prostředí je tvořeno množinou map postavených na čtvercové síti. Každá mapa je tvořena třemi vrstvami: terénem, objekty a "událostmi". Terén a objekty jsou hlavními prostředky tvorby vizuální stránky a také určují, která pole jsou pro postavy průchozí. "Události" jsou objekty, na které je připojený skript, popisující jejich chování.

Pohyb po mapě je omezen na kroky odpovídající velikosti jednoho pole sítě. Každé mapě je možné přiřadit několik druhů nepřátel a pravděpodobnost setkání s nimi. Při procházení mapou je pak hráč čas od času konfrontován se skupinou nepřátel - dějiště se přesune na obrazovku určenou pro souboj. Pro každou mapu je možné definovat bitmapu použitou jako pozadí pro souboje. Boj probíhá na kola, postupně se střídají akce nepřátel a postav.

Vnitřní logika a události - systém umožňuje do určité míry naprogramovat události ve hře. Prostředky, které má autor k dispozici, jsou sada příkazů, vnitřní proměnné a "přepínače".

Příkazy odpovídají mnoha různým akcím v rámci enginu, nejběžnější je výpis textu se zobrazením portrétu jedné z postav - tedy tvorba dialogů, dále změny hodnot vlastností postav, manipulace s předměty, pohyb po mapě nebo přesuny mezi mapami, vyvolání souboje, ale třeba také manipulace s grafikou nebo přehrávanou hudbou.

Proměnné a "přepínače" (boolovské proměnné) umožňují uchovávat stavové informace hry a přenášet je mezi různými nezávislými "událostmi".

Konstrukce zahrnuje i jednoduché větvení, vše se zadává a sestavuje v grafickém prostředí.

Data - součástí instalace je i výchozí balík dat (označovaný jako RTP - Runtime Package), obsahující množství dat ze všech kategorií:

- Tilesets - sady textur pro prostředí - čtvercové bitmapy terénu a objektů
- Charsets, Facesets - textury postav - animované sprity pro pohyb prostředím, kreslené obličeje a v některých verzích i zvláštní sada animovaných spritů jednotlivých postav a jejich pohybů v souboji
- textury pro souboje - kreslená pozadí, nepřátelé, animace útoků a kouzel
- zvuky
- hudba v MIDI formátu

Velikou přidanou hodnotou je také práce komunity uživatelů, která se na internetu sdružuje kolem četných portálů a fór, kde je shromážděné velké množství dat, v kvalitě zpracování často srovnatelných s daty připojenými k programu.

Uživatelské rozhraní editoru je navrženo jako jediná aplikace s výchozím oknem editoru map. Dalšími součástmi jsou rozsáhlé formuláře pro vyplňování hodnot do databáze postav, dovedností a nepřátel a rozhraní pro správu dat.

Uživatelské rozhraní her je minimalistické, tvořené sadou několika menu a informačních obrazovek, hlavní zobrazení je pouze pohled na hlavní postavu a okolí. Veškerý uživatelský vstup je zajištěn klávesnicí.

4.2 Open-source řešení

Mezi open-source aplikacemi bohužel nenajdeme žádný zcela dokončený systém tohoto typu, přesto alespoň dvě aplikace stojí za zmínku.

4.2.1 jClassicRPG

jClassicRPG[3] je open-source hra v pokročilém stadiu implementace, s 3D grafikou[1] a herním světem realizovaným opět nad čtvercovou sítí. Všechny důležité části vnitřní logiky a RPG systému fungují (tvorba postav, dovednosti, souboje probíhající na kola, předměty).

Hra zatím obsahuje pouze jedno "ukázkové" prostředí a nastavení pravidel a ve stabilní verzi není přiložen editor ani žádné další nástroje pro úpravy obsahu. Vzhledem k open-source přístupu se ale dá počítat, že budou brzy doplněny - pak by jClassicRPG byl významnou alternativou ke komerčním aplikacím.

Velkým pozitivem je multiplatformnost řešení, systém běží a je testován na platformách Windows, Linux i Mac OS X.

Za uvedení stojí také díky několika zajímavým funkcím a nápadům:

[1] Použit je jMonkeyEngine

Obrázek 4.2: RPG Maker - engine

- náhodná tvorba herního prostředí
- dynamicky se vyvíjející herní prostředí
- důraz na možnost nenásilného řešení zadaných úkolů

4.2.2 RPG Builder 3D

RPG Builder 3D[7] je systém s 3D grafikou, disponuje rovněž velice dobře navrženým editorem prostředí - bohužel se zatím jedná o velmi ranou vývojovou verzi, je vidět, že autoři se zaměřili především na vizuální stránku a ostatní části systému jsou zatím buďto částečně dokončené nebo zcela chybí.

4.3 Závěry

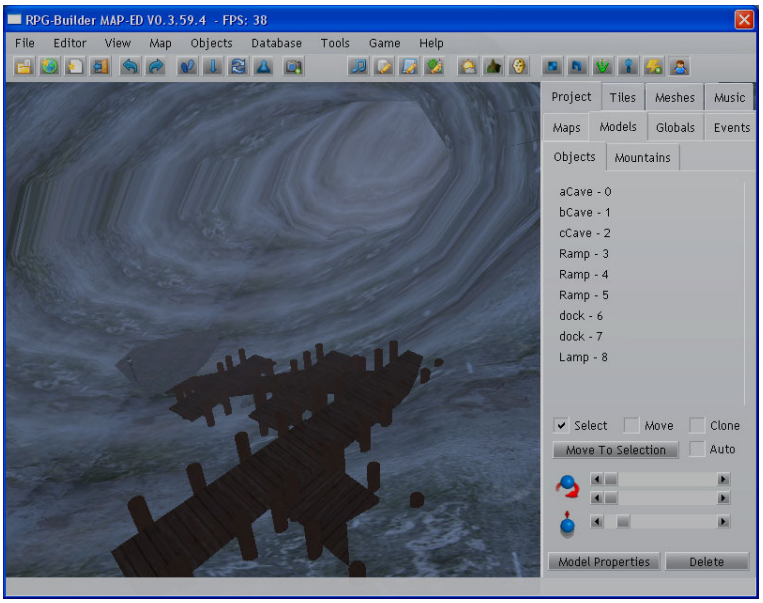
Jediným skutečně použitelným systémem pro tvorbu RPG her je komerční aplikace RPG Maker, která ale nevyhovuje některým ze zvolených kritérií.

Mezi zdarma dostupnými nebo open-source aplikacemi se nepodařilo najít žádný dokončený systém, existuje několik zajímavých projektů ve fázi vývoje, většinou ale nejsou zaměřeny přímo na autory her.

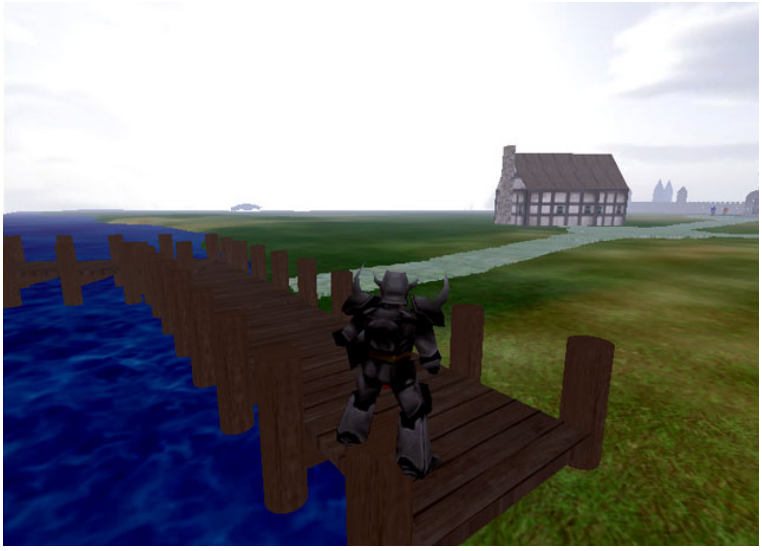
Žádná ze zkoumaných aplikací hlavně neplní stěžejní požadavek na konfigurovatelnost.

Po analýze existujících aplikací tedy docházím k závěru, že má smysl pořistoupit k samostatnému návrhu a realizaci projektu, který bude daná kritéria plnit a umožní autorům plně se realizovat.

Obrázek 4.3: jClassicRPG - engine



Obrázek 4.4: RPG Builder 3D - editor



Obrázek 4.5: RPG Builder 3D - engine

Kapitola 5

Analýza a návrh systému jako celku

V této kapitole se zaměřím nejprve na analýzu a návrh hlavních rysů systému, posléze na návrh formátu konfiguračních a datových souborů, struktury herního enginu i uživatelského rozhraní pro autory.

Prvním klíčovým rozhodnutím (učiněným již v rámci zadání) byla volba prostředků použitých k jeho realizaci, především programovacího jazyka.

5.1 Volba technologie

Jako hlavní prostředek byl zvolen jazyk Java, především díky snadné přenositelnosti aplikací napříč operačními systémy. Vzhledem k tomu, že realizovaný systém za předpokladu efektivního návrhu není aplikací výrazně náročnou na výpočetní výkon, není problémem jedna z hlavních vlastností Javy, tedy běh na virtuálním stroji a s ním spojený určitý výkonový propad oproti aplikacím kompilovaným do strojového kódu.

Vnitřní knihovny Javy mohou být zcela dostatečné pro realizaci celého systému včetně 2D grafiky, pro některé moduly je možné uvažovat použití specializovaných knihoven. Příkladem mohou být pokročilejší grafické efekty a použití buďto vlastní implementace přes rozhraní OpenGL (a tedy knihovnu JOGL[5]), nebo rovnou celého grafického systému (nabízejí se například open-source enginy OGRE 3D[6] nebo jMonkeyEngine[4]). Je však třeba si uvědomit, že instalace těchto knihoven většinou obsahuje soubory specifické pro jednotlivé platformy a komplikuje tak vytvoření univerzálního instalačního balíku.

5.2 Obecné principy návrhu

Při návrhu systému, datových struktur a formátu datových souborů se budu řídit několika principy, které vyházejí jednak z rešerše existujících projektů, jednak z osobní zkušenosti při jejich používání i nezávislém vývoji her.

5.2.1 K.I.S. - Keep it simple!

Jednoduchost je zcela klíčovým parametrem, je nutné brát v úvahu, že systém musí umožnit vytvořit hru i jediné osobě (i když asi nepůjde o rozsáhlé dílo). Největší podíl času

autoři většinou věnují modelování herního prostředí, rozmisťování objektů a celkovému vytváření vizuální stránky. Značným ulehčením je použití pevně daného/sjednoceného formátu dat pro jednotlivá prostředí a použití pravidelné struktury pro terén a umisťování objektů (v počítačových hrách nejběžnější je čtvercová mřížka, ve specifických případech se používá i šestiúhelníková). Především operační paměť pak šetří rozdělení herního světa na menší oblasti, mapy.

V rámci žánru her s velice komplikovanými pravidly není pochopitelně snadné vždy plnit požadavek na jednoduchost, přesto bude důsledně dodržován především v následující podobě: vždy poskytnout nejjednodušší možnost a volbu využít komplexnější přístup nechat na autorovi. Toto se bude týkat i dat - například pokud bude v grafickém subsystému realizován systém animací pro postavy a nepřátele, bude stále umožněno používat i statické obrázky/sprity (každý autor bude chtít mít ve své hře velké množství různých druhů nepřátel, mnoho autorů si rádo vytvoří vlastní originální druhy - málokterý autor ale bude ochotný a schopný každý z nich animovat).

Také z pohledu implementace (opět i vzhledem k rozsahu projektu) je nutné volit co nejefektivnější řešení jednotlivých problémů. Tedy, pokud je možné dosáhnout uspokojivého funkčního výsledku s využitím jednoduchých prostředků ze strany implementace (a později také autora) nebo výborného výsledku s použitím komplexních algoritmů a datových struktur, bude použit jednodušší princip.

5.2.2 Podpora tvorby originálních dat

S předchozím bodem souvisí i potřeba snadného způsobu přidávání datových souborů, například pouhým nakopírováním do určeného adresáře a pojmenováním podle určených konvencí, případně ještě zapojením do systému přidáním záznamu v databázi. Graficky či muzikálně nadaným autorům pak nebude nic bránit doplnit výchozí data o vlastní textury terénu, sprity postav nebo hudbu.

5.2.3 Úpravy a příklady jako výchozí bod

Úpravy existujícího funkčního celku a využití příkladů bývají efektivnější a především rychlejší cestou, jak pochopit principy jeho fungování, než (byť při nastudované dokumentaci) začínat tvořit "od nuly". Proto je výhodnější, aby proces tvorby hry začal z již sestavené základní sady pravidel a dat, na kterých bude možné provádět veškeré úpravy. Mimo jiné je tento princip použit i u RPG Makeru.

I při tvorbě uživatelské dokumentace bude na příklady kladen důraz. Dokumentace však bude součástí práce pouze částečně - především proto, že s ohledem na budoucí reálné použití bude hlavním jazykem angličtina. V současné podobě budou hlavním zdrojem informací především rozsáhlé vysvětlující komentáře v konfiguračních souborech.

5.2.4 Otevřenost

Otevřený textový formát souborů pro uchování nastavení a dat umožní úpravy s použitím běžného textového editoru, usnadní tvorbu specifických externích nástrojů a autorům umožní navzájem svá díla studovat a případně se inspirovat.

V případě chybějící funkcionality v grafických nástrojích nebo v případě chyb v jejich implementaci také budou mít autoři stále možnost realizovat své záměry pomocí přímé editace datových souborů.

Formát by měl být pochopitelně jednotný pro co nejširší množinu popisovaných částí systému.

Rovnou zavrhuji použití některé SQL databáze, kromě otevřenosti také kvůli nutnosti instalace dalších aplikací nebo knihoven.

5.3 Poznámky

V návrhu a implementaci bude jako výchozí jazyk použita výhradně angličtina - týkat se to bude jmen tříd a proměnných ve zdrojových kódech i klíčových slov v konfiguračních souborech. V textu práce budou při rozboru implementace obvykle uváděny anglické názvy společně s českým překladem.

5.4 Konfigurační a datové soubory

Nejprve si soubory rozdělíme do dvou kategorií:

Konfigurační soubory popisují hru jako celek, v jejich rámci je definována prázdná kostra pravidel a nastavení.

Datové soubory do kostry doplňují rozličná data:

- v binární podobě, týká se především grafiky (textur) a některých částí herního prostředí
- v textovém formátu, zde se de facto jedná o textovou databázi

Formát textových datových i konfiguračních souborů bude shodný, uvidíme i příklad, kdy jeden soubor popisující konkrétní část systému bude obsahovat jednak konfigurační, jednak datovou část.

5.4.1 Data tvořící projekt

Z pohledu projektu je data možné rozdělit na několik částečně se prolínajících částí:

- Hlavní nastavení - výběr modulů, globální nastavení - *konfigurační soubory*
- "Pravidla", tedy především nastavení RPG systému (**výčet vlastností** postav, nepřátel, popis účinků dovedností, vlastností předmětů) a odpovídajících částí uživatelského rozhraní - *konfigurační soubory*
- Databáze s konkrétními **hodnotami vlastností** jednotlivých druhů a typů nepřátel, předmětů - *datové soubory*

- Prostředí - jednotlivé mapy, objekty v nich umístěné, pravidla interaktivity a navázání na další části systému - *binární i textové datové soubory*

  (terén o velikosti např. 512 x 512 polí nemá smysl ukládat do textového souboru, zde je na úpravy nutný grafický editor - oproti tomu interaktivní objekty již mohou být uloženy v textovém formátu. I v případě binárních souborů bude ale známa a popsána jejich vnitřní struktura)

- Grafická data

5.4.2 Data-based design

Při implementaci bude návrh struktury konkrétního datového nebo konfiguračního souboru prvním krokem, třídy v jazyce Java jí budou přesně odpovídat.

5.4.3 Formát souborů

Při návrhu formátu souborů vycházím z výše vyjmenovaných principů a uvažuji ještě několik dalších kritérií:

- **otevřenost** - hlavní požadavek
- **čitelnost a přehlednost** - příliš komplikovaný formát by mohl uživatele-laiky odradit, bude třeba se zamyslet například nad počtem používaných speciálních znaků
- **intuitivnost** - v ideálním případě bude na první pohled jasné, jak struktura funguje a jak přidávat nebo upravovat data
- **úspornost** - především pro dlouhé seznamy jednotlivých položek stejné struktury by měl být formát co nejúspornější

Nejdříve budeme uvažovat standardizované formáty - pokud by se tím ušetřila práce a formát by byl vhodný i pro autory-laiky, má smysl ho využít. Například jazyk XML se jeví jako zajímavá možnost, především díky hotové implementaci rozhraní pro práci s tímto formátem v runtime knihovnách jazyka Java.

5.4.3.1 Analýza jazyka XML

Jazyk XML (Extensible Markup Language, [9]) je univerzální značkovací jazyk s otevřenou specifikací, dnes používaný především v prostředí internetu a také kupříkladu v kancelářských balících. Pomocí schémat, například v jazyce XSD, je možné definovat strukturu a klíčová slova (značky a atributy) pro jednotlivé soubory. Velmi elegantní řešení by pak bylo ze schématu rovnou vygenerovat i zdrojový kód v Javě a naprogramovat systém, který ze vstupního XML souboru automaticky vytvoří instance objektů.

Proti tomuto řešení však stojí několik argumentů: systém jako takový pravděpodobně nebude fungovat v některých specifických situacích, které by musely být výslovně ošetřeny - například odkazy mezi vzdálenými uzly hierarchie - struktura XML dokumentu odpovídá stromu, vzájemné reference mezi položkami v datových souborech budou oproti tomu často

tvořit cyklický graf. Automatická tvorba instancí by se tak musela rozšířit o zpracování odkazů, pravděpodobně s použitím ID. Jedná o tedy velmi rozsáhlý problém a možná zbytečně komplikované řešení.

Nejprve si ale vyhodnoťme samotný jazyk XML z pohledu zvažovaných kritérií:

- **otevřenost** - splněna
- **čitelnost a přehlednost** - opakované používání speciálních znaků "<", ">" a "/" a neustálé uvádění jmen atributů v přiřazeních mohou činit XML dokumenty (především při absenci odsazování) velmi nepřehlednými - v praxi je téměř nutností důsledně formátovat a používat editor s barevným zvýrazňováním syntaxe
- **intuitivnost** - XML je jazyk, který se uživatel musí vědomě naučit, pokud již nemá zkušenosti s nějakým jiným jazykem podobného typu
- **úspornost** - párové značky mohou až zdvojnásobit rozsah dokumentu, stejně tak opakovaně uváděná jména atributů přidávají množství zbytečných znaků - například pro několik záznamů stejné struktury (představme si jako řádky tabulky v databázi):

  ```
  <Struktura ID="0"~Jmeno="JmenoA"~Hodnota1="1"~Hodnota2="2"~/>
  <Struktura ID="1"~Jmeno="JmenoB"~Hodnota1="10"~Hodnota2="20"~/>
  ```

  Neúspornost vynikne při srovnání XML s holými daty:

  ```
  0 JmenoA 1  2
  1 JmenoB 10 20
  ```

Z daných kritérií tedy XML bez výhrad splňuje jen jedno - otevřenost. Dalším krokem bude tedy zkusit navrhnout formát, který podmínky splní lépe a zároveň nebude jeho zpracování po implementační stránce zbytečně náročné.

5.4.3.2 Návrh vlastního formátu

Dané již je, že použity budou textové soubory s pevně daným formátem zápisu hodnot. Struktura se v jednotlivých souborech může lišit (pro přehlednost, v závislosti na komplexnosti dat apod.)

Hodnoty v souborech odpovídají instancím struktur/objektů, často provázaným mezi sebou – navázání referencí se bude řešit při načítání enginem, proto je v těchto případech třeba záznamy identifikovat. Každému záznamu bude přiřazeno jednoznačné ID a někdy - pro usnadnění orientace autora - také zkratka. Indexování bude začínat od nuly[1].

Potřebné hodnoty/typy budou přinejmenším:

- deklarace ID struktury:

[1] Zde se jedná o určitý kompromis, indexování od 1 je jistě přirozenější, obvykle je však spolehlivým zdrojem chyb, je-li navázáno na programovací jazyk, který indexuje od 0.

```
Struktura 0
```

- celá čísla použitá jednak jako hodnoty, jednak jako deklarace počtu záznamů - této hodnoty se využívá v parserech, umožňuje realizovat načítání pomocí for cyklu

```
Hodnota=1
PocetStruktur=2
```

- pole celých čísel

```
DesetHodnot:10   1 2 3 4 5 6 7 8 9 10
```

- řetězce

```
Retezec="Slovo slovo slovo"
```

- boolovské proměnné

```
BoolovskaHodnota="false"
```

Komentáře

Jako komentáře jsou vyhodnoceny řádky začínající znakem "#". Jejich hlavním použitím bude vkládání vysvětlujících poznámek a příkladů do výchozích souborů nebo, jak bude vidět u následujících příkladů, jako vodítko pro vyplňování hodnot do struktur zapsaných v řádku.

Formát zápisu

Na příkladech uvádím dva možné způsoby zápisu stejné struktury:

- hodnoty jako jednotlivá přiřazení

```
PocetStruktur=2

Struktura 0
Hodnota1=10
Hodnota2="text"

Struktura 1
Hodnota1=20
Hodnota2="text"
```

- úspornější výčet v řádku - hodí se pro dlouhé seznamy instancí stejného typu

```
PocetStruktur=2

#Struktura ID Hodnota1 Hodnota2
Struktura   0       10   "text"
Struktura   1       20   "text"
```

- pro srovnání ještě obdobná struktura v XML:

```
<Struktury>
    <Struktura
        ID="0"
        Hodnota1="10"
        Hodnota2="text"
    />
    <Struktura
        ID="1"
        Hodnota1="20"
        Hodnota2="text"
    />
</Struktury>
```

5.4.3.3 Zpětné vyhodnocení

Plní navržený formát kritéria lépe, než XML?

- **otevřenost** - splněna
- **čitelnost a přehlednost** - formát je oproti XML výrazně jednodušší, jediným používaným speciálním znakem (kromě -"a uvozovek) je "#", který je ovšem součástí komentářů. Drobným problémem může být povinná deklarace počtu záznamů, spíše ale v případě, že autor zapomene hodnotu změnit po úpravách, než kvůli samotnému jejímu významu a použití.
- **intuitivnost** - toto kritérium se hodnotí obtížně, při srovnání s XML se především hodnoty v řádku jeví jako srozumitelnější
- **úspornost** - zde je výhoda oproti XML nesporná

V přímém srovnání s XML vychází navrhovaný formát lépe, i když je možné předpokládat některé drobné problémy. Přesto se ale jedná o rozumný kompromis mezi splněním zvolených kritérií a náročností implementace načítání. Výhodou je naopak možnost (do)definovat specifické konstrukce a typy hodnot přesně podle potřeby jednotlivých částí systému.

5.4.3.4 Implementace načítání souborů

Načítání souborů řeší jednotlivé moduly samostatně, implementace odpovídá formátu konkrétních souborů. Některé moduly načítají pouze jeden soubor, například u RPG systému se jedná o souborů několik, odpovídajících jednotlivým částem funkcionality. V rámci těchto konkrétních částí programu jsou také definována klíčová slova používaná v souborech.

Pro zefektivnění procesu byla vytvořena pomocná třída s implementací rutinních funkcí:

Třída Parser

Hlavním účelem je zefektivnit psaní parserů pro další konfigurační soubory a zajistit, aby jejich struktura byla jednotná. Pracuje s obvyklými třídami Javy pro práci se soubory a řetězci, jmenovitě BufferedReader a StringTokenizer.

Zahrnuje funkce pro:

- management načítání, zpracování chyb, důležitá je především funkce pro načtení dalšího řádku s daty (tedy s přeskočením komentářů a prázdných řádků)

  ```
  getLine(BufferedReader in)
  ```

- funkce pro načtení ID struktury

  ```
  getIdFromHeader(String propertyName, String line)
  ```

- funkce pro načítání jednotlivých hodnot

  ```
  getSingleValue(String propertyName, String line)
  getSingleStringValue(String propertyName, String line)
  getSingleBoolean(String propertyName, String line)
  ```

- funkce pro načítání hodnot z řádku

  ```
  getNextIntValue(StringTokenizer tokenizer)
  getNextStringValue(StringTokenizer tokenizer)
  getNextBooleanValue(StringTokenizer tokenizer)
  ```

- funkce pro načtení pole

  ```
  getIntArray(String propertyName, String line)
  ```

- specifické funkce - například načtení rozsahu hodnot ve formátu (A-B)+(C-D)

  ```
  getExtent(String propertyName, String line)
  ```

Shrnutí

Po rozboru možností ukládání dat byl implementován systém načítající data z textových souborů s pevně daným formátem, zvoleným s ohledem na čitelnost a intuitivitu i náročnost realizace na straně programu.

S definovaným formátem souborů už můžeme přistoupit k návrhu dat, která budou obsahovat a k realizaci jednotlivých částí systému.

5.5 Engine

5.5.1 Úvod

Herní engine je nejrozsáhlejší část systému, je možné jej rozdělit na několik modulů:

Nejdůležitějším prvkem bude jádro, zajišťující časování jednak pro hlavní programovou smyčku, která v jeho rámci bude volána, jednak pro grafické zobrazení. Dále zde budou implementovány základy vnitřní logiky, jako zpracování událostí a uživatelského vstupu.

Dále bude engine uchovávat v paměti herní prostředí v podobě množiny map, udržovat jejich stav a bude zajišťovat jejich načítání, stejně tak jako včasné načtení odpovídajících grafických dat.

Po mapě se bude pohybovat množství postav - avatar a nepřátelé. Avatar bude ovládán přes vstup z klávesnice a myši, v případě nepřátel bude nutné implementovat pravidla jejich chování, tedy umělou inteligenci a pohyb po mapě (hledání cest v grafu). Dalším prvkem realizovaným v rámci prostředí budou dva typy interaktivních objektů - pevné objekty jako dveře či přepínače a pohyblivé, používané jako střely či kouzla.

I několikrát zmiňovaný RPG systém je součástí enginu, v rámci implementace půjde o popis vlastností a dovedností postav a konkrétní implementaci účinků dovedností a pravidel pro vyhodnocování výsledků soubojů. Na rozhraní mezi RPG systémem a vnitřní logikou stojí systém předmětů, jednak popis vlastností a účinků vybavení, jednak jejich umísťování do mapy a realizace inventáře.

Uživatelské rozhraní, předávající události do jádra enginu, je další rozsáhlou částí. Je možné jej rozdělit na dvě poloviny, první je "GameView", pohled na avatara a jeho okolí, druhá je tvořena veškerými informačními a ovládacími prvky.

5.5.2 Návrh a realizace v rámci práce

U každé části byla podle návrhu realizována konkrétní implementace, pokud některé detaily zůstaly pouze ve fázi návrhu, bude to vždy výslovně uvedeno v nadpisu nebo v textu.

Některé části jsou navzájem pevně propojené, jiné relativně samostatné. Dalším krokem je navrhnout, jak a u kterých modulů umožnit různé implementace, jak je vyžadováno v zadání.

5.5.3 Modularita

"Možnost volby různých implementací jednotlivých částí" bude realizována nezávisle pro jednotlivé moduly, autor hry jednu z dostupných implementací zvolí v konfiguračním souboru s hlavním nastavením projektu a dále se už bude pracovat jen se zvoleným modulem a jeho konfiguračními soubory.

Modulární návrh bude implementován s využitím principu dědičnosti/polymorfismu, v prostředí Javy konkrétně pomocí rozhraní tvořeného několika abstraktními třídami, používaného ke komunikaci se zbytkem systému a konkrétních, libovolně složitých realizací od těchto tříd odděděných.

Součásti systému, které budou umožňovat různé implementace budou přinejmenším:

- Grafika - "Game View" - zobrazení výřezu prostředí s postavami
- RPG systém včetně konkrétních částí uživatelského rozhraní
- umělá inteligence

5.5.4 Lokalizace

Vzhledem k možnosti neomezeně distribuovat programy přes internet má smysl zabývat se i lokalizací, návrh se bude snažit zvolit co nejjednodušší funkční variantu.

5.5.4.1 Multiplatformnost

Překvapivé množství problémů spojených s lokalizací je způsobeno použitím rozdílného kódování, znakových sad i sekvence označující konec řádku v různých operačních systémech.

Převody mezi různými kódováními nebo různé verze textových souborů pro jednotlivé operační systémy nejsou řešením, jako jediná používaná sada bylo zvoleno kódování Unicode (UTF-8), vzhledem k očekávané budoucí standardizaci i z praktického důvodu vývoje na platformě linux.

Především v operačních systémech Windows je pak třeba počítat s několika důsledky:

- Nutnost použít pokročilejší textový editor, který toto kódování ovládá. Pokročilejší textový editor by se dal doporučit všeobecně, například kvůli zvýrazňování syntaxe[1]. Existuje řada vhodných open-source nebo zdarma dostupných programů, v prostředí Windows například Notepad++.
- Nastavení IDE při vývoji - pokud by některé zdrojové kódy (v rámci dobrého návrhu pouze v průběhu testování) obsahovaly řetězce s jazykově-specifickými znaky, je nutné i vývojářské prostředí nastavit na UTF-8 (jako jsem například musel sám učinit u používaného IDE Eclipse).
- Ošetření načítání textových souborů v programu výslovným určením kódování:

```
BufferedReader in = new BufferedReader(new InputStreamReader(
                        new FileInputStream(configFile), "UTF-8"));
```

5.5.4.2 Realizace - Třída StringDatabase

Třída StringDatabase načítá a spravuje veškerý text v projektu pro zvolený jazyk. Jazykový balík pro jeden jazyk (rozsahem jeden konfigurační soubor) obsahuje dvojice – propojení klíčových slov (názvů) používaných v konfiguračních souborech nebo v enginu hry s textem zobrazovaným ve hře.

V případě kompletního přeložení bude mít každé klíčové slovo obsažené ve zdrojových kódech nebo v konfiguračních souborech (tedy názvy vlastností, povolání, dovedností, ...) svou obdobu v lokalizačním souboru.

Pokud není ekvivalent nalezen, je použit původní řetězec, je tedy možné překládat projety postupně.

Pro rychlejší vyhledávání v databázi jsou dvojice uloženy ve struktuře TreeMap<String, String>.

5.6 Uživatelské rozhraní pro autory her

Výchozím přístupem při návrhu bude důraz na konfigurační soubory a využití jednoduchých pravidel a konvencí pro umisťování a pojmenovávání datových souborů - konkrétní nástroje, které uživatelům ulehčí práci, lze vždy doplnit dodatečně.

[1] Nebylo to autorovým záměrem, ale konfigurační soubory používané v systému jsou velice pěkně zvýrazněny při nastavení zvýrazňování pro jazyk Ruby

Nástroj, který je jednoznačně potřebný, je editor herního prostředí. Také je třeba uvažovat o zapouzdřujícím uživatelském rozhraní, především z důvodu integrace dalších nástrojů.

5.6.1 Projekty v systému

Instalované projekty a jejich struktura jsou příkladem navržených konvencí.

Pracovní adresář aplikace obsahuje adresář **games**. Každý jeho podadresář odpovídá jedné hře.

Zvláštním případem je podadresář **common**, obsahující výchozí sadu dat a základní konfiguraci. K vytvoření hry tedy stačí tento adresář zkopírovat, obdobně lze nakopírováním do adresáře games "nainstalovat" další hry.

5.6.1.1 Struktura projektu

Každý adresář se hrou obsahuje tři další adresáře:

- config_files - zde budou umístěny veškeré konfigurační soubory jednotlivých částí systému
- maps - obsahuje binární datové soubory map
- textures - pro umístění adresářů obsahujících sady textur terénu, nepřátel, předmětů atd.

Správa dat projektů (především textur) a celkový návrh i jednotlivé funkce editoru map budou podrobně popsány v kapitole 6.

Kapitola 6

Návrh a realizace dílčích součástí a modulů

V následující - nejrozsáhlejší - kapitole bude popsán konkrétní návrh a realizace jednotlivých částí enginu i uživatelských nástrojů pro autory.

6.1 Jádro enginu

Jádro zajišťuje zcela základní funkcionalitu - načítání projektů, inicializaci modulů, běh hlavní nekonečné smyčky se správným časováním, zpracování událostí ve hře a událostí z uživatelského vstupu.

V rámci implementace je vhodné oddělit data od logiky, proto bude základ aplikace tvořen dvěmi třídami - GameProject, obsluhující veškerá nastavení včetně výběru a inicializace modulů a GameEngine, ve které poběží hlavní programová smyčka a bude zajišťovat volání odpovídajících funkcí grafiky, zpracování událostí a herního prostředí.

6.1.1 Třída GameProject

Třída GameProject při startu systému podle předaného parametru - vybraného adresáře s projektem - načte nastavení z hlavního konfiguračního souboru projektu a vyžádá načtení nastavení vybraných modulů (specifický je RPG systém, který bude podrobně rozebrán v odpovídající podkapitole). Dále udržuje databázi nepřátel a předmětů a poskytuje přístup ke konfiguračním souborům ostatním částem systému. Rozhraní je realizováno jeko množina statických metod.

6.1.2 Třída GameEngine

Vnitřní logika jádra je realizována ve třídě GameEngine.

Při startu enginu se podle odpovídajícího konfiguračního souboru inicializují datové struktury popisující herní svět a několik instancí dalších tříd, ve kterých je implementována konkrétní funkcionalita:

- **GlobalClock a GUIClock** - časovače pro grafiku a vnitřní logiku
- **GameInputEvents** - zpracování uživatelského vstupu z klávesnice a myši
- **EventProcessor** - zpracování událostí ve hře i událostí předaných z GameInputEvents

Dále je spuštěno grafické rozhraní a vybrán výchozí GUI modul (například rozhraní pro tvorbu postavy).

Při spuštění hry je hráčem vytvořená postava umístěna do světa, grafické rozhraní se přepne na zobrazení "GameView" a s využitím třídy GlobalClock je spuštěna hlavní smyčka programu.

Periodicky volaná hlavní smyčka má na starosti simulaci herního světa (v rámci aktivní mapy), volá funkce pro:

- **Nepřátele a objekty** - rozhodování a pohyb po mapě
- **Management datových struktur** - viz. Herní svět - mapy
- **EventProcessor** - zpracování událostí zaznamenaných od posledního volání

6.1.3 Logika a grafika - nazávislá vlákna

Krok simulace herního světa a překreslení grafiky jsou volány periodicky s pevným časovým krokem, nezávisle na sobě. Instance GUIClock a GlobalClock, které volání funkcí zajišťují, běží každá v nezávislém vlákně, které se uspává na časový interval daný inverzní hodnotou k zadané frekvenci. Jako vhodná fodnota, zaručující plynulé zobrzení, se jeví 60 snímků za vtařinu.

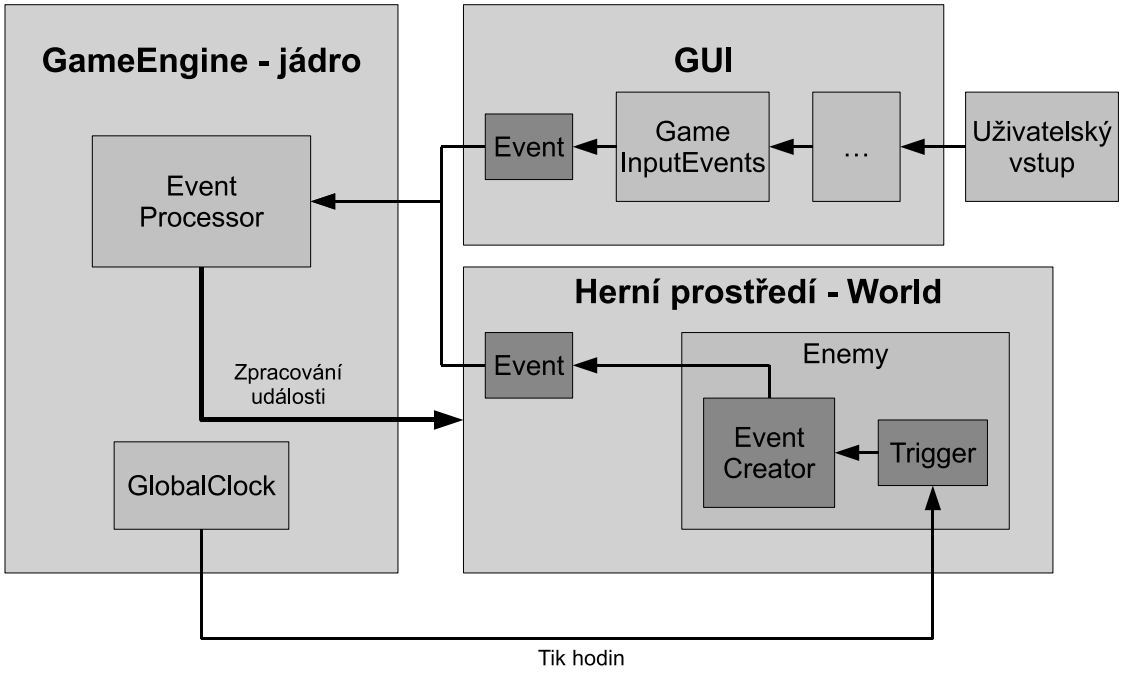
Všechny kritické sekce jsou synchronizovány pomocí konstrukce jazyka Java:

```
synchronized(Object)
```

Třída GlobalClock má ještě další funkci - "impulsy" z jejích hodin jsou nezávisle na hlavní programové smyčce předávány také instancím třídy Trigger, které mohou v určitém časovém intervalu vyvolávat různé události (akce nepřátel, změny hodnot vlastností).

6.1.4 Události - vnitřní logika

Mnoho částí systému mezi sebou potřebuje komunikovat. Realizace pomocí přímých referencí a vzájemného volání metod se od určitého rozsahu projektu stává neúnosnou, proto bude použit sjednocený systém událostí (Events), generovaných libovolnou částí programu, ale zpracovávaných systematicky jedinou třídou.



Obrázek 6.1: Příklad vytváření událostí

6.1.4.1 Events, Tiggers, EventCreator - vytváření událostí

Třída Event obsahuje univerzální strukturu popisující jednu událost, do které je možné uložit:

- typ události (jednoznačné určení, každému typu odpovídá konstanta)
- polohu na mapě
- několik dalších celočíselných hodnot pro libovolná data (ID postav a objektů apod.)

Třída EventCreator dědí od třídy Event a je určena k "nezávislému" vytváření jejích instancí. Obvyklé využití je v kombinaci s instancí třídy Trigger.

Třída Trigger slouží jako detektor různých statistik/událostí (uplynulý čas, změny hodnot vlastností, vzdálenost mezi postavami). Vždy je napojena na instanci třídy EventCreator, která publikuje detekovanou událost. Detekce je spouštěna buď periodicky přes GlobalClock nebo z konkrétního modulu.

Dalším zdrojem "závislých" událostí je uživatelský vstup, který je propagován přes několik vrstev (GameListener, GUI, GameInputEvents - bude probráno v samostatné podkapitole).

6.1.4.2 EventProcessor - zpracování událostí

Nejdůležitější součástí třídy EventProcessor je fronta, do které se řadí zveřejněné události, nepřímo přístupná přes statickou metodu:

```
public static synchronized void addEvent(Event event)
```

Samotné zpracování událostí probíhá v metodě

```
public synchronized void processQueue(),
```

která postupně zpracovává jednotlivé události (volá odpovídající metody v daných částech systému - výsledkem může být přímá operace s daty nebo vytvoření další události jako reakce). Jako příklady událostí je možné uvést pohyb avatara, použití dovednosti avatarem nebo některým z nepřátel, přidání zkušeností, dosažení vyšší úrovně, vytvoření/sebrání/položení předmětu, operace s objekty a mnoho dalších. Podrobnější informace následují v částech textu popisujících konkrétní části systému, které události vytvářejí nebo jsou jimi ovlivňovány.

V průběhu práce na implementaci se do systému událostí postupně přesouvá další a další původně nezávislá funkcionalita.

6.2 RPG systém - vývoj postavy

6.2.1 Úvod

Vzhledem k tomu, že jednotlivé části systému jsou poměrně složitě propojeny, bude třeba v některých podkapitolách pracovat s termíny, které ještě nebyly detailně probrány. I proto bude předmětem následující části RPG systém, stěžejní součást, která zasahuje do všech ostatních funkcí systému a bude zmiňovaná v dalších podkapitolách.

Jedná se také o první modul, u kterého bude umožněno přepínat implementace.

Vhodné bude opět začít připomenutím funkcionality, specifické pro žánr RPG, kterou bude tento modul bude muset poskytovat:

- popis vlastností postav
- popis dovedností postav a jejich účinků (tedy i konkrétní implementace logiky)
- popis vývoje postavy - tedy jak a za co získává zkušenosti a jak se díky nim budou zlepšovat hodnoty vlastností a účinnost dovedností, do jaké míry bude moci hráč tento vývoj ovlivňovat
- popis vlastností předmětů

Vzhledem k modularitě je prvním krokem implementace vytvořit společné a univerzální rozhraní pro ostatní součásti systému.

6.2.2 Rozhraní pro ostatní části systému

V rámci rozhraní bude bude třeba prvky společné pro všechny RPG hry systematicky rozdělit a za pomoci abstraktních tříd, od kterých posléze budou muset konkrétní implementace dědit, zpřístupnit zbytku systému.

Kromě abstraktních tříd poskytujících přístup k jednotlivým součástem funkcionality a datových struktur bude rozhraní zapouzdřeno třídou RPGSystem, spravující výběr konkrétní implementace, inicializaci a načítání konfiguračních souborů.

Pomocná třída Experience umožňuje popsat vztah mezi získanými zkušenostmi a úrovní postavy a třída GMath poskytuje veškeré funkce pro výpočty rozsahu efektů, míry úspěšnosti a udržuje informace o tom, které hodnoty pro který výpočet použít.

6.2.2.1 Balík abstract_rpg_system

Balík (Java package) abstract_rpg_system obsahuje několik abstraktních tříd vytvářejících základní kostru RPG systému a určuje tak základní strukturu konkrétních implementací. Součástmi jsou:

- **Kategorie postav** - třídy AbstractClassSet a AbstractClass slouží při tvorbě postavy a umožňují hráči vybrat si původ postavy z několika výchozích kategorií, například "rasa" nebo "povolání". Tento výběr pak ovlivní výchozí hodnoty vlastností postavy.

- **Vlastosti postav** - třída AbstractStatSet obsahuje množinu vlastností typu AbstractStat. Každá vlastnost je určena pomocí ID, zkratky a má jméno. Udržuje svou aktuální a maximální číselnou hodnotu, které je možné nastavovat, měnit a doplňovat přes definované funkce.
- **Dovednosti postav** - třída AbstractSkillSet obsahuje několik dovedností AbstractSkill. Dovednost má opět jednoznačné ID a zkratku, je pojmenovaná, každá dovednost udržuje hodnotu úrovně, na jaké ji postava ovládá. Rozhraní přidává dvě metody pro aktivaci (použití) dané dovednosti.

  AbstractSkillSet obsahuje několik dalších metod pro manipulaci s dovednostmi - propojení s vlastnostmi a předměty (v případech, kdy používání dovednosti ovlivňuje vlastnosti nebo když je používaná dovednost zvolena podle používaného předmětu) a metodu pro zpětné předávání informací po použití dovednosti.
- **Vlastnosti předmětů** - AbstractItemClass kategorizuje předměty do tříd a popisuje, jaké vlastnosti jednotlivé třídy charakterizují, AbstractItemStatSet udržuje hodnoty vlastností jedné instance předmětu, AbstractItemSpecialSet umožňuje některým předmětům přidat další vlastnosti nad rámec vytyčený předchozí třídou a AbstractItemDatabase udržuje záznamy o jednotlivých kategoriích a druzích předmětů.

Většina tříd pak obsahuje metodu createInstance(), pomocí které se umí naklonovat. Příkladem použití je například skupina nepřátel jednoho druhu, pro který je podle konfiguračních a datových souborů vytvořena první společná instance popisu vlastností. Při vložení nepřátel do mapy se každému z nich tato struktura naklonuje a dále již funguje nezávisle. Obdobný princip se dá uplatnit v mnoha dalších případech, například u vytváření předmětů.

Všechny hodnoty popisující postavy jsou celočíselné.

6.2.2.2 Třída RPGSystem

RPGSystem slouží jako správce modulu. Obsahuje globální nastavení a šablony sad vlastností a dovedností avatara a nepřátel, aby mohly být využity následujícím způsobem:

Sada vlastností avatara je obvykle složitá a v průběhu hry se neustále vyvíjí - ostatně je to právě a jen avatar, který hráče provází od začátku do konce hry.

Sada vlastností nepřátel bývá jednodušší a většina jsou jen statické parametry, popisující všeobecné silné a slabé stránky konkrétního duhu nepřátel. Obvyklá doba životnosti nepřítele bývá přeci jen pouze jeden souboj s avatarem, často jednotky vteřin.

Pro **sadu dovedností avatara** platí stejná pravidla jako pro sadu vlastností, tedy komplikovanost a postupný vývoj.

Sada dovedností nepřátel bude opět o něco jednodušší a především společná pro všechny nepřátele - každý druh nepřátel pak bude mít pouze určeno, které dovednosti ovládá a jak dobře (číselný údaj - úroveň dovednosti).

Při startu systému RPGSystem načte hlavní konfigurační soubor a zajišťuje inicializaci dalších pomocných tříd (Experience a GMath) a například také databází předmětů a nepřátel.

Podle vybraného RPG systému a na základě odpovídajících konfiguračních souborů proběhne také inicializace zmíněných sad vlastností a dovedností:

```
public static void initAvatarStatSet()
public static void initEnemyStatSet()
public static void initSkills()
```

Metody pro přístup k vytvořeným šablonám pak jsou:

- Naklonování instance sady vlastností avatara, vstupním parametrem je pole výchozích hodnot vlastností získané z GUI modulu pro tvorbu postavy

  ```
  public static AbstractStatSet createAvatarStatSet(int[] statMaxValues)
  ```

- Naklonování instance sady vlastností nepřítele, vstupním parametrem je ID jeho druhu v databázi nepřátel, pomocí kterého se získají výchozí hodnoty vlastností

  ```
  public static AbstractStatSet createEnemyStatSet(int enemyClassID)
  ```

- Naklonování instance sady dovedností avatara a její napojení na sadu vlastností

  ```
  public static AbstractSkillSet createAvatarSkillSet(AbstractStatSet statSet)
  ```

- *Přístup* ke společné sadě dovedností nepřátel

  ```
  public static AbstractSkillSet getEnemySkillSet()
  ```

6.2.2.3 Zkušenosti - třída Experience

Zkušenosti jsou užitečným motivačním nástrojem v rukách autora hry, slouží jako odměna pro hráče za vyhrané souboje a plnění úkolů (obvykle v roli kladných hrdinů) a ovlivňují vývoj postavy. Z pohledu implementace jde o jedinou číselnou hodnotu, na které podle nějaké funkce závisí jedna z vlastností postavy, označená jako **úroveň (level)** - ta již má přímý vliv na vlastnosti a dovednosti postavy, příklady:

- hodnota některé vlastnosti může být násobkem úrovně nebo na ní jiným způsobem záviset
- po dosažení vyšší úrovně jsou hráči přiděleny body, kterými může trvale zvýšit hodnoty některých vlastností; obdobně mohou být přiděleny body pro dovednosti

Třída Experience slouží především pro definici vztahu mezi hodnotami úrovně a zkušeností, pro usnadnění a urychlení udržuje tabulku s předpočítanými hodnotami zkušeností pro jednotlivé úrovně, od 1 do maximální povolené úrovně postavy. Vztah je definován pomocí co nejvšeobecnějšího vzorce:

$z = A + L \times u + Q \times u^2 + u^E$,

kde z a u jsou zkušenosti a úroveň, A, L, Q a E pak absolutní, lineární, kvadratický a exponenciální koeficient.

Všechny koeficienty jsou typu double. Hodnotám v tabulce je ještě možné určit míru zaokrouhlení, například na desítky.

6.2.2.4 Třída GMath - výpočty pro vnitřní logiku

Poslední "všeobecnou" třídou v RPG systému je GMath, zajišťující prakticky všechny výpočty v rámci systému vlastností a dovedností i jejich vlivu na vnitřní logiku ostatních částí systému, je navržena jako univerzální a nezávislá na konkrétní zvolené implementaci.

Aby systém vůbec mohl výpočty provádět, musí být správně napojen na vstupní hodnoty. K tomu slouží konfigurační soubor, ve kterém jsou jednotlivým vstupním parametrům vzorců přiřazeny vlastnosti ze sad vlastností avatara a nepřátel (všimněme si použití zkratek místo ID pro intuitivnější přístup). Jedna vlastnost může být použita ve více kontextech, nepoužívané parametry nemusí být přiřazeny.

```
StatAvatarLevel="LVL"
StatAvatarAttack="ATT"
StatAvatarDefense="DEF"
StatAvatarDamageMin=""
StatAvatarDamageMax="DMG"
StatAvatarRunningSpeed="SPD"
StatAvatarAttackSpeed="ASP"
StatAvatarLife="HP"
StatAvatarRange="RNG"

StatEnemyLevel="LVL"
StatEnemyAttack="ATT"
StatEnemyDefense="DEF"
StatEnemyDamageMin=""
StatEnemyDamageMax="DMG"
StatEnemyRunningSpeed="RNN"
StatEnemyAttackSpeed="SPD"
StatEnemySize="SZE"
StatEnemyRange="RNG"
StatEnemyLife="HP"
StatEnemyExperience="EXP"
StatEnemyMoney="MNY"
```

Důležité vzorce zahrnují výpočet úspěšnosti útoku a obrany a způsoby vzájemné závislosti hodnot dvou vlastností. V obou případech bylo implementováno několik možností, v konfiguračním souboru je určeno, které budou použity, případně doplněny parametry pro specifikaci.

```
#AttackFormula
#=============
# 1 COMBAT_FORMULA_ATTACK_DEFENSE
# 2 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE
# 3 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_MIN_MAX
# 4 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_ERENIA
AttackFormula=4
CombatCoef=2
```

Podrobněji se podíváme na poslední dva vzorce:

COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_MIN_MAX pracuje se čtyřmi hodnotami vlastností: "Útok", "Minimální zranění" a "Maximální zranění" útočníka a "Obrana" obránce. Podle dvojice hodnot vlastností ("Útok" útočníka a "Obrana" obránce) se určí pravděpodobnost úspěšného zásahu v procentech:

$P = u/o \times 50$

V případě vyrovnaných hodnot je tedy pravděpodobnost padesátiprocentní.

Pokud útočník zasáhl, obránce je zraněn a míra zranění je dána náhodnou hodnotou v intervalu <"Minimální zranění", "Maximální zranění">.

Vzorec COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_ERENIA je převzat z jednoho z předchozích projektů a na rozdíl od minulého vzorce pracuje se třemi hodnotami vlastností a jedním parametrem: "Útok" a "Maximální zranění" útočníka, "Obrana" obránce a koeficient pro nastavení rozsahu.

Nejprve je ze vstupních hodnot určen interval rozsahu zranění - pokud je Obrana > Útok:

$IRZ = (-(MZ \times (1 - (O/U))) \times K, MZ)$

Graficky:

```
        |-------------------------------------------------|----------------|
-(MZ x (1 - (O / U)) x K)                                 0               MZ
```

Naopak, je-li útočník silnější (Útok > Obrana), pak bude interval určen:

$IRZ = (0, MZ \times U/O \times K)$

Graficky:

```
        |-------------------|---------------------------------------------|
        0                   MZ                                      MZ x (U / O) x K
```

V intervalu je pak náhodně zvolena výsledná hodnota. Pokud přesahuje rozsah intervalu <0, MZ>, je změněna na jeho dolní či horní hranici.

Ve výsledku tedy bude silnější útočník častěji zasazovat maximální zranění, zatímco útočník slabší než obránce bude často útočit zcela neúspěšně. Pomocí koeficientu je možné vzorec přizpůsobit pro různé rozsahy hodnot vlastností.

Vzájemné závislosti vlastností - tedy situace, kdy jedna vlastnost získává "bonus" od další vlastnosti, používaná například jako pomocný vzorec v konkrétní implementaci RPG systému (ve třídě ComplexStat a ComplexTerm) - jsou posány dalším zvoleným vzorcem, který může popsat různé závilosti od lineární (tedy velmi silné) po jemnější možnosti, jako je závislost přes odmocninu nebo závislost logaritmická.

```
#BonusFormula
#============
#BONUS_LINEAR=1
#BONUS_SQRT=2
#BONUS_LN=3
BonusFormula=1
```

Další funkcí třídy GMath je převod celočíselných hodnot vlastností postav na parametry používané v jiných částech enginu. Jediným implementovaným příkladem je zatím rychlost pohybu postav (týká se avatara i nepřátel), určená pomocí výchozí rychlosti a míry ovlivnění vlastnostmi určenými jako StatAvatarRunningSpeed a StatEnemyRunningSpeed.

```
#Speed
#===========
# Overall movement speed
BaseSpeed=10
SpeedFactor=20
```

Zbytek RPG systému již bude záviset na konkrétní implementaci.

6.2.3 Konkrétní implementace - ComplexRPGSystem

ComplexRPGSystem (realizovaný v balíku complex_rpg_system, obsahujícím 23 tříd) je příkladem implementace složitějšího RPG systému, navržen byl opět jako pokud možno univerzální a zároveň poskytující co nejširší možnosti.

6.2.3.1 Vlastnosti postavy

Systém umožňuje definovat sadu vlastností dvou typů, které mohou být navíc odvozené od ostatních a v průběhu hry mohou být ovlivňovány předměty nebo dovednostmi pomocí systému modifikátorů.

Prvním typem jsou vlastnosti s parametrem "fixed", které se navenek jeví jako jediná hodnota. Druhý typ je možné označit jako "energie", mají danou maximální hodnotu a hodnotu aktuální, která postupně klesá, je li daná "energie" čerpána (konkrétní příklad - magická energie a vyvolávání kouzel).

Odvozování vlastností

Třída ComplexTerm umožňuje popsat závislost vlastnosti na jedné nebo několika dalších vlastnostech. Je tak možné například definovat vlastnosti popisující šance postavy v soubojích (například "Útok" a "Obrana", jak byly použity v popisu třídy GMath) jako odvozené vlastnosti závisející na hlavních vlastnostech postavy (mohou se jmenovat třeba "Síla" a "Rychlost"). Pro přepočet bylo definováno několik vzorců pokrývajících nejběžnější možnosti závislosti:

```
#Formulas:
#---------
#0: value = statA
#1: value = statA + statB
#2: value = bonus(statA)  //for attack & defense
#3: value = bonus(statA) + bonus(statB)
#4: value = (statA * statB) + bonus(statC)
#5: value = (statA * statB) + statC + bonus(statD) //energies
#6: value = (statA * statB) + bonus(statC) + bonus(statD)
#7: value = (statA * statB) + statC + bonus(statD) + bonus(statE)
```

Klíčové slovo bonus() se odkazuje na popsanou funkci ze třídy GMath.

Základní, maximální a aktuální hodnota vlastnosti

Systém udržuje stav vlastnosti jako tři hodnoty:

- základní hodnota - hodnota nastavená v okamžiku vytvoření postavy, mění se pouze, když hráč přidělí body získané postavou po dosažení vyšší úrovně. U odvozených vlastností je použit výsledek výpočtu ve třídě ComplexTerm.
- maximální hodnota - základní hodnota ovlivněná modifikátory
- aktuální hodnota - u vlastností zařazených jako "fixed" odpovídá maximální hodnotě, u "energií" se mění v závislosti na událostech ve hře, především průběhu soubojů. Na této hodnotě také závisí hodnoty odvozených vlastností.

Modifikátory

Modifikátory realizované instancemi třídy ComplexModifier a ComplexModifierSet umožňují ovlivnit hodnotu vlastnosti vnějšími vlivy, například zbraň držená v ruce přidá kladný modifikátor k hodnotám vlastností "Útok" a "Maximální zranění", oslabující kouzlo/dovednost aktivovaná některým z nepřátel bude mít opačný efekt.

Každý modifikátor je zařazen do jedné z kategorií "TYPE_ITEM", "TYPE_SKILL", "TYPE_STAT" nebo "TYPE_SPECIAL", tedy předměty, dovednosti, jiné vlastnosti a ostatní. Pak je možné definovat zvláštní pravidla pro některé specifické případy - například ze všech modifikátorů od předmětů může být použita pouze minimální nebo maximální hodnota (příklad použití jsou rychlost útoku a dosah zbraní).

Použití některého z těchto omezení je pro každou vlastnost nezávisle určeno v konfiguračním souboru (klíčová slova Sum, ItemMaxSkillSum a ItemMinSkillSum).

Příklad konfiguračního souboru

Konfigurační soubor popisující vlastnosti avatara:

```
#Number of statistics
Stats=24

#Pseudo statistics
Stat 0 LVL Level Sum level fixed
Stat 1 LVLHP HitPointsPerLevel Sum fixed
Stat 2 LVLSP SpellPointsPerLevel Sum fixed
Stat 3 HP_B HitPointsBase Sum fixed
Stat 4 SP_B SpellPointsBase Sum fixed

#Primary statistics
Stat 5 STR Strength Sum fixed
Stat 6 DEX Dexterity Sum fixed
Stat 7 INT Intelect Sum fixed
Stat 8 WLP Willpower Sum fixed
Stat 9 END Endurance Sum fixed
Stat 10 SPD Speed Sum fixed speed
Stat 11 CHR Charisma Sum fixed
```

```
Stat 12 LCK Luck Sum fixed

#Secondary statistics
#                    HP=LVL*LVLHP+HP_B+bonus(END)
Stat 13 HP HitPoints Sum Term: 5  0 1 3 9 vital
#                    SP=LVL*LVLSP+SP_B+bonus(INT)+bonus(WLP)
Stat 14 SP SpellPoints Sum Term: 7  0 2 4 7 8

Stat 15 ATT AttackRate Sum Term: 2  6 fixed
Stat 16 DEF DefenseRate Sum Term: 2  10 fixed
Stat 17 DMG Damage Sum Term: 2  5 fixed

Stat 18 SK01 FightingSkills Sum fixed
Stat 19 SK02 MagicSkills Sum fixed
Stat 20 SK03 MiscSkills Sum fixed

Stat 21 RNG Range ItemMaxSkillSum fixed
Stat 22 ASB AttackSpeedBase Sum fixed
Stat 23 ASP AttackSpeed ItemMinSkillSum Term: 0 22 fixed
```

Odvozené vlastnosti jsou skryté za ID 13 - 17 a také 23. Příklad konstrukce "Term: 5 0 1 3 9" značí, že byl použit vzorec 5 (value = (statA * statB) + statC + bonus(statD)) počítající hodnotu z vlastností 0, 1, 3 a 9.

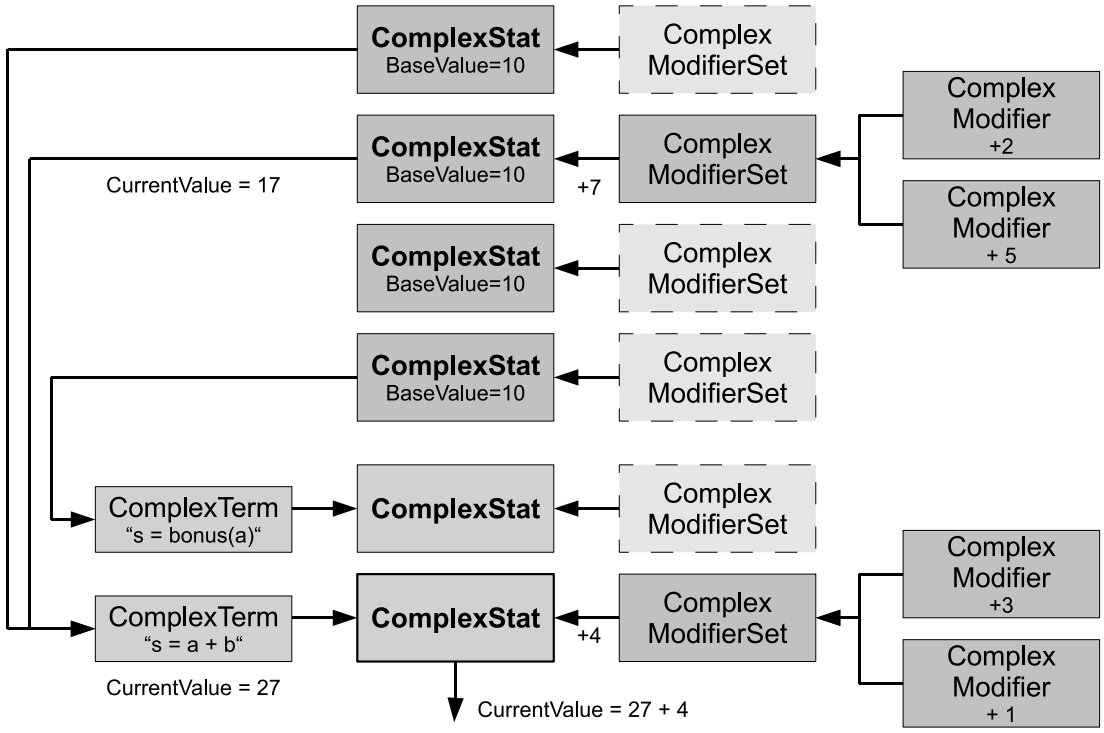
6.2.3.2 Třídy postavy

Třídy nebo kategorie postav používané při jejich tvorbě jsou realizovány ve třídách ComplexClassSet, ComplexClass a ComplexClassDatabase. Třída ComplexClass dědí od AbstractClass, třída ComplexClassSet od AbstractClassSet. ComplexClassDatabase zapouzdřuje celý systém a zajišťuje načítání konfiguračně-datového souboru. S jeho využitím si popíšeme implementovanou funkcionalitu.

Prvním záznamem v souboru je seznam vlastností avatara, které budou používány, a nastavení jejich výchozích hodnot pro "všeobecného" avatara. Jednotlivé třídy postavy v jednotlivých kategoriích pak budou tyto hodnoty modifikovat.

```
Initial stat values
#-------------------

StatValues:
StatsAffected=22
Stats  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20 21 22
Values 1 4 4 12 12 8 8 8 8 8 8 8 8  10 10 10 10 10 10 10 10 10
```

Následuje definice jednotlivých kategorií (ClassSet). Každá kategorie ovlivňuje podmnožinu vlastností, určených pomocí ID. Jednotlivé položky v dané kategorii pak obsahují odpovídající parametry měnící výchozí hodnoty jednotlivých vlastností. Je vhodné hodnoty volit tak, aby se celkový součet rovnal nule.



Obrázek 6.2: Možné propojení vlastností a modifikátorů

```
#Races as separate class set
#---------------------------

ClassSet:
Name="Race"
StatsAffected=9
Stats 5 6 7 8 9 10 11 12 21

Entries=5

Class 0 "Human"~0 0 0 0 0 0 0 0 0 Constraints=1 0
Class 1 "Elf"~-3 2 2 0 -3 0 2 0 0
Class 2 "Dwarf"~3 0 -1 -1 3 -2 -2 0 -2
Class 3 "Ork"~4 2 -3 -3 2 0 -3 1 2
Class 4 "Halfling"~-2 0 0 0 -2 4 0 0 -2 Constraints=2 0 2


#Class as another class set -> many combinations allowed
#-------------------------------------------------------

ClassSet:
Name="Class"
```

```
StatsAffected=7
Stats 1 2 3 4 18 19 20
Entries=5

Class 0 "Fighter"~2 -2 6 -6 3 -3 0 Constraints=1 0
Class 1 "Sorcerer"~-2 2 -6 6 -3 3 0  Constraints=2 6 7
Class 2 "Paladin"~1 -1 4 -4 1 -1 0
Class 3 "Archer"~1 -1 3 -3 1 0 -1
Class 4 "Alchemist"~-1 1 -2 2 -3 1 2
```

Constraints - návrh

Pomocí třídy ComplexConstraint bude možné definovat specifická omezení a znaky jednotlivých tříd v jednotlivých kategoriích. Často se jedná na návaznost na specifické funkce, které by musely být implementovány v různých dalších modulech.

Několik příkladů v konfiguračním souboru:

```
#Constraints
#-----------

Constraints=6

Constraint 0 "Quick progress in all skills"~4 Params: 10
Constraint 1 "Very quick progress in all skills"~4 Params: 20
Constraint 2 "Skills limited to Master proficiency level"~2 Params: 2
Constraint 3 "Skills limited to Expert proficiency level"~2 Params: 1
Constraint 4 "Stronger during day, weaker during night"~6 Params: 10
Constraint 5 "Very strong during night, very weak during day"~6 Params: -40
...
```

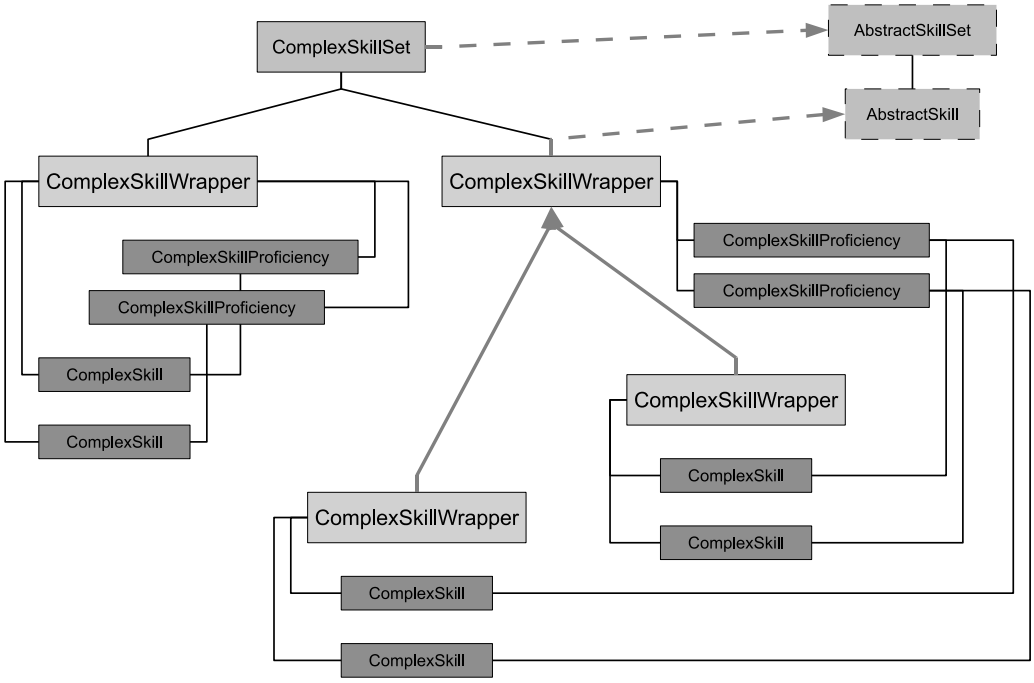
6.2.3.3 Systém dovedností

Systém dovedností je pravděpodobně nejrozsáhlejší částí vnitřní logiky každé RPG hry, zpracovává většinu akcí hráčova avatara stejně tak jako akce nepřátel. Je také součástí, ve které se jednotlivé hry nejvíce vzájemně odlišují.

Complex Skill System

Balík complex_skill_system je součástí RPG systému, která dědí od tříd AbstractSkillSet a AbstractSkill. Implementace se dá rozdělit na dvě části:

- Vývoj dovedností, jejich propojení a navázání na vlastnosti avatara a nepřátel
- Vnitřní logika realizující jejich účinky a propojení se zbytkem systému



Obrázek 6.3: Možné propojení dovedností

Třídy ComplexSkillWrapper, ComplexSkill a ComplexSkillProficiency

Třída ComplexSkillWrapper dědí od třídy AbstractSkill a zapouzdřuje veškeré atributy a funkce potřebné pro všeobecný popis jedné dovednosti (např. ovládání některého druhu zbraní, druhu magie nebo konkrétního kouzla). Nejdůležitějším atributem je úroveň dovednosti, tedy číselné vyjádření míry znalosti/účinků. Každá instance může odkazovat na rodičovskou instanci a počítat s jejími hodnotami – typickým příkladem je rodičovská instance – druh magie a několik kouzel – závislých instancí.

Pomocí instancí **třídy ComplexSkillProficiency** ("stupeň") je možné rozdělit znalost dané dovednosti do několika stupňů, každý stupeň je přístupný od určité úrovně dovednosti a obvykle přidává další účinky a ovlivňuje více vlastností.

Třída ComplexSkill popisuje konkrétní účinky/události spouštěné použitím dovednosti. V rámci jednoho SkillWrapperu je možné opět definovat několik instancí, například několik různě účinných verzí kouzla, obvyklý útok/několik druhů speciálních útoků v případě dovednosti popisující ovládání zbraně apod. Výběr konkrétního ComplexSkillu v rámci instance třídy SkillWrapper může být podle:

- ComplexSkillProficiency (propojení 1:1 proficiency:skill)
- Percentage – náhodný výběr, u instancí třídy ComplexSkill je možné definovat pravděpodobnost podle úrovně dovednosti a úroveň dovednosti, pro kterou je daná instance

dostupná.

- Selection – podobně jako předchozí, ovšem hráč dostane na daných úrovních na výběr, kterou instanci se jeho postava naučí.

Ukázky z konfiguračního souboru

V konfiguračním souboru jsou postupně popsány jednotlivé instance ComplexSkillWrapper, v jejichž rámci může být definováno několik instancí ComplexSKillProficiency a ComplexSkill.

Parametr SkillsBy určuje výběr konkrétních účinků, Parent pomocí ID odkazuje na případnou rodičovskou instanci. ItemClasses definují, při používání kterých předmětů se tato dovednost automaticky nastaví jako používaná.

U deklarace "Proficiency" jsou důležité parametry level, tedy úroveň, od které je daný stupeň dostupný, stat, vlastnost, ke které je jako modifikátor (ComplexModifier) přičtena úroveň dovednosti a automatic, určující, zda bude po dosažení dané úrovně dovednosti popisovaný stupeň zvolen automaticky (zatím jediná implementovaná možnost, alternativou by bylo nechat postavu vycvičit u některé NPC postavy).

Následuje výčet konkrétních účinků dané dovednosti, "Skills", důležitý je parametr "Type", určující konkrétní implementaci použitou při aktivaci.

Konkrétní příklady:

- Dovednost s účinky navázanými na Proficiency

```
#Skill          id   abbr     name
#-------------------------------------------
SkillWrapper     6   "BOW"     "Bow"

SkillsBy="Proficiency"
Parent=-1
Element=0
Passive="False"
ItemClasses:1   5

ProficiencyLevels=4

#Proficiency   id    abbr     name       level      stat   special     automatic    price
Proficiency     0    "NV"   "Novice"        1     "ATT"~       -1        "true"        0
Proficiency     1    "EX"   "Expert"        2     "ASP"~       -1        "true"        0
Proficiency     2    "MS"   "Master"        3     "ATT"~       -1        "true"        0
Proficiency     3    "GM"   "GrandMaster"   4     "ATT"~       -1        "true"        0

#Subskills      levels by prof.
Skills:4        0 0 0 0

Skill 0
Name="Shoot"
Type="Shoot"

Skill 1
Name="Piercing arrow"
```

```
Type="Shoot piercing"

Skill 2
Name="Double arrow"
Type="Shoot multiple"

Skill 3
Name="Piercing double arrow"
Type="Shoot multiple piercing"
```

- Dvě dovednosti, z nichž jedna závisí na druhé

```
#Skill          id   abbr     name
#--------------------------------------------
SkillWrapper     1    "FM"      "Fire Magic"

SkillsBy="Proficiency"
Parent=-1
Element=1
Passive="True"

ItemClasses:0

ProficiencyLevels=3

#Proficiency   id     abbr       name      level    stat   special   automatic    price
Proficiency     0     "NV"   "Novice"~       1     "x"        -1        "true"        0
Proficiency     1     "EX"   "Expert"~       6     "x"        -1        "true"        0
Proficiency     2     "MS"   "Master"~      12     "x"        -1        "true"        0

#Subskills       num      levels-when-available
Skills:0


#Skill          id   abbr     name
#--------------------------------------------
SkillWrapper     2    "FB"      "Fireball"

SkillsBy="Proficiency"
Parent=1
Element=0
Passive="False"
#other params based on parent

ItemClasses:0

ProficiencyLevels=0

#Subskills       num      levels-when-available
Skills:3        0 0 0

Skill 0
Name="Fireball (novice)"
Type="Bolt"
Stats:1 14
```

```
Effect=(6-6)+(1-6)
Consumption=(4-4)+(2-2)
ConsumedStats:1 14

Skill 1
Name="Fireball (expert)"
Type="Ball"
Stats:1 14
Effect=(8-8)+(1-8)
Consumption=(4-4)+(2-2)
ConsumedStats:1 14

Skill 2
Name="Fireball (master)"
Type="BallAimed"
Stats:1 14
Effect=(8-8)+(1-10)
Consumption=(4-4)+(1-1)
ConsumedStats:1 14
```

Zdokonalování postavy v ovládání dovedností

Systém bude nabízet tři možnosti:

Skill points při dosažení vyšší úrovně postava obdrží několik bodů, které hráč rozdělí mezi jednotlivé dovednosti. Autor má možnost specifikovat, podle jakého vzorce budou body přiděleny:

- X při dosažení úrovně X
- konstantní počet

Obdobně je volitelné, ovlivňuje-li úroveň dovednosti počet bodů potřebných pro její zvýšení:

- X pro dosažení úrovně X
- konstantní počet

Zdokonalování používáním každá dovednost (podobně jako samotná hráčova postava) si zaznamenává množství získaných zkušeností a s pomocí tabulky ze třídy ComplexExperience určuje svou úroveň dovednosti. Zkušenosti se získávají používáním, ať už aktivním (magie, zbraně) nebo pasivním (např. obranné dovednosti).

Tato možnost byla implementována a je nastavena jako výchozí.

Zdokonalování používáním + BoostPoints kombinace předchozích možností, umožňující navíc hráči více ovlivnit, na které dovednosti se jeho postava zaměří. Body získané při dosažení vyšší úrovně postavy se opět přidělují k dovednostem, v tomto případě ale zajistí rychlejší dosažení vyšší úrovně dovednosti.

Ovlivňování pomocí hodnot vlastností postavy a Constraints

Každý ComplexSkillWrapper může být navázán na jednu vlastnost (ComplexStat) postavy. Její hodnota pak ovlivní rychlost zdokonalování se v dané dovednosti.

Typický příklad jsou tři vlastnosti: "Boj", "Magie", "Ostatní"a rozdělení všech dovedností do těchto tří kategorií. Při tvorbě postavy má pak hráč další možnost, jak si její vlastnosti přizpůsobit.

Také některé z Constraints daných povoláním postavy se mohou týkat dovedností – např. "Rychlejší zdokonalování se"nebo "Omezení nejvyššího stupně ".

Vnitřní logika v rámci systému dovedností

Třída ComplexSkill obsahuje několik metod implementujících konkrétní funkcionalitu specifickou pro jednotlivé dovednosti. Výběr funkcí je navázán na atribut definovaný klíčovým slovem "Type"v konfiguračních souborech, možnosti jsou například:

```
Type="Attack"
Type="Double Attack"
...
Type="Shoot"
Type="Shoot piercing"
Type="Shoot multiple"
...
Type="Bolt"
Type="Ball"
Type="BallAimed"
...
Type="Buff"
```

Na každý z typů je navázán krátký úsek zdrojového kódu volající některé z definovaných funkcí.

Pro některé funkce je používána pomocná třída ComplexSkillValueExtent.

Třída ComplexSkillValueExtent popisuje číselný rozsah účinků, trvání a "spotřeby" (tedy jaké množství připojené vlastnosti - "energie" bude použito na jednu aktivaci) dovednosti. Obsahuje jediný vzorec, dávající číselnou hodnotu v podobě:

$V = random(A - B) + u \times random(C - D)$,

kde V je výsledná hodnota, u úroveň a random(X - Y) náhodná hodnota v rozsahu X - Y.

Pokud jsou relevantní, jsou v konfiguračním souboru parametry A, B, C, D definovány pro danou dovednost a všechny rozsahy (účinky, trvání a "spotřeba").

Metody třídy ComplexSkill je možné rozdělit na dvě skupiny:

První skupinou jsou metody zděděné od abstraktního rozhraní a sloužící pro aktivaci. Dvě metody activate() se liší v předávaném parametru skillLevel, který u dovedností použitých nepřáteli (kteří sdílejí jediný ComplexSkillSet) určuje míru efektu. U avatara si pak každá dovednost uchovává vlastní hodnotu a bude použita první verze metody activate.

Metoda activateSecondary() slouží například pro dovednosti v metodě activate() vytvářející pohyblivé objekty (střely apod.), je volána, když objekt dorazí k cíli.

```
public void activate(MapPosition sourcePosition, MapPosition destinationPosition, Map map,
                     int ownerID)
public void activate(MapPosition sourcePosition, MapPosition destinationPosition, Map map,
                     int ownerID, int skillLevel)
public void activateSecondary(MapPosition sourcePosition, MapPosition destinationPosition,
                              Map map, int ownerID)
```

Druhou skupinou jsou již konkrétní implementace efektů dovedností, zahrnující výpočet výsledků/důsledků vzájemných útoků postav (například metoda attack() pro útok avatara na nepřítele vyhodnocuje nejen úspěšnost, ale také případné získané zkušenosti pro danou dovednost).

Metoda createActiveObject() slouží k vytvoření pohyblivého objektu s daným cílem a referencí na dovednost.

Metoda createBuff() slouží autorům pro vytváření posilujících kouzel nebo schopností, dočasně přidá modifikátory k některým vlastnostem. Instance dovednosti si pak zachovává informaci o svém aktivním stavu (trvání popsáno pomocí ComplexSkillValueExtent) a po uplynutí daného času se v rámci metody onDurationExpired() deaktivuje. Opětovná aktivace na již aktivovanou dovednost způsobí pouze prodloužení časového intervalu na původní maximální délku.

```
public void attack(Avatar avatar, Enemy enemy, int range)
public void attack(Enemy enemy, Avatar avatar, int range)
public void attackElemental(Avatar avatar, Enemy enemy)
public void attackElemental(Enemy enemy, Avatar avatar)

public void createActiveObject(MapPosition sourcePosition, MapPosition destinationPosition,
                               Map map, int ownerID)
public void createBuff()
public void onDurationExpired()
```

6.2.3.4 Systém předmětů

Systém předmětů přesahuje rámec RPG systému, bude ale popsán jako celek. Nejdříve se zaměříme na část vnitřní logiky spojenou s postavami.

Předměty – herní logika

Nejběžnější cesta předmětu ve hře začíná vytvořením po poražení některého z nepřátel, kdy je předmět umístěn do mapy na jeho pozici. Pokud není hráčem přehlédnut, je sebrán a přesouvá se do inventáře. Pokud hráč vyhodnotí, že daný předmět má lepší vlastnosti, než avatarovo stávající vybavení, bude nějakou dobu používán, dokud ho nevystřídá vybavení opět o něco lepší.

Z pohledu implementace tak jde především o realizaci správy předmětů v rámci map, navázání systému předmětů a nepřátel (záznam v databázi, které druhy nepřátel budou "vytvářet které předměty") a implementaci inventáře.

Inventory

Inventář by se dal charakterizovat jako datová struktura obsahující množinu avatarova majetku. Skládá se ze dvou hlavních částí – slotů pro umístění používaných předmětů a zásobníku/zavazadla pro ostatní předměty ("backpack").

Vnitřní logika zahrnuje řadu metod pro manipulaci s předměty v závislosti na předaných událostech z uživatelského rozhraní.

Item slots - u slotů je předdefinovaná dvojice reprezentující levou a pravou ruku – jedná se o specifický případ, tyto předměty přímo ovlivňují herní logiku, pro příklad lze uvést výběr používané dovednosti v závislosti na předmětu drženém v pravé ruce. Ostatní sloty je možné definovat v konfiguračních souborech. Zvláštní vlastností slotů je parametr "stackable", umožňující do jednoho slotu umístit více předmětů stejné kategorie – typické využití je třeba u léčivých lektvarů.

Backpack - z pohledu implementace je backpack dvojrozměrné pole, na každou pozici je možné umístit libovolný předmět.

Rozdílné velikosti předmětů a předměty zabírající větší prostor (například 2×3 pole) nebudou uvažovány s odkazem na principy používané pro celý návrh.

Předměty v rámci RPG systému - realizace

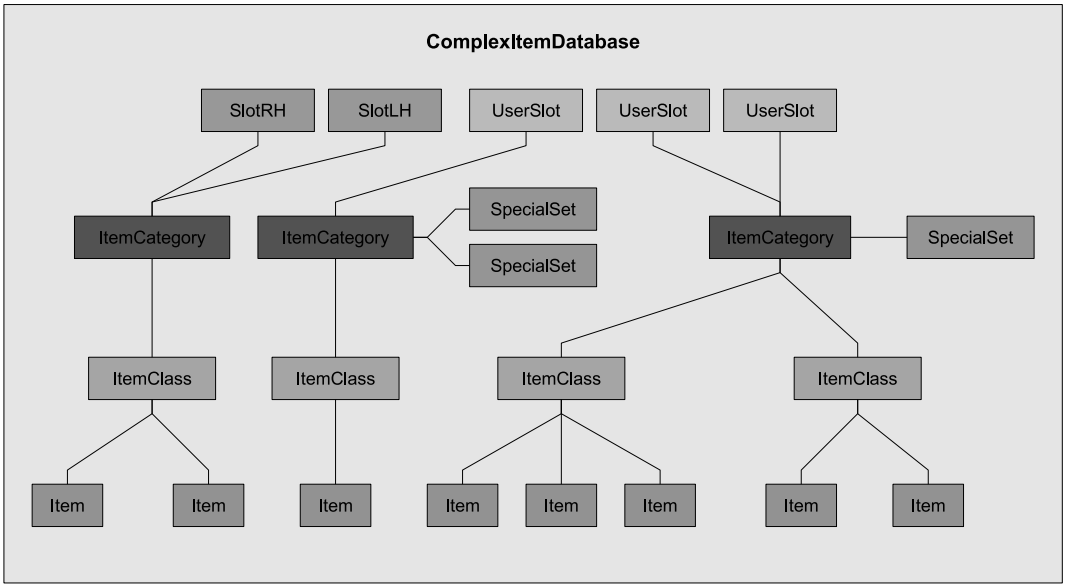
Implementovaný modul je tvořen třídami ComplexItemDatabase (dědí od AbstractItemDatabase), ComplexItemClass, ComplexItemCategory (dědí od AbstractItemClass, v realizovaném systému jsou implementovány dvě úrovně třídění předmětů), ComplexItemStatSet (dědí od AbstractItemStatSet) a ComplexItemSpecialSet (dědí od AbstractItemSpecialSet).

Třída ComplexItemDatabase obsahuje všechny instance datových struktur kategorizujících a popisujících předměty a implementaci jejich inicializace podle konfiguračního souboru.

Třída ComplexItemCategory zavádí kategorie, které umožní autorům rozdělit předměty podle co nejobecnějších parametrů – příkladem kategorií mohou být "brnění", "jednoruční zbraň" nebo "obouruční zbraň".

Kategorie určuje, které vlastnosti postavy budou modifikovány, když bude předmět používat a do kterého/kterých slotů v inventáři je možné předměty dané kategorie umístit.

Parametr "consumable" označuje předměty, které je možné jednorázově použít - jejich modifikátory se pak přímo přičtou k aktuálním hodnotám daných vlastností a předmět je odebrán ze systému.



Obrázek 6.4: Struktura databáze předmětů

Třída ComplexItemClass přináší druhou, podrobnější, úroveň dělení předmětů - "třídu předmětu". Použití je například v navázání dovedností - zbraně ze hry zasazené do fantasy prostředí, nazvané například "Obouruční meč" a "Obouruční sekera" budou obě spadat do stejné kategorie "obouruční zbraň", obě budou zabírat sloty pro levou i pravou ruku, obě budou ovlivňovat stejné vlastnosti. S pomocí tříd předmětů bude možné vytvořit třídy "meč" a "sekera" a následně je navázat na dovednosti "Boj s mečem" a "Boj se sekerou", které se mohou výrazně lišit.

Třída předmětu dále definuje atribut "kvalita", podle kterého je omezena účinnost další třídy: ComplexSpecialSet.

Třída ComplexSpecialSet umožňuje vytvářet "upravené"či "magické"předměty různých druhů.

Speciální vlastnosti předmětu jsou sadou modifikátorů, ComplexSpecialSet určuje, které vlastnosti postavy (ComplexStats) budou ovlivněny, obdobně jako u třídy ComplexItemCategory. Vždy je také určeno, na které kategorie předmětů se může daná instance uplatnit.

V praxi bývá většinou pouze malá část předmětů vygenerována se speciálními vlastnostmi, pokud je pro danou kategorii více možností sad speciálních vlastností, výběr bývá náhodný, stejně jako určení hodnot modifikátorů - maximum je omezeno kvalitou třídy předmětu, do které daný předmět spadá a parametrem předaným při vytvoření, obvykle bude navázán na některou vlastnost nepřátel.

Třída ComplexItemStatSet - zde jsou již definovány jednotlivé předměty – každému je přiřazeno jméno a patřičná třída předmětu.

Popis vlastností jednotlivých předmětů je jednoduchým přiřazením hodnot jednotlivým modifikátorům daným kategorií předmětu.

Podle konfiguračního souboru se vytvoří šablony jednotlivých předmětů, metodou createInstance() se vytvářejí objekty pro umístění do inventáře nebo do herní mapy. Ukázka konfiguračního souboru následuje:

```
# Categories
#===========
ItemCategories=12

ItemCategory 0
Name="Weapon one handed"
Stats:5 15 16 17 21 23
OverrideStats:0
Slots="RH"
UserSlots:0
Consumable="false"
Ammo=-1
...

# ItemClass ID Name Category
#==========================
ItemClasses=18
ItemClass 0 "Short sword" 0
ItemClass 1 "Long sword" 0
...

# Special Sets
#=============
SpecialSets=5
SpecialSet 0
Name="Powerfull"
ItemCategories:3   0 1 2
Stats:2  15 17
Skills:0
...

# The item list
# (Item ID Class Quality Subclass Name StatValues)
#==================================================
Items=16
Item 0 0 2 "Rusty sword" 2 0 4 2 10
Item 1 0 4 "Broadsword" 4 0 8 2 8
```

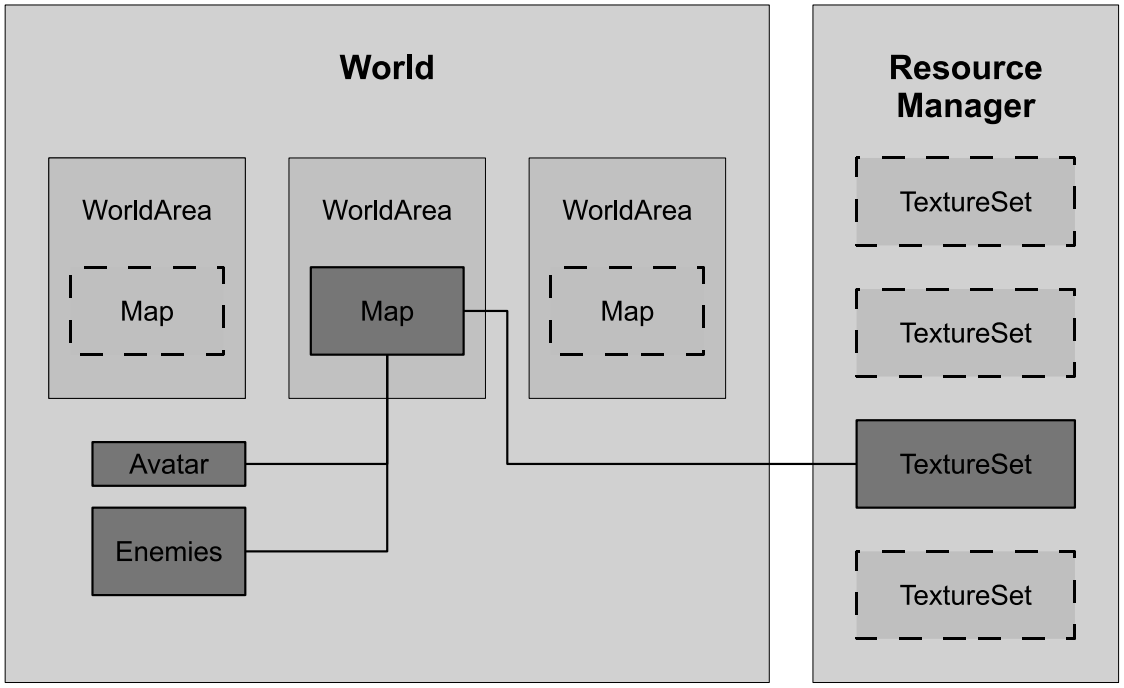
6.3 Herní svět

6.3.1 Úvod

Herní svět se obvykle skládá z různých exteriérů a interiérů, mezi kterými mohou postavy přecházet. Každé z těchto oblastí bude odpovídat jedna instance třídy WorldArea obsahující referenci na instanci třídy Map[1]. WorldArea udržuje všeobecné informace o oblasti, Map pak načítá terén v podobě čtvercové sítě a udržuje aktuální stav. Pohybovat se budeme pouze ve 2D prostředí, v úrovni úvah bude zmíněna možnost použití výškové mapy.

V paměti budou za běhu enginu načteny všechny WorldAreas, ale vždy jen jedna instance Map.

Logická vrstva, zpracovávající události a stav světa a postav, by měla být co nejméně závislá na grafice (přestože v některých případech, například u terénu, bude jisté propojení nezbytné), tak, aby bylo možné doplňovat další (různé) implementace grafického zobrazení.



Obrázek 6.5: Herní prostředí - struktura

[1] Výraz mapa nebude používán v obvyklém geografickém kontextu, ale ve významu obvyklém v počítačových hrách, tedy část herního světa, oblast

6.3.2 Mapy

Z pohledu implementace obsahuje třída Map několik stejně velkých dvojrozměrných polí ve kterých jsou uchovány:

- terén - v podobě celočíselných hodnot určujících druh terénu na daném poli, tedy odkaz na podrobnější popis jeho vlastností (průchodnost, viditelnost apod.) ve třídě TerrainTypeSet
- textury terénu - některým druhům terénu může být přiřazeno více různých textur
- odkazy na sektory, uchovávající pro každé pole informace o dynamických datech (přítomnost nepřátel, objektů, předmětů)
- možnost umístit při načtení mapy na dané pole nepřátele

6.3.2.1 TerrainTypeSets

Třída TerrainTypeSet a odpovídající konfigurační soubor obsahují popis druhů terénu, které je možné použít při tvorbě map. V současné implementaci je použit jeden společný konfigurační soubor pro všechny mapy, obsahuje výchozí set druhů terénu, který také určuje ID textur pro jednotlivé typy. Pro každý typ je definováno, zda je průchozí, je-li skrz něj vidět a je-li možné přes něj případně střílet.

- základní průchozí typ + 15 variací
- dva další průchozí typy (například cesty)
- dva neprůchozí typy, přes které je ale stále vidět a je možné přes ně střílet (voda apod.)
- jeden neprůchozí typ, přes který je vidět, ale není ho možné prostřelit
- dva neprůchozí neprůhledné typy (skály)
- množství nezávislých průchozích a neprůchozích variací

Důležitý je vztah k texturám: základní typy terénu jsou navržené tak, aby mezi sebou navazovaly (definováno klíčovým slovem "set"), každý typ tedy odpovídá 16ti texturám (kombinace pro 4 směry, podle toho, zda na dané straně pokračuje stejný nebo odlišný terén).

Uvedu i příklad konfiguračního souboru, ID textur jsou uvedeny v komentářích:

```
#Map terrain types - configuration file - default set

#Number of terrain types
TerrainTypes=151

#Terrain types - the list

#Basic terrain + variations
TerrainType 0 Grass-default passable visible shootable
TerrainType 1 Grass-variation01 passable visible shootable
```

```
TerrainType 2 Grass-variation02 passable visible shootable
...
TerrainType 14 Grass-variation14 passable visible shootable
TerrainType 15 Grass-variation15 passable visible shootable

# Road1: 16 - 31
TerrainType 16 Road1 passable visible shootable set

# Road2: 32 - 47
TerrainType 17 Road2 passable visible shootable set

# Water 48 - 63
TerrainType 18 Water visible shootable set

# Swamp 64 - 79
TerrainType 19 Swamp visible shootable set

# Forest 80 - 95
TerrainType 20 Bushes visible set

# Rock 96 - 111
TerrainType 21 Rock1 set

# Rock2 112 - 127
TerrainType 22 Rock2 set

# Special terrain-types - passable, 128 - 159
# example of connecting terrain types: bridges
TerrainType 23 Passable-bridge1 passable visible shootable connects
TerrainType 24 Passable-bridge2 passable visible shootable connects
TerrainType 25 Passable-terrain-type-03 passable visible shootable
TerrainType 26 Passable-terrain-type-04 passable visible shootable
...

# Special terrain-types - solid
# (not passable, not visible), 224 - 255
TerrainType 119 Solid-terrain-type01
TerrainType 120 Solid-terrain-type02
...
TerrainType 149 Solid-terrain-type31
TerrainType 150 Solid-terrain-type32
```

Dané typy a kombinace byly navrženy s jednak s ohledem na univerzalitu (umožňují dostatečnou variabilitu v rámci jedné mapy, ať už interiérové nebo exteriérové), nezávislost na sadě textur, která se často bude při přechodech mezi jednotlivými mapami měnit a rozumný počet textur - 256.

Konkrétní sada textur navíc nemusí být úplná, může obsahovat jen textury skutečně použité při designu mapy.

6.3.3 Sektory

Již bylo zmíněno, že každé pole mapy obsahuje "sektor" pro práci s dynamickými daty. Hlavním účelem použití sektorů je omezení vzájemného testování kolizí mezi postavami navzájem

a postavami a objekty. Pokud zavedeme omezení, že postava (při reprezentaci kruhovou ohraničující obálkou a s pozicí v plovoucí řádové čárce) nebude zabírat více, než jeden sektor, stačí vždy testovat pouze průniky postav/objektů v okolních devíti (3x3) sektorech.

Implementace se nachází ve třídě Sector v balíku world.

6.3.3.1 Sektory - interaktivita

Dalším využitím sektorů je snadné vytvoření statických (nepohyblivých) interaktivních objektů, zabírajících celý sektor, například:

- dveře, dveře ovládané jiným objektem
- přepínače dvou typů - jeden pouze aktivovatelný, druhý umožňující přepínání mezi dvěma stavy - umožňují ovlivňovat jiný objekt
- teleport přenášející postavu na jiné místo na mapě nebo využitelný jako vchod/východ pro přechod mezi mapami
- různé kulisy, umožňující například převážit průchodnost danou terénem

Tato funkcionalita je implementována ve třídě SectorFunctionality v balíku world. U každého objektu je možné nastavit, jakým způsobem bude aktivován:

- Dotykem (Touch) - objekt se automaticky aktivuje, když postava vstoupí na dané pole
- Přímá aktivace (Press) - aktivace je spuštěna přes uživatelský vstup (kliknutím myši)
- Nepřímá aktivace (None) - objekt je možné aktivovat pouze přes připojený přepínač

Vzájemné (jednosměrné) propojení objektů je realizováno přes souřadnice, u přepínačů je zadána pozice cíle.

V rámci implementovaného systému by bylo možné doplnit další varianty objektů:

- dveře se zámkem, k jejichž otevření je potřeba vlastnit specifický předmět (klíč)
- objekt, který po aktivaci do mapy na určené místo umístí zvoleného nepřítele
- různé druhy pastí
- přepínač ovlivňující více objektů najednou
- objekt aktivovatelný pouze specifickou kombinací přepínačů

6.3.4 Příklad konfiguračního souboru

Následuje ukázka konfiguračního souboru s popisem herního světa. Prvních několik řádků definuje virtuální velikost světa jako celku (v jednotkách "mapy" x "mapy"), určuje mapu načtenou při prvním startu hry a pozici avatara v ní.

Následuje postupná definice oblastí (WorldAreas), pro každou je určeno, který soubor s mapou načíst, jakou sadu textur zvolit pro terén a nepřátele, je dána virtuální pozice v rámci světa a popsáno, jaké druhy nepřátel a v jakém množství do mapy rozmístit.

Pro každou oblast je dále popsána množina objektů (SektorFunctionality, "FunctionalSectors").

```
#World
WorldSize:2 3 3
StartingArea=0
StartingSector:2    10 10

#WorldArea:
#World position - virtual value, x y depth

WorldAreas=2

WorldArea 0
Name="Beginner's Area"
MapFile="00.map"
TerrainTextureSet=0
EnemyTextureSet=2
WorldPosition:3    1 1 0
EnemyCount=75
EnemyTypes:4    0 1 2 4

FunctionalSectors=7

FunctionalSector 0
Position:2     16 12
Type="DOOR"
Activation="NONE"
Effect=-1
EffectPosition:2 0 0
Activated="false"
Passable="false"

FunctionalSector 1
Position:2     17 12
Type="SWITCH"
Activation="PRESS"
Effect=-1
EffectPosition:2 16 12
Activated="false"
Passable="false"
...
```

6.3.5 Pohyblivé objekty

Ve hrách se bude objevovat ještě další typ objektů, reprezentující střely, kouzla a různé pohyblivé objekty obecně. Důležité je, že na každý takovýto objekt je navázána konkrétní dovednost, která určuje účinky. Konkrétní implementace byla rozebrána v podkapitole "Vnitřní logika v rámci systému dovedností".

6.3.5.1 Implementace ve třídě ActiveObject

Implementován byl prozatím typ "Bolt", objekt, který se po umístění na danou pozici v mapě a zvolení cílové pozice začne po přímce pohybovat směrem k cíli, přičemž neustále testuje kolize s terénem nebo postavami, v případě kolize dojde k aktivaci připojené dovednosti. Stále se pohybujeme ve 2D prostředí, objekt je reprezentován jako ohraničující kružnice.

U objektu je možné zvolit, zda se po dosažení cíle objekt automaticky aktivuje nebo bude pokračovat ve vytyčeném směru, případně, zda bude po aktivaci kolizí s postavou smazán, nebo bude pokračovat po své trase a bude moci zasáhnout další postavy (parametr "piercing").

Dále bylo v implementaci ošetřeno, aby objekty pocházející od některého z nepřátel nekolidovaly s ostatními nepřáteli, zatímco objekty vyslané avatarem ano. Toto řešení sice není realistické, ale umožňuje lépe vyvážit souboje, navíc odpadá potřeba speciálně ošetřovat kolizi s postavou, která objekt vyslala.

Dalšími možnými typy objektů k implementaci by mohly být:

- Lightning/blesk - realizovaný jako úsečka nebo množina úseček, zasaženy by byly všechny postavy, jejichž ohraničující obálky by byly protnuty
- Nova - kruh s postupně se rozšiřujícím průměrem

6.4 Postavy ve hrách

6.4.1 Úvod

V práci jsem se zatím zabýval dvěma druhy postav - hráčovou postavou (avatarem) a nepřáteli, dalšími postavami jsou NPC, obvykle reprezentující obyvatele herního světa. V rámci práce zůstaneme ve fázi návrhu - možností realizace je více, například pokud nebudeme vyžadovat, aby se tyto postavy volně pohybovaly, stačí zavést další druh interaktivních objektů v rámci "SectorFunctionality", který po aktivaci zobrazí dialog mezi NPC a avatarem. Pokud by ale záměrem bylo doplnit postavy, které se budou aktivně zapojovat do soubojů na hráčově straně, šlo by o velmi komplikovaný problém, řešitelný například odděděním od třídy popisující nepřátele a doplnění potřebné funkcionality.

Zpět k avatarovi a nepřátelům: při implementaci bude použita dědičnost, rodičovská třída Character bude obsahovat data společná všem postavám. V prvé řadě je to navázání na RPG systém - každá postava je popsána sadou vlastností. Druhou důležitou společnou proměnnou je pozice na mapě.

Další výhodou přístupu založeného na dědičnosti je zajištění společného rozhraní pro ostatní části systému.

6.4.2 Pozice postav

Mnoho her s prostředím omezeným čtvercovou mřížkou omezuje podobným způsobem i pozice a pohyb postav, ať již jde o "skoky" mezi poli nebo na pohled plynulý přechod doplněný animací.

V implementovaném systému se budou postavy naopak pohybovat volně - jde sice o výrazně komplikovanější řešení, přesto má smysl je realizovat.

Pro uchování pozice postav i objektů byla vytvořena třída MapPosition, udržující pozici postavy v plovoucí řádové čárce v následující podobě:

- celočíselná pozice určující sektor
- offset v rámci sektoru v rozsahu $\langle 0, 1)$ uložený jako double

Dále třída obsahuje metody pro změny pozice (s přepočítáním hodnot sektoru a offsetu) a výpočty vzdáleností a směrů.

6.4.3 Avatar

Třída Avatar, popisující hráčovu postavu, oproti třídě Character navíc přidává metody reagující na uživatelské vstupy a specifické situace ve hře. U referencí na RPG systém přidává odkaz na sadu dovedností a udržuje odkaz na dvě aktuálně používané dovednosti.

Specifické pro Avatara je navázání na systém předmětů - třída také obsahuje inventář, popsaný třídou Inventory.

6.4.4 Nepřátelé

Situace u implementace nepřátel je výrazně složitější, na rozdíl od avatara nejsou nikým ovládány a proto je třeba zajistit, aby se samostatně pohybovali po mapě a byli schopní se rozhodovat v rámci soubojů.

Z pohledu autora pak půjde o mnoho času stráveného vyvažováním množství a druhů nepřátel na mapě a jejich vlastností v rámci RPG systému tak, aby výsledek splnil svůj účel (na začátku hry umožnit hráčům seznámit se s pravidly soubojů konfrontací se slabými nepřáteli, v pozdějších fázích postavit hráče před výzvy v podobě množství silných nepřátel).

6.4.4.1 EnemyDB

Třída EnemyDB a odpovídající datový soubor obsahují informace o druzích nepřátel. Každý záznam o jednom druhu obsahuje výchozí hodnoty vlastností pro vytvoření instance sady vlastností v modulu RPG systému, ID a úrovně dovedností, které nepřátelé tohoto druhu ovládají a také informace o předmětech - tedy jaké druhy předmětů se mohou objevit po jejich poražení, jaké jsou pravděpodobnosti vygenerování předmětu a - pokud již bude vytvořen - že bude mít speciální vlastnosti.

```
#                     LVL HP   SP    Att Def Dmg  spdA spdR   size range exp money
Enemy 1 "Ork_Leader" 2   16   0     6   8   4    6    10     40   20    20  0
Skills:1 0
SkillLevels:1 1
Items:3 0 1 4
ItemProbability=20
ItemSpecialProbability=10

#Ranged                     LVL HP   SP    Att Def Dmg  spdA spdR   size range exp money
Enemy 2 "Lizardman_Shooter" 8   60    0    14  18  4    4    15     35   260   30  0
Skills:3 1 2 3
SkillLevels:3 1 1 1
Items:4 2 3 4 5
ItemProbability=40
ItemSpecialProbability=25
```

6.4.5 Pohyb postav

V každém průchodu hlavní smyčkou programu se vyhodnocuje stav nepřátel a u aktivních se počítá nová pozice.

6.4.5.1 Chování nepřátel, umělá inteligence

Umělá inteligence (v našem případě model chování nepřátel a jejich schopnost najít cestu ke svému cíli) bude dalším modulem s možností volby implementace. Přesto bude konkrétní implementace opět navrhována jako konfigurovatelná a rozšiřitelná.

V žánru RPG je obvyklou praxí u nepřátel upřednostňovat spíše kvantitu než kvalitu a pokud je hráč postaven proti silným nepřátelům, vyplývá jejich nebezpečnost spíše z hodnot vlastností výrazně převyšujících vlastnosti avatara než z pokročilé umělé inteligence. Proto by

bylo možné v implementaci popsat jejich chování pouze několika pravidly a parametry a i tak dosáhnout použitelného výsledku. Přesto, především kvůli možnostem pozdějšího rozšíření, bude navržen poněkud sofistikovanější model, používající pro popis chování nepřátel konečný automat.

Konečný automat

Stavy konečného automatu (FSM - Finite State Machine) popisujícího chování nepřátel budou odpovídat konkrétní činnosti (nepřítel je neaktivní/pronásleduje avatara a útočí, když je v dosahu/snaží se uprchnout/náhodně se pohybuje po mapě), přechody budou definované jen mezi některými stavy a budou navázány na nějaké detekovatelné události.

Mezi argumenty vedoucí k této volbě řešení patřily:

- Možnost v budoucnu popsat přechody mezi stavy pomocí nějakého skriptovacího systému
- Pro autory dostupné snadné řešení - ve zdrojových kódech popsaný základní stavový automat použitelný pro většinu nepřátel

Realizace - použity jsou třídy StateMachine, StateMachineState, StateMachineTransition a již popsané třídy Trigger a EventCreator. Struktura je zřejmá - StateMachine obsahuje pole stavů StateMachineState a každý stav pole přechodů StateMachineTransition.

StateMachineTransition navíc v sobě obsahuje na sebe navázané instance tříd Trigger a EventCreator - vyžádání změny stavu prochází přes systém zpracování událostí. Toto řešení umožňuje reagovat na libovolné detekovatelné události: vzdálenosti od avatara, hodnoty vlastností překračujících zadané meze, uplynulý čas. Metoda

```
createBasicEnemyMachine(Enemy enemy)
```

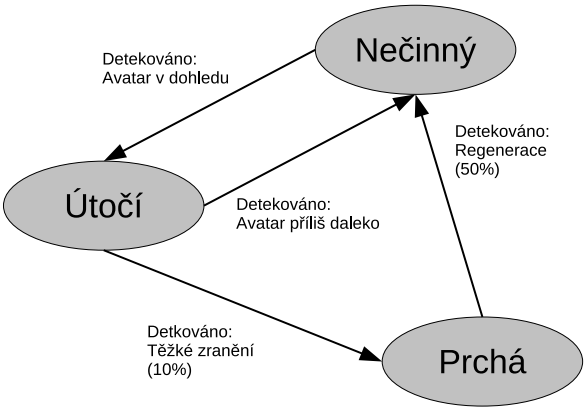
ve třídě StateMachine pak vytvoří základní stavový automat - obrázek 6.6 ho přehledně popisuje.

Hledání cest

Hledání cest nepřátel k avatarovi je jedním z typických problémů, které lze převést na prohledávání stavového prostoru.

Při návrhu je nutné brát v úvahu:

- libovolné umístění nepřátel nad pravidelnou čtvercovou strukturou - bude stačit, když bude jedno pole mřížky odpovídat jednomu uzlu grafu?
- nepřátelé musí být schopní se navzájem vyhýbat a řešit případné zablokování
- měnící se podmínky - jako často bude trasa přepočítávána?
- výpočetní náročnost, závislá na počtu najednou aktivních nepřátel



Obrázek 6.6: Výchozí automat popisující chování nepřátel
(zranění a regenerace podle vlastnosti přiřazené ve třídě GMath jako "statEnemyLife")

Při realizaci pak bude třeba implementovat algoritmus prohledávání a reprezentaci stavového prostoru.

Stavový prostor bude realizován navigačním grafem, implementovaným ve třídě MapNavigatonGraph, obsahující dvojrozměrné pole uzlů, instancí třídy MapNode. Každý uzel uchovává informaci o své pozici na mapě.

Inicializovány jsou pouze uzly ležící na polích s průchodným terénem. Každý uzel je pak propojen s osmi sousedními uzly (tedy i po úhlopříčkách).

Pro dočasné uložení nalezených cest slouží třída MapRoute, která obsahuje sekvenci uzlů MapNode rekonstruovanou z výsledku prohledávání.

Při volbě algoritmu bude potřeba jeho chování co nejdříve otestovat v praxi. Smysl má jistě použít některý z popsaných algoritmů - jako vhodný kandidát se jeví v podobných situacích nejběžněji používaný A*.

A* je algoritmus prohledávání grafu, který najde nejkratší cestu mezi výchozím a cílovým uzlem. Jde o algoritmus typu best-first, informované prohledávání s využitím heuristické funkce odhadu vzdálenosti z aktuálního stavu do koncového uzlu. Uzly jsou vybírány prioritně podle minimálních hodnot funkce:

$f(x) = g(x) + h(x)$,

kde g(x) je vzdálenost aktuálního uzlu od počátečního uzlu a h(x) heuristický odhad vzdálenosti do koncového uzlu.

Datovou strukturou pro uložení stavů, která charakterizuje A*, je prioritní fronta. Přímo v knihovnách Javy existuje generická třída PriorityQueue<Type>, pro dosažení požadované funkčnosti je potřeba v třídě jejích prvků, v našem případě MapNavigationState implementovat funkci compareTo() pro porovnávání objektů - jde o realizaci zmíněné heuristiky:

```
public int compareTo(MapNavigationState otherState)
{
    if (distanceFrom + distanceTo < otherState.distanceFrom + otherState.distanceTo)
    {
        return -1;
    }
    else
    {
        return 1;
    }
}
```

Samotné hledání cest je realizováno v třídě MapNavigationGraph v metodě

```
public MapRoute findTheWay(MapNode nodeFrom, MapNode nodeTo),
```

odkud ho mohou využívat různé implementace umělé inteligence.

Při ohodnocování uzlů je kromě vzdálenosti také potřeba zvýhodnit volná pole - nejdřív vyzkoušíme jednoduché řešení (po testování se ukáže, že pracuje překvapivě dobře): uzly v navigačním grafu jsou ohodnoceny podle počtu nepřátel v sektoru odpovídajícím danému uzlu - zaplněné uzly jsou tedy méně výhodné. Pokud se někde vytvoří zástup nepřátel, budou další upřednostňovat jinou (volnou) cestu, i kdyby byla delší.

Nepřátelé schopní najít cestu mezi dvěma sektory na mapě se již mohou začít s jejím využitím pohybovat:

Třída EnemyAIBasic dědí od abstraktní třídy AbstractEnemyAI a zapouzdřuje realizovanou implementaci chování nepřátel, každý nepřítel na mapě má vlastní instanci. Ta obsahuje konečný automat a instanci "MapRoute" (nalezenou trasu) s ukazatelem na aktuální uzel.

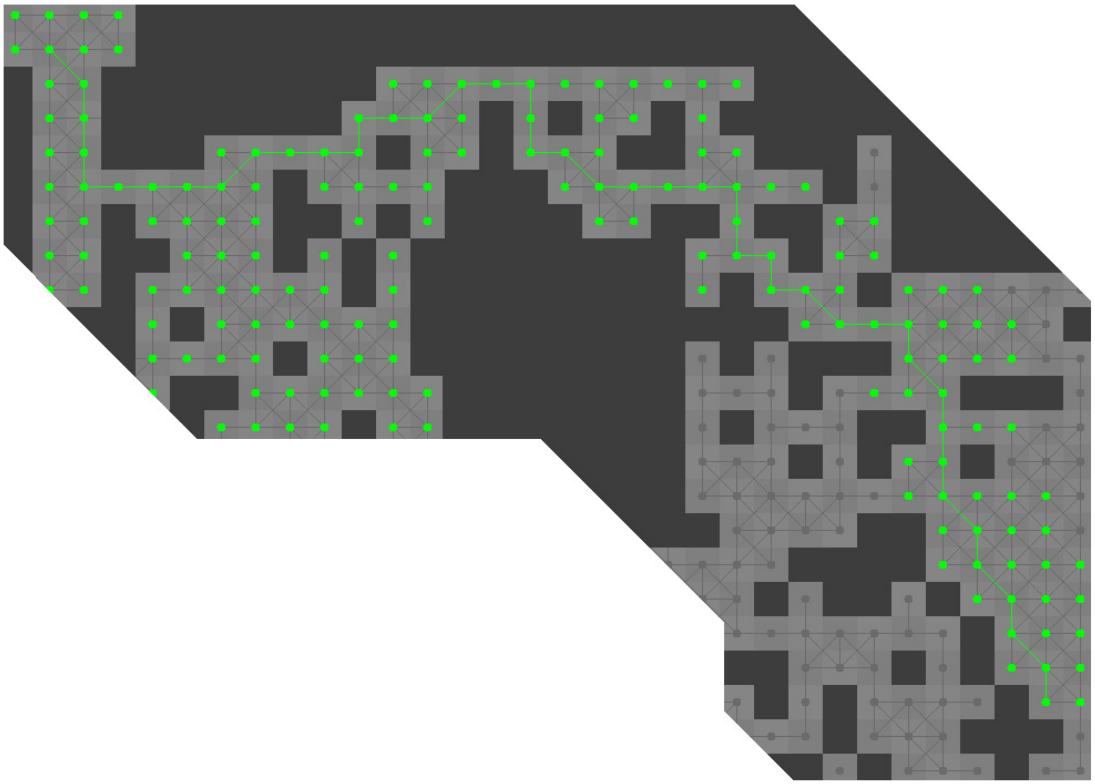
Aktivní nepřátelé se pohybují v přímém směru k aktuálnímu uzlu trasy, přiblíží-li se dostatečně k dalšímu uzlu, ukazatel se přesune. Při volbě větší "uzlově-reakční" vzdálenosti se výsledná trasa viditelně vyhlazuje, hrozí však zablokování o terén.

Pokud je vzdálenost k cíli (avatarovi) dostatečně malá, vydá se již nepřítel po přímé trase nezávisle na posledních uzlech trasy.

Pokud není nalezena cesta (nemusí existovat, nebo je příliš dlouhá), snaží se nepřátelé také pohybovat v přímém směru. Třeba se dostanou na dostřel nebo na pole, ze kterého již algoritmus cestu najde.

Pokud je již nepřítel tak blízko k avatarovi, že může útočit, dále se nepřibližuje a začne se pohybovat po kružnici, jejíž střed je dán pozicí avatara. Hlavním důvodem je usnadnění přístupu dalším nepřátelům.

Implementováno bylo i jednoduché řešení kolizí - především v situaci, kdy se několik "velkých" nepřátel seskupí před úzkým průchodem, může dojít k zablokování. Každý nepřítel má počítadlo cyklů, ve kterých se nemohl pohybovat. Když čeká příliš dlouho, zkusí se vydat několik kroků opačným směrem. Když se použijí náhodné hodnoty, situace se po nějakém čase vyřeší. Dobrou prevencí ze strany autora je volba rozumné velikosti nepřátel vzhledem k sektoru - například omezení na 75% velikosti sektoru/pole.



Obrázek 6.7: Příklad nalezené cesty v relativně komplikovaném prostředí
(Pozn.: všechny prozkoumané uzly jsou zvýrazněny zeleně, zbytek grafu naznačen šedě)

Parametry systému Především pro vyvážení účinnosti a výpočetní náročnosti je systém popsán několika parametry:

- maximální hloubka prohledávání
- vzdálenost k dalšímu uzlu pro přehození ukazatele (reflektuje velikost nepřítele)
- vzdálenost k cíli pro přechod na přímý pohyb
- počet cyklů čekání v zablokovaném stavu
- vzdálenost (počet kroků) při pokusu vymanit se ze zablokovaného stavu
- frekvence přepočítávání cesty

(poslední 3 parametry je vhodné mírně randomizovat - vypadá lépe, když si všichni nepřátelé "nerozmyslí" svůj pohyb ve stejný okamžik, pro řešení blokací je také potřeba vyzkoušet několik náhodných kombinací pohybů - zvláště, když jde o velké seskupení nepřátel)

Po postupném vyladění jednotlivých parametrů se podařilo docílit uspokojivého výsledku - pohyb nepřátel není sice vyloženě realistický, ale nevypadá nepřirozeně a plní svůj hlavní účel: nepřátelé jsou schopni se efektivně dostat k avatarovi a ve složitějších situacích volí napohled vhodnější trasy.

Jediným problémem je možné zablokování velkých nepřátel, pokud se jich sejde velké množství na malém prostoru, například úzké chodbě. Naštěstí je to ale poměrně neobvyklá situace, které se autoři mohou různými způsoby vyvarovat (menší nepřátelé, dobře navržené prostory, umísťování nepřátel jen do určitých částí mapy).

6.4.6 Souboje

Průběh soubojů je z pohledu systému sérií akcí avatara a nepřátel, využívajících funkce ostatních částí enginu, především RPG modulu - konkrétně:

- pohyb postav podle událostí generovaných uživatelským vstupem nebo modulem umělé inteligence
- aktivace dovedností (útoky, zvláštní schopnosti - RPG systém)
- zpracování výsledků, například porážky nepřítele - RPG systém ve spolupráci s implementací herního světa

Typy soubojů - jak bylo zmíněno již v analýze existujících systémů, i souboje mohou být realizovány různými způsoby:

- **V reálném čase** - avatar i nepřátelé mají určeno (dle některé vlastnosti), jak často jsou schopní používat své dovednosti, akce se vyhodnocují v každém průběhu hlavní smyčky programu.

  Tento systém byl implementován v rámci práce.

- **Na kola ve vyhrazeném prostředí** - postavy se postupně střídají v provádění akcí (pořadí a frekvence může záviset na vlastnostech postav), souboj probíhá tak dlouho, dokud není jedna strana poražena nebo ze souboje neuteče (nabízí-li systém tuto možnost). Vzájemná poloha a vzdálenost postav může být zcela zanedbána.

- **Kombinace - souboje na kola přímo v mapě** - může vzniknout modifikací prvního typu soubojů zavedením zásobníku "akčních bodů", jejich počet bude opět záviset na některé z vlastností. Tyto body budou přiděleny avatarovi v každém kole a hráč je postupně vyčerpá na pohyb a používání dovedností. Následně proběhne určitý počet obvyklých iterací hlavní smyčky programu, která realizuje akce nepřátel, zatímco hráčova postava bude neaktivní.

  Zajímavou možností doplnění implementace by bylo umožnit hráči přepínat mezi možnostmi 1 a 3.

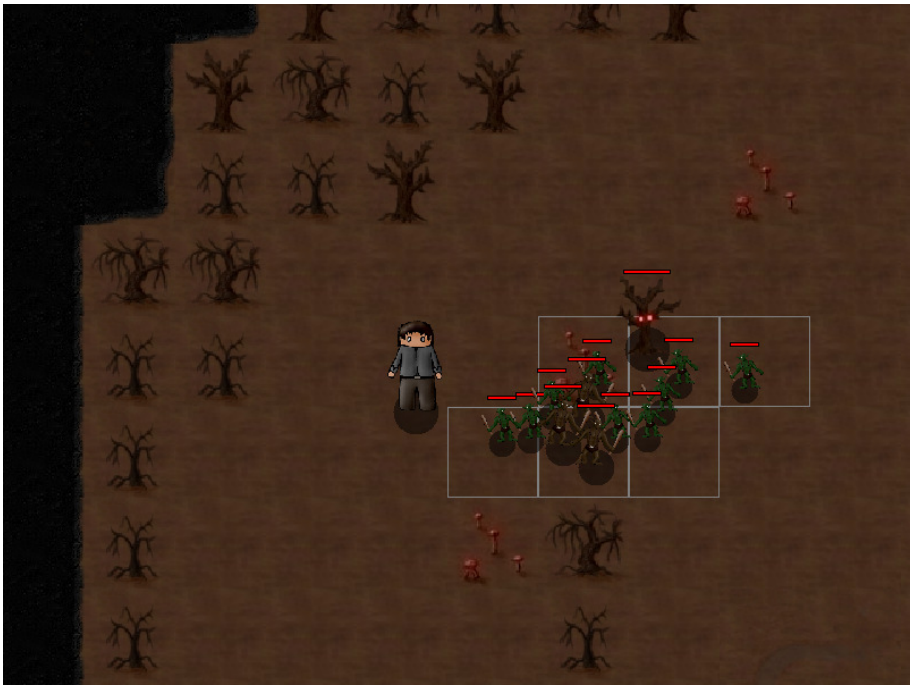
Realizace - jediná funkcionalita, kterou je potřeba doplnit, je práce s informací, zda jsou nepřátelé či avatar připraveni znovu použít některou dovednost.

Při implementaci byla vytvořena třída **ReadyGauge**, dědící od třídy Trigger. Je tedy schopná přijímat impulsy hlavních hodin GlobalClock.

Třída obsahuje dvě celočíselné hodnoty - aktuální hodnotu a maximum (konstantu), pokud je aktuální hodnota rovna maximu, je postava připravena. V závislosti na hodnotě navázané vlastnosti se určitou rychlostí zvyšuje aktuální hodnota až do dosažení hodnoty maximální. Obě hodnoty je možné využít například při tvorbě grafických prvků informujících hráče o stavu avatara a nepřátel.

Dále je na ReadyGauge napojena instance třídy EventCreator, generující události ve chvíli, kdy je dosaženo maxima - na tyto události jsou pak navázány dovednosti nepřátel.

U avatara je situace jednodušší, pokud je připraven, je možné použít libovolnou dovednost, čímž se stav ReadyGauge (tedy aktuální hodnoty) vynuluje. Různá časová náročnost různých dovedností může být realizována navázáním dovednosti na vlastnost, která ovlivňuje doplňování ReadyGauge.



Obrázek 6.8: Výřez mapy s avatarem a nepřáteli

(Pozn.: šedě označeny jsou aktivní sektory, vytvořené pro testování kolizí a uchování seznamu nepřátel, červené sloupečky nad nepřáteli ukazují stav jejich vlastnosti určené v třídě GMath jako "statEnemyLife")

Herní svět je tedy oživen a naplněn bytostmi usilujícími o avatarovo zdraví. Bude se však nacházet ve tmě, dokud nebude doplněna poslední část implementace enginu - grafické zobrazení.

6.5 Grafické uživatelské rozhraní her

Grafické uživatelské rozhraní ve hrách nejenže zobrazuje výřez mapy a různé informace ze hry (přehled o hodnotách nejdůležitějších vlastností, zobrazení předmětů v inventáři), ale obsahuje také vrstvu zpracovávající uživatelský vstup z klávesnice a myši.

6.5.1 Návrh

Ovládací prvky uživatelského rozhraní ve hrách často funkcí odpovídají běžným prvkům z desktopových aplikací, vzhledem se ale často výrazně liší - ať už proto, aby zapadly do vizuálního stylu hry nebo kvůli úspoře prostoru. RPG hry také až na vyjímky využívají zobrazení na celou obrazovku.

V případě realizované implementace budou uvažovány obě možnosti - tedy zobrazení v okně a zobrazení přes celou obrazovku.

Grafické rozhraní bude samo vykreslovat celou plochu hlavního okna, bez použití prvků grafického rozhraní ze systémových knihoven.

Systém bude strukturovaný - bude se skládat z grafických modulů, z nichž bude vždy pouze jeden aktivní (zobrazení mapy, menu, tvorby postavy). Moduly budou dále moci obsahovat různé ovládací prvky (textová pole, tlačítka). Při realizaci bude opět využito dědičnosti/polymorfismu.

6.5.2 Realizace

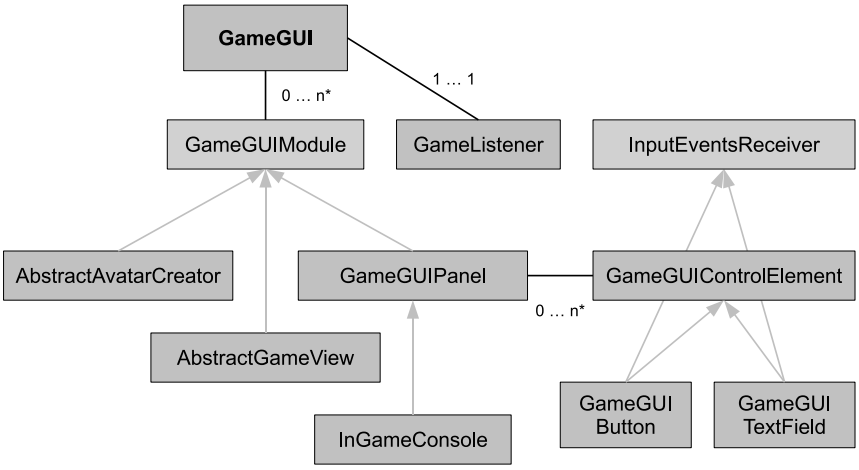
Hlavní zapouzdřující třídou je **GameGUI** (využívající Java GUI toolkit Swing), která zajišťuje:

- inicializaci celého grafického rozhraní, vytvoření hlavního okna (JFrame)
- realizaci back-bufferingu, buď s využitím BufferedImage pro zobrazení v okně nebo page-flipping při zobrazení na celou obrazovku
- správu grafických modulů, tedy jejich inicializaci, umísťení do okna, přepínání a především volání vykreslovací metody aktuálního modulu při příchodu impulsu z hodin grafického systému
- předávání uživatelského vstupu aktuálnímu modulu

6.5.2.1 Systém modulů

Interface **GameGUIModule** poskytuje rozhraní pro komunikaci modulů a GameGUI, obsahuje metody pro předávání pozice a velikosti a také uživatelského vstupu, v případě událostí předaných z myši s pozicí kurzoru přepočítanou do lokálních souřadnic v rámci modulu.

Třída GameGUIPanel umožňuje dále moduly dále strukturovat. Příkladem konkrétního použití je odděděná třída **InGameConsole**, učená k zobrazování barevně rozlišeného textového výstupu (ať už při ladění her, nebo pro poskytování informací hráčům), jejíž instanci je možné použít v libovolném z modulů.



Obrázek 6.9: Třídy v balíku gui - světleji Java interfaces, šipky značí dědičnost

Zvláštními případy grafických modulů, určenými k vytvoření konkrétních implementací, jsou **AbstractAvatarCreator** a **AbstractGameView** - první je pevně navázán na zvolený RPG systém, druhý vyžaduje vytvoření specifického modulu zobrazujícího výřez mapy s postavami.

Interface **InputEventsReceiver** poskytuje metody pro předávání uživatelského vstupu ovládacím prvkům, dědícím od třídy **GameGUIControlElement**. Každý prvek má metodu pro vlastní vykreslení, vše je realizováno pouze s využitím interních grafických knihoven Javy (především třídy Graphics/Graphics2D).

Uživatelský vstup na nižší úrovni je zpracováván třídou GameListener.

6.5.2.2 Uživatelský vstup - GameListener a GameInputEvents

Třída **GameListener** je typickou Java třídou zpracovávající uživatelský vstup - implementuje rozhraní MouseListener, MouseMotionListener a KeyListener a přijímá tedy události od myši i klávesnice vzniklé v rámci hlavního okna aplikace.

Zachycené události jsou dále transformovány na volání "standardizovaných" funkcí, které jsou implementovány napříč třídami uživatelského rozhraní:

```
void recieveMouseEvent(int x, int y, int button, int action);
void recieveKeyEvent(int keyCode, char keyChar, int action);
```

Pokud je aktivním modulem zobrazení mapy, je událost dále předána třídě **GameInputEvents** (pozice kurzoru myši se přepočítá na pozici na mapě), která z ní v konkrétních případech vytvoří událost (Event) ke zpracování v jádru enginu.

V případě událostí z klávesnice si systém ukládá stav všech kláves, důležité je to především u kláves navázaných na pohyb avatara (je možné zamezit obvyklé prodlevě po prvním stisku klávesy). Pokud je klávesa pro pohyb v daném směru stisknutá, je po každém impulsu od hlavních hodin vygenerována událost zajišující pohyb postavy.

6.5.2.3 AvatarCreator - uživatelské rozhraní pro tvorbu postavy

Prvním důležitým modulem je uživatelské rozhraní pro tvorbu postavy. Třída AbstractAvatarCreator, tvořící opět rozhraní pro zbytek systému, obsahuje jedinou důležitou metodu:

```
public abstract Avatar createAvatar();
```

Konkrétní řešení, zapouzdřené třídou ComplexAvatarCreator a navázané na implementovaný RPG systém, pak bylo navrženo obvyklým způsobem: podle prvků grafického rozhraní potřebných pro tvorbu postavy byla navržena struktura konfiguračního souboru a následně do několika tříd implementována potřebná funkcionalita. Jaké prvky tedy bude tento grafický modul obsahovat?

- textová pole pro zadání jména postavy, případně dalších údajů
- rozhraní pro výběr portrétu/obličeje postavy
- rozhraní pro výběr tříd postavy (povolání, rasa, ...)
- rozhraní pro nastavení hodnot množiny vlastností

Odpovídající konfigurační soubor může vypadat například takto:

```
Textfield:
Name="Name"
Position 500 60
Size 120 60
TextLength 24


FaceSelector:
ID=0
Name="Face"
Position 640 60
Size 120 120
Locked="true"


#Classes
ClassSelector:
Name="Class"
Position 500 370
Size 260 140
ForceFace=-1
```

```
StatPool:
StatsInPool=2
Stats 1 2
Position 20 160
Points=3
Min=1
Max=15
Variance=3
Locked=false
BonusWhenMax=0
...
```

U každého prvku je definována pozice a velikost v rámci okna, ostatní vlastnosti už jsou specifické. Z konkrétní funkcionality stojí za uvedení:

ClassSelector - výběr třídy postavy s možností navázat konkrétní obličej na danou třídu, tedy napojit instanci FaceSelector

StatPool - tato třída obsahuje hodnoty několika vlastností a určité množství bodů, které mezi ně hráč při tvorbě postavy rozdělí. Hodnoty je možné upravovat v obou směrech, až do určených hodnot "Min" a "Max". Hodnota "Variance" určuje, do jaké míry se mohou odchýlit od původní hodnoty. "BonusWhenMax" přidá k vlastnosti, jež byla nastavena na maximální hodnotu, ještě daný bonus. Poslední parametr, "Locked", umožní daný StatPool uzamknout - hodnoty vlastností budou dále ovlivňovány výběrem tříd postavy, ale StatPool bude sloužit pouze pro informaci.

Systém jako celek umí zkontrolovat, zda jsou rozděleny všechny body a vyplněny všechny hodnoty, poté je po stisknutí příslušného tlačítka vytvořena postava a spuštěna hra.

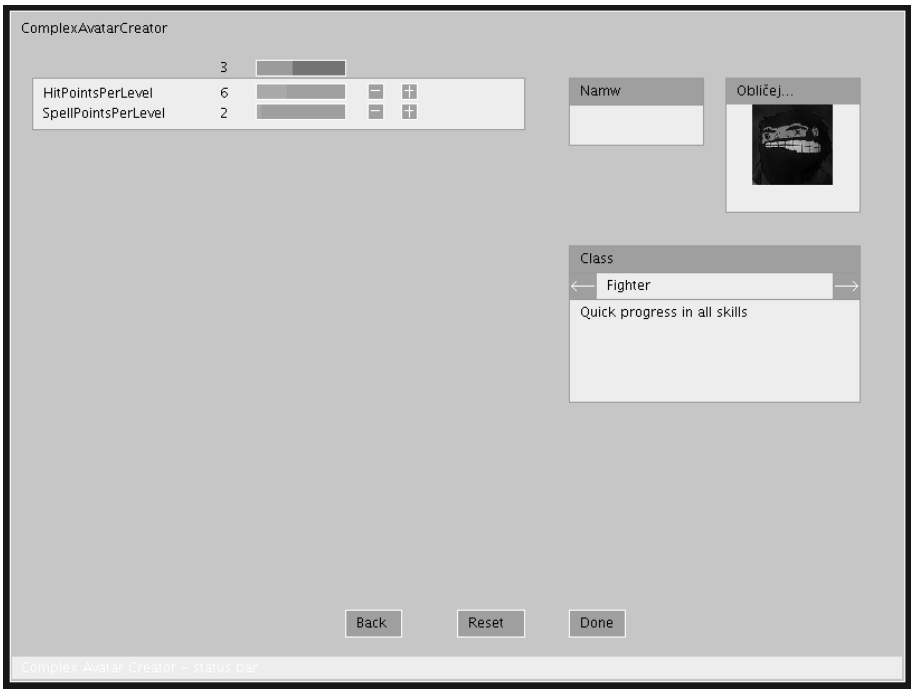
6.5.2.4 GameView

Zobrazení výřezu mapy s avatarem, nepřáteli, objekty a předměty bude ve hrách nejčastěji aktivním grafickým modulem.

V rámci realizace byla vytvořena testovací implementace "GameView2D", spolu s panely GameGUIInventoryPanelTesting a GameGUIAvatarPanelTesting. V budoucnu, po přidání možnosti konfigurovat prvky rozhraní, by se tato implementace mohla stát základem pro základní/standardní grafické herní zobrazení v systému.

Systém jako výchozí rozlišení používá aktuální rozlišení obrazovky, velikost jednotlivých panelů je pevně daná - v různých rozlišeních bude tedy samotný pohled na mapu zabírat různě velkou část obrazovky. Podle toho je přepočítána velikost zobrazovaných sektorů (v pixelech).

Na obrázku 6.11 je snímek obrazovky ze hry, na kterém je na levé straně vidět GameGUIAvatarPanelTesting, zobrazující důležité vlastnosti avatara ("energie" a čas zbývající do dalšího možného použití dovednosti) jako barevné sloupce a vypisující všechny vlastnosti a dovednosti. Na pravé straně nahoře je umístěn GameGUIInventoryPanelTesting, zobrazující sloty a backpack popsané v konfiguračním souboru. Barevně zvýrazněné jsou předměty obsahující sadu speciálních vlastností. InGameConsole je pak umístěna vpravo dole.



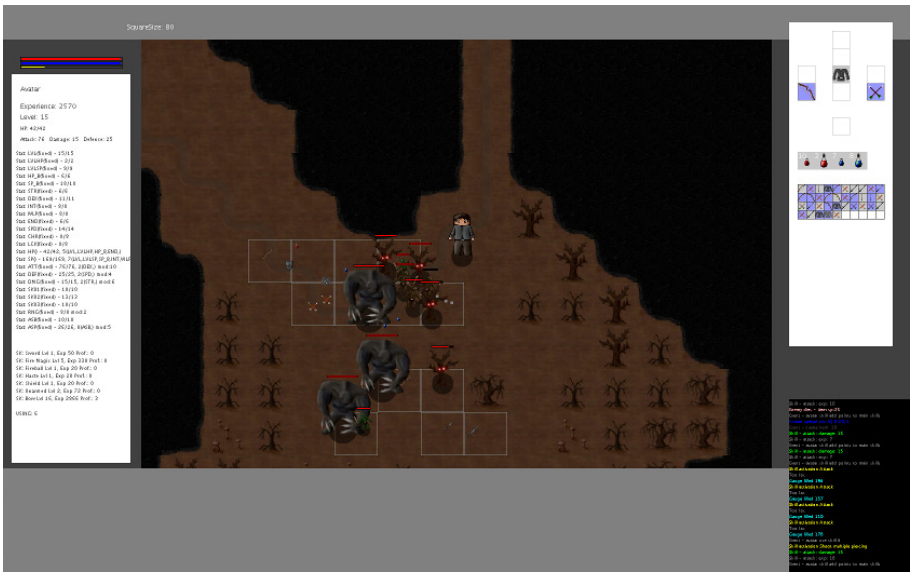
Obrázek 6.10: ComplexAvatarCreator

Zobrazení inventáře (s funkcemi pro přesuny a používání předmětů) se již dá považovat za zcela dokončené. Informační panel s údaji o avatarovi by bylo vhodné nahradit konfigurovatelnou implementací, u které by autoři sami mohli vybrat a rozmístit jednotlivé prvky - zahrnuty by byly přinejmenším:

- textová pole, například pro zobrazení jména
- rámeček se zobrazeným portrétem avatara
- výpisy hodnot vlastností (seznam s barevným odlišením vlastností ovlivněných modifikátory)
- grafické zobrazení hodnot vlastností (použití u "energií") jako barevný sloupeček

Konfigurační soubor popisující prvky uživatelského rozhraní by mohl vypadat například takto:

```
#GameView
Panels=3

Panel 0
Type="GAME_VIEW_2D"
Position="CENTER"
Elements=0

Panel 1
```



Obrázek 6.11: Zobrazení ve hře

```
Type="INFO_PANEL"
Position="LEFT"
Size:2   192 480
Elements=3

Element 0
Type="FACE"
PositionXY:2   64 16
Size:2         64 64

Element 1
Type="STATGAUGES"
PositionXY:2   64 16
Size:2         64 64
Align="HORIZONTAL"
Stats:2    13 14

Element 2
Type="STATBOX"
Stats:3    15 16 17
```

6.5.3 Správa dat

Systém grafického rozhraní zajišťuje ještě jednu funkci - načítání a uvolňování potřebných grafických dat, především textur. Navržen a implementován byl systém, pracující pouze se systémovými knihovnami Javy a 2D grafikou. Pokud budeme uvažovat o 3D grafice, bude i tento systém potřeba doplnit o práci s texturami v rámci použitých 3D knihoven.

6.5.3.1 Správa sad textur

Při práci s 2D grafikou v Javě je dostupná funkcionalita pro načítání rastrových obrázků v různých formátech (použity budou JPEG a PNG) a třída Image, spravující jeden obrázek/texturu načtenou v paměti.

Textury jsou načítány po sadách, kdy každá sada odpovídá množině souborů v určeném adresáři. Všechny soubory v adresáři musí být stejného typu, názvy obsahují pouze třímístné číslo. Číslování začíná od 000.jpg nebo 000.png a pokračuje bez omezení.

6.5.3.2 Implementace - TextureManager

Při imlementaci byla opět použita dědičnost,rodičovskou třídou je TextureManager, který přes singleton objekt poskytuje přístup ke konkrétní implementaci - jediná zatím dostupná je **TextureManagerJ2D**, spravující několik sad textur, každou spravovanou instancí třídy **TextureSetJ2D**.

Jak systém pracuje, je možné demonstrovat na konfiguračním souboru:

```
TextureSets=6

#TextureSet ID  Name         Type        Folder              TextureCount  FileTypes
TextureSet   0  "Terrain 1"  "TERRAIN"   "default_terrain"           256    "jpg"
TextureSet   1  "Terrain 2"  "TERRAIN"   "default_terrain_interior"  256    "jpg"
TextureSet   2  "Enemies"    "ENEMIES"   "default_enemies"           6      "png"
TextureSet   3  "Items"      "ITEMS"     "default_items"             16     "png"
TextureSet   4  "Avatar 1"   "AVATAR"    "default_avatar"            1      "png"
TextureSet   5  "Faces"      "FACES"     "default_faces"             2      "jpg"
```

V systému je spravováno několik kategorií sad textur ("terén", "nepřátelé", "předměty", "avatar", "obličeje"). Každá kategorie může obsahovat několik sad textur, udržuje vždy jednu aktivní/načtenou. Nejlepším příkladem je terén, pod který jsou v tomto konfiguračním souboru zařazeny první dvě sady textur. Pro každou mapu je určeno, která sada textur terénu bude nastavena jako aktivní, pokud se při přechodu mezi mapami změní, je původní sada textur uvolněna z paměti a načtena nová.

Načítání a uvolňování je implementováno ve třídě TextureSetJ2D.

Obdobně může být systém použit také pro textury nepřátel, které jsou rovněž závislé na mapě.

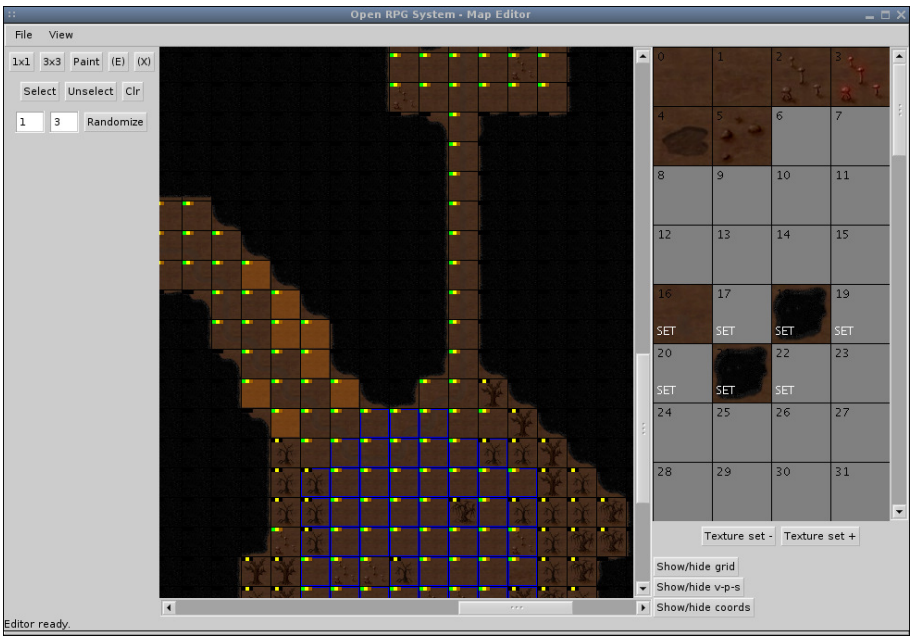
6.6 Uživatelské rozhraní pro autory her

Zbývá popsat poslední důležitou část systému - uživatelské rozhraní, pomocí kterého mohou autoři vytvářet obsah her.

Nejdůležitější aplikací z této kategorie je editor map, dále byly vytvořeny ještě dvě další drobné aplikace pro tvorbu nových projektů a jejich přepínání.

6.6.1 Editor map

Editor map je aplikace poskytující sadu grafických nástrojů pro design terénu a určení oblastí, kde se budou generovat nepřátelé.



Obrázek 6.12: Okno editoru map

6.6.1.1 Uživatelské rozhraní

Uživatelské rozhraní odpovídá běžné desktopové aplikaci, včetně menu a stavového řádku. Dále je tvořeno třemi panely:

- levý panel s výběrem nástrojů
- prostřední panel s grafickým zobrazením mapy
- pravý panel s "paletou" typů terénu

Layout je dynamický, levý a pravý panel mají pevně danou šířku, zatímco velikost prostředního panelu se mění podle velikosti okna.

Zobrazení mapy umožňuje přepínat mezi třemi úrovněmi přiblížení a dále umožňuje zapnout či vypnout zobrazení:

- souřadnic a textového popisu terénu (pro každé pole, pouze při největším přiblížení)
- vlastností terénu (průchodnost, průstřelnost)
- pomocné mřížky

"Paleta" terénů pracuje podle návaznosti textur a terénu, na sadě textur je nezávislá - navíc umožňuje mezi všemi sadami textur terénu volně přepínat.

6.6.1.2 Nástroje

Editor obsahuje několik základních nástrojů, některé ne nepodobné nástrojům z grafických editorů:

- nástroje pro "kreslení" terénu (1×1 a 3×3), pokud má daný terén navázánu "směrovou" sadu textur, jsou textury automaticky vybrány tak, aby navazovaly
- nástroj pro vyplnění ohraničené oblasti daným terénem
- nástroje pro výběr (opět jde o formu "kreslení")
- randomizér, který pro vybraná pole náhodně vybere terén z číselně daného rozsahu
- nástroje pro označení/zrušení označení daného pole jako povolené oblasti pro generování nepřátel

6.6.1.3 Poznámky k implementaci

Hlavní třídou je MapEditor, obsahující většinu uživatelského rozhraní (společně s MapEditorMapView - realizující zobrazení mapy a MapEditorPalette - realizující výběr terénu), datové struktury (množina dvojrozměrných polí) jsou obsaženy v instancích tříd MapEditorTerrain a MapEditorEnemyLayer.

MapEditorTerrain zajišťuje načítání a ukládání map, přepočet typů terénu na ID textur a obsahuje také algoritmus pro vybarvování (použit Flood fill algoritmus s vlastní frontou).

Přepočet typu terénu a ID textur pro "směrové" sady textur probíhá následovně:

- z TerrainTypeSet se určí výchozí textura daného terénu
- pokud je terén nahoře stejný, přičte se +1
- pokud je terén napravo stejný, přičte se +2
- pokud je terén dole stejný, přičte se +4
- pokud je terén nalevo stejný, přičte se +8

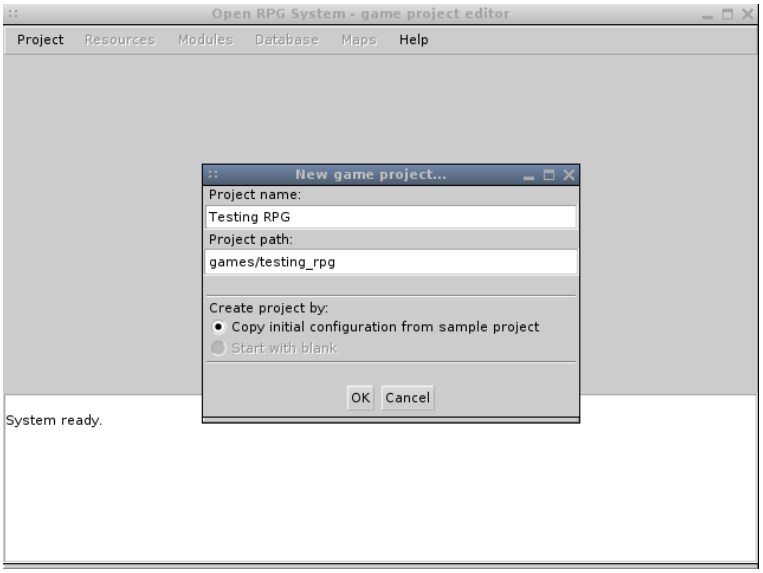
Zbývá ještě popsat formát, ve kterém jsou mapy ukládány.

6.6.1.4 Binární formát ukládání map

Formát souborů je velmi jednoduchý, do souboru je nejdříve uložena velikost mapy a následně (po sloupcích) pro všechna pole typ terénu (int), textura (int) a možnost umístit na dané pole nepřátele (boolean).

6.6.2 Editor projektů

Editor projektů zatím slouží pouze pro vytváření nových projektů a je tedy pouze alternativou k ručnímu kopírování adresářů. Rozhraní však může posloužit jako základ pro integraci dalších součástí, včetně editoru map, který již je možné z prostředí spustit.



Obrázek 6.13: Editor projektů

Nyní je již tedy možné začít systém používat jako celek, veškeré konfigurovatelné součásti enginu jsou přístupné přes konfigurační soubory, MapEditor umožňuje pohodlně tvořit herní prostředí.

Kapitola 7

Výsledky testování

Předmětem kapitoly 7 bude popis výsledků testování možností nastavení systému, jeho efektivity a funkčnosti na odlišných platformách.

Pro otestování samotné implementace a různých možností nastavení je nutné vytvořit příslušná nastavení her.

7.1 Možnosti systému

Pro testování byla v systému za použití implementovaných verzí jednotlivých modulů vytvořena základní nastavení dvou her, využívající v prvním případě minimum, ve druhém maximum možností systémem nabízených, obvykle s výběrem odlišných nastavení jednotlivých součástí systému, tak, aby byla otestována co nejširší škála implementované funkcionality.

Pro každou hru vznikla také základní sada dat, obsahující popis několika nepřátel, předmětů a dvě mapy.

Nejdříve se podívejme, co vše budou mít obě hry společné - půjde pravděpodobně o velkou množinu vlastností, především kvůli tomu, že i přes možnost přepínat různé verze modulů máme v tuto chvíli u každého pouze jednu implementaci.

- výběr modulů ComplexRPGSystem (a rozhraní pro tvorbu postavy ComplexAvatarCreator), GameView2D, EnemyAIBasic
- souboje v reálném čase
- ovládání dovedností se zlepšuje jejich používáním
- stejné textury terénu

Obě verze teď budou popsány s pomocí ukázek konfiguračních souborů a snímků obrazovky, na přiloženém CD je také k dispozici několik videí, ukazujících průběh každé hry a (z demonstračních účelů velice zrychleně) vývoj hodnot v rámci RPG systému, především úrovní dovedností. Dále videa předvádějí inventář a operace s předměty.

7.1.1 Vzorová hra A - jednoduché řešení

Prvním příkladem bude hra s jednoduchým PRG systémem, více zaměřená na akční pasáže. U RPG her je všeobecně možné vypozorovat určitou závislost - čím rychlejší průběh hry a čím více akce, tím jednodušší bývají pravidla v rámci RPG systémů. Naopak hry se složitými systémy dovedností mají často soubojový systém probíhající na kola, aby měl hráč dost času naplánovat další akce avatara.

7.1.1.1 Komentované konfigurační soubory

Nastavení systému a jednotlivých modulů je vidět z následujících klíčových konfiguračních souborů:

Globální nastavení systému

V konfiguračním souboru třídy GMath je vybrán nejjednodušší vzorec pro výpočet výsledků soubojů (operující pouze se dvěma hodnotami) a nastaveny poměrně vysoké koeficienty rychlostí postav - svůj význam tak budou mít i rychlé reakce hráče.

```
#Item database
#AttackFormula
#=============
# 1 COMBAT_FORMULA_ATTACK_DEFENSE
# 2 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE
# 3 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_MIN_MAX
# 4 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_ERENIA
AttackFormula=1
CombatCoef=2
...

#Speed
#===========
# Overall movement speed
BaseSpeed=40
SpeedFactor=30
```

Předměty

Ve vzorové hře budou pouze dvě kategorie předmětů - střelné zbraně, rozdělené do tří tříd, a "lékárničky". Speciální vlastnosti nebudou využity.

```
#Item database
#=============

UserSlots=1
UserSlot 0 "Medkits" "P1" "true"

# Categories
#===========
ItemCategories=2
```

```
ItemCategory 0
Name="Weapon"
Stats:2 7 10
OverrideStats:0
Slots="RH"
UserSlots:0
Consumable="false"
Ammo=-1


ItemCategory 1
Name="Medkit"
Stats:1 6
OverrideStats:0
Slots="NONE"
UserSlots:1 0
Consumable="true"
Ammo=-1


# Sorting
# ItemClass ID Name Category
#==========================
ItemClasses=4
ItemClass 0 "Gun" 0
ItemClass 1 "Railgun" 0
ItemClass 2 "Shotgun" 0
ItemClass 3 "Medkit" 1

# Special Sets
#=============
SpecialSets=0


# The item list
# (Item ID Class Quality Name StatValues)
#=======================================
Items=7
Item 0 0 2 "Gun 1" 0 12
Item 1 0 4 "Gun 2" 2 16

Item 2 1 6 "Railgun 1" 2 18
Item 3 1 8 "Railgun 2" 4 26

Item 4 2 2 "Shotgun 1" 6 8
Item 5 2 4 "Shotgun 2" 8 10

Item 6 3 0 "Medkit" 25
```

Dovednosti avatara

Avatar bude ovládat pouze čtyři dovednosti - tři odpovídající určitým druhům zbraní a jednu pro boj beze zbraně. Struktura jejich popisu bude shodná, každá bude obsahovat pouze jeden stupeň (Proficiency) a jedinou referenci na konkrétní funkcionalitu (Skill).

```
#SKILLS
#======

SkillWrappers=4

#Skill          id   abbr     name
#-------------------------------------------
SkillWrapper     0   "GUN"     "Gun"

SkillsBy="Proficiency"
Parent=-1
Element=0
Passive="False"
ItemClasses:1   0

ProficiencyLevels=1

#Proficiency   id    abbr     name      level      stat   special     automatic    price
Proficiency     0    "NV"   "Novice"        1     "ATT"        -1        "true"        1

#Subskills      levels-when-selection
Skills:1        0

#Special
Skill 0
Name="Attack"
Type="Shoot"
Percentages:0
...
```

Nepřátelé

Struktura konfiguračního souboru popisujícího nepřátele se mezi jednotlivými hrami bude lišit jen minimálně. Zde jsou definovány tři druhy nepřátel, poslední dva jsou schopné útočit na dálku. Poslední, nejsilnější, se nebude objevovat v první mapě.

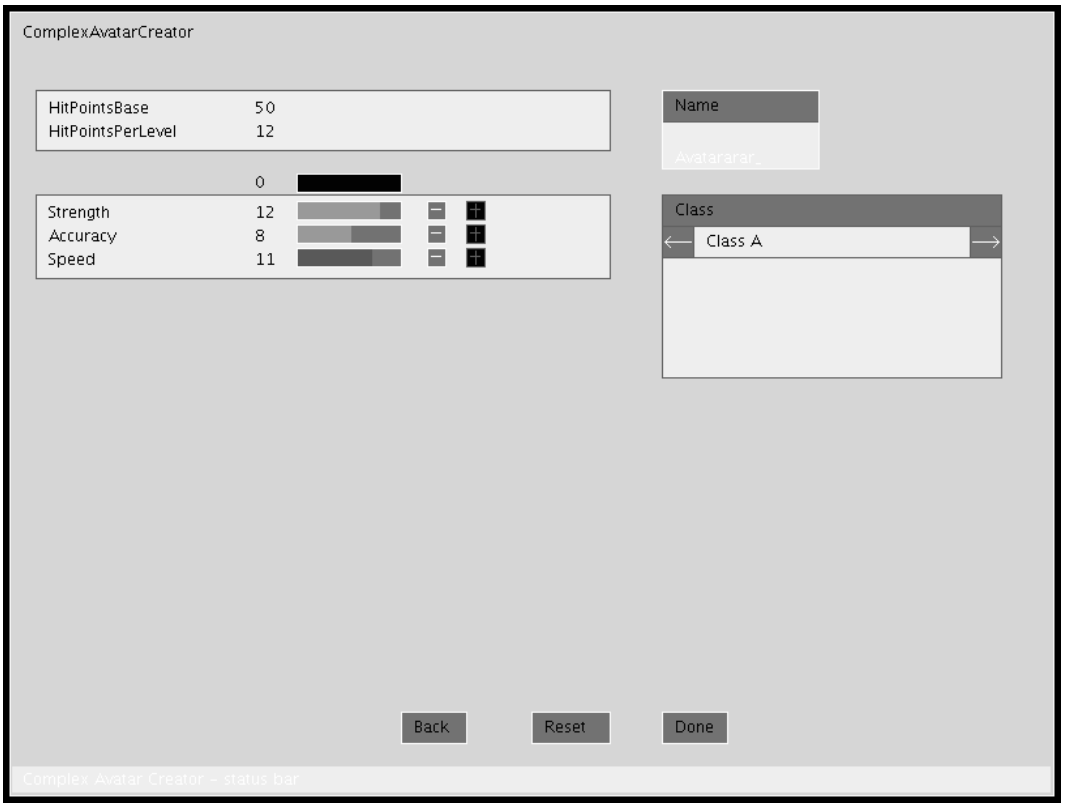
```
Enemies=3

#            LVL HP   SP    Att Def Dmg  spdA spdR   size range exp money
Enemy 0 "A"  1   20   0     1   4   2    4    16      30   10    1  0
Skills:1 0
SkillLevels:1 1
Items:4 0 6 6 6
ItemProbability=33
ItemSpecialProbability=0

#           LVL HP   SP    Att Def Dmg  spdA spdR   size range exp money
Enemy 1 "B" 2   40   0     2   8   4    2    14       40   220    2  0
Skills:1 2
SkillLevels:1 2
Items:4 0 1 2 6
ItemProbability=24
ItemSpecialProbability=0

#Ranged    LVL HP   SP    Att Def Dmg  spdA spdR   size range exp money
```

```
Enemy 2 "C" 8   200    0    3     18  4    3    12     50   360   4  0
Skills:2 2 3
SkillLevels:3 1 1 1
Items:4 2 3 4 5
ItemProbability=20
ItemSpecialProbability=0
```

7.1.1.2 Komentované snímky obrazovky

Tvorba postavy

Tvorba postavy je velmi jednoduchá, spočívá pouze ve vyplnění jména, volbě povolání a nastavení tří vlastností.



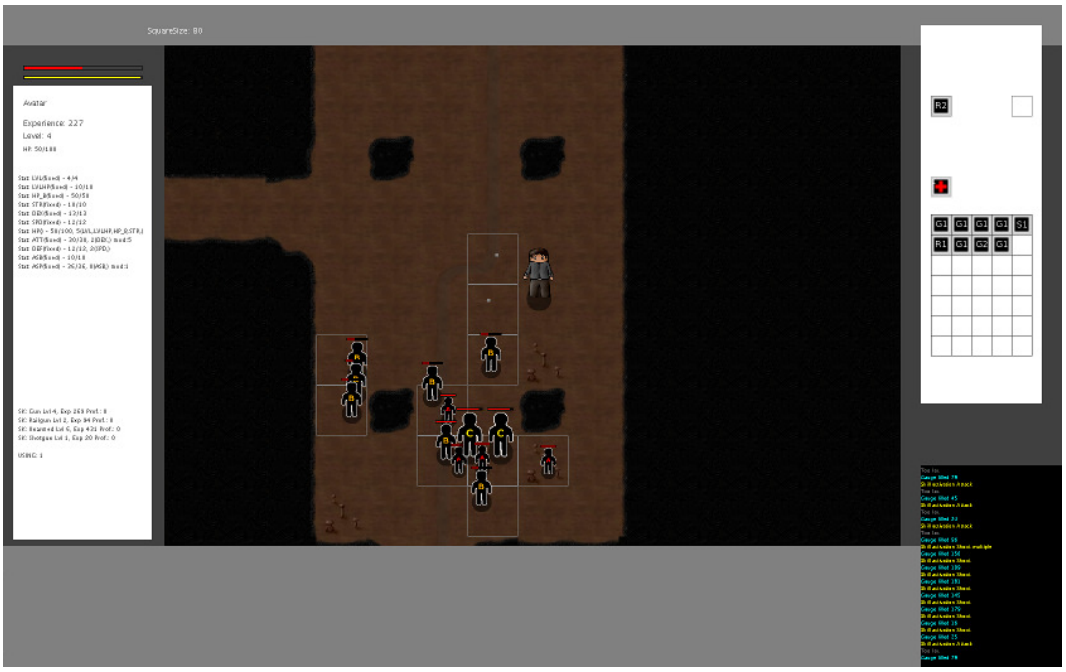
Obrázek 7.1: ComplexAvatarCreator

GameView

Zobrazení ve hře ukazuje jednoduchou sadu vlastností a minimalisticky rozvržený inventář.

Jako první příklad byl tedy vytvořen základ hry, která by mohla být zařazena jako "akční hra s RPG prvky", avatar se postupně zlepšuje v ovládání zbraní, které používá.

Vhodným prostředím pro podobnou hru by mohl být svět zapadající do žánru sci-fi.



Obrázek 7.2: Hra A - GameView

7.1.2 Vzorová hra B - ukázka komplexnosti systému

S mnohem komplikovanějším nastavením jednotlivých modulů ve "Hře B" jsme se již setkávali v podobě příkladů uváděných v průběhu práce, až na vyjímku - tvorbu postavy, kde byl uveden jednodušší příklad. "Hra B" bude používat názvosloví typické pro fantasy prostředí.

7.1.2.1 Komentované konfigurační soubory

Globální nastavení systému

Pro výpočet úspěšnosti útoků je použit sofistikovanější vzorec se třemi vstupními parametry, rychlosti pohybu postav jsou výrazně nižší v porovnání s předchozí hrou.

```
#Item database
#AttackFormula
#=============
# 1 COMBAT_FORMULA_ATTACK_DEFENSE
# 2 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE
# 3 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_MIN_MAX
# 4 COMBAT_FORMULA_ATTACK_DEFENSE_DAMAGE_ERENIA
AttackFormula=4
CombatCoef=2
...

#Speed
#===========
```

```
# Overall movement speed
BaseSpeed=10
SpeedFactor=20
```

Předměty a inventář

V systému předmětů je definováno mnoho druhů vybavení a jednorázově použitelných předmětů, tomu odpovídá i množství specifických slotů v inventáři. Avatarovo "zavazadlo" je rovněž větší, s prostorem na 80 předmětů. Také bylo definováno několik sad speciálních vlastností.

```
UserSlots=11
UserSlot 0 "Boots" "BTS" "false"
UserSlot 1 "Belt" "BLT" "false"
UserSlot 2 "Amulet" "AMT" "false"
UserSlot 3 "RingL" "RL" "false"
UserSlot 4 "RingR" "RR" "false"
UserSlot 5 "Body" "BD" "false"
UserSlot 6 "Head" "HD" "false"
UserSlot 7 "Small healing potions" "P1" "true"
UserSlot 8 "Grand healing potions" "P2" "true"
UserSlot 9 "Small mana potions" "P3" "true"
UserSlot 10 "Grand mana potions" "P4" "true"
...

# Inventory
#==========

#GUI settings
TextureSize=32
TextureSizeBackpack=16
InventoryPosition:2 16 16

BackpackSize:2 10 8
BackpackPosition:2 16 300
```

Dovednosti avatara

V příkladech v kapitole 6 již bylo ukázáno, že dovednosti mohou obsahovat několik stupňů (Proficiency), aktivovat několik různých funkcionalit a vzájemně na sebe navazovat. Při delším používání některé dovednosti se tak nejen zlepšují číselné hodnoty vlastností, ale v rámci jednotlivých dovedností se otvírají další možnosti - jedním z příkladů by mohla být střelba z luku, u které avatar od druhého stupně střílí průrazné šípy a od třetího dokonce několik střel najednou:

```
...

#Subskills      levels by prof.
Skills:3        0 0 0

#Special
Skill 0
```

```
Name="Shoot"
Type="Shoot"

Skill 1
Name="Piercing arrow"
Type="Shoot piercing"

Skill 2
Name="Double arrow"
Type="Shoot multiple"
```

Na příkladu dovednosti pro boj beze zbraně si ještě ukážeme, jak je s pomocí stupňů (a tedy ovlivněných vlastností) možné vyvážit dovednosti, používané v kombinaci s modifikátory z předmětů (například u boje s mečem), s dovedností, která je v tomto směru znevýhodněna.

Pro vyřešení této situace bylo u dovednosti pro boj beze zbraně definováno 5 stupňů, tedy o 2 více, než u ostatních dovedností - v případě dosažení nejvyššího stupně je pak úroveň dovednosti přičtena ke třem vlastnostem, ke dvěma dokonce dvakrát (ATT, DMG).

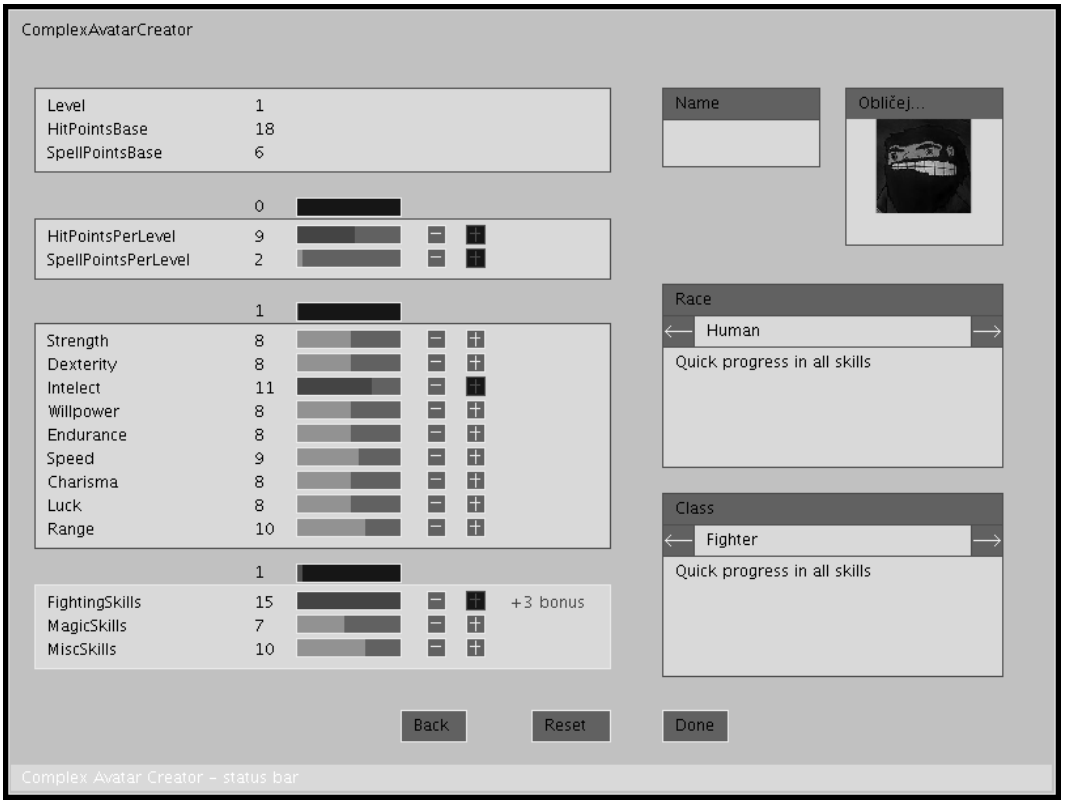
```
#Skill          id   abbr     name
#-------------------------------------------
SkillWrapper     5   "UNA"     "Unarmed"

SkillsBy="Selection"
Parent=-1
Element=0
Passive="False"
ItemClasses:1   -1

ProficiencyLevels=5

#Proficiency   id    abbr   name           level    stat   special    automatic    price
Proficiency     0    "NV"   "Novice"         1     "ATT"        -1      "true"     0
Proficiency     1    "EX"   "Expert"         3     "DMG"        -1      "true"     0
Proficiency     2    "MS"   "Master"         6     "ASP"        -1      "true"     0
Proficiency     3    "GM"   "Grandmaster"    9     "DMG"        -1      "true"     0
Proficiency     4    "GM2"  "Grandmaster2"  12     "ATT"        -1      "true"     0

#Subskills      levels-when-selection
Skills:4        1 6 12 18

#Special
Skill 0
Name="Attack"
Type="Double Attack"
...
```

7.1.2.2 Komentované snímky obrazovky

Tvorba postavy

Tvorba postavy ve "Hře B" umožňuje nezávisle na sobě vybrat nebo nastavit:

- poměr "energií", tedy vyvážit, zda postava v boji více vydrží či bude mít více energie na sesílání kouzel
- vyvážit jednotlivé základní vlastnosti avatara (síla, rychlost, odolnost, intelekt, ...)
- vyvážit předpoklady pro ovládání různých druhů dovedností (ve hře bude ovlivňovat to, jak rychle se bude v daných kategoriích dovedností postava zlepšovat)
- rasu - třídu postavy ovlivňující základní vlastnosti
- povolání - třídu postavy ovlivňující předpoklady pro dovednosti a poměr "energií"



Obrázek 7.3: ComplexAvatarCreator

Hra B je tedy příkladem nastavení s velmi komplikovanými pravidly, která na druhou stranu příliš nezatěžují hráče, protože velká část funkcionality RPG systému je automatická.

V další podkapitole následuje rozbor výpočetní náročnosti systému v reálných podmínkách.

7.2 Testování výkonu

V ideálním případě by měl systém bez problémů běžet i na starších počítačích s jednojádrovými procesory s frekvencemi pod 1 GHz.

Obrázek 7.4: Hra B - GameView

Volba pevně dané frekvence hodin zaručuje, že běžící aplikace nebude zbytečně zatěžovat procesor, pokud bude počítač disponovat přebytkem výpočetního výkonu. Naopak, pokud by byl procesor zcela vytížen, dojde pouze ke zpomalení frekvence průchodů hlavní a grafickou smyčkou.

U důležitých části systému má smysl analyzovat jejich vliv na celkový výkon.

7.2.1 Efektivita implementace hledání cest

Může být implementovaný algoritmus A* pro hledání cest v mapě problémem z hlediska výkonu?

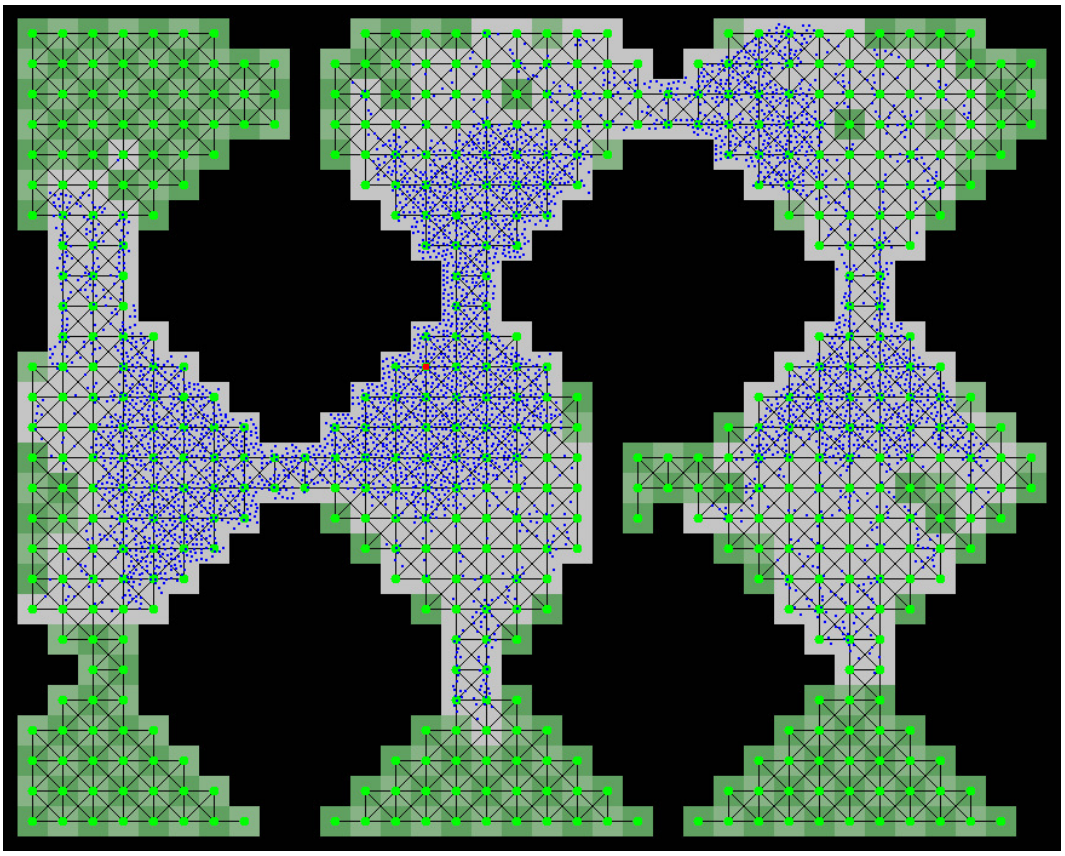
Při implementaci bylo provedeno několik testů, nejrozsáhlejší spočíval v umístění 5000 aktivních nepřátel do mapy a ponechání maximální hloubky prohledávání na hodnotě 100. Přibližný čas výpočtu pro všechny nepřátele byl přibližně jedna vteřina.

Již z tohoto údaje vyplývá, že efektivita implementovaného algoritmu je zcela dostačující. Reálné maximální množství aktivních nepřátel bude v řádu desítek (nezapomeňme, že nepřátelé se deaktivují, pokud se od nich avatar dostatečně vzdálí), přepočítávání cesty pro jednoho nepřítele probíhá navíc v průměru pouze jednou za 15 průchodů hlavní smyčkou.

Pro přehled přikládám ještě výpis nastavení některých parametrů hledání cest/chování nepřátel, vzešlého z dlouhodobého ladění systému:

- Omezení hloubky prohledávání: 100 stavů
- Frekvence přepočítávání cesty: jednou za 10 - 20 (náhodně) průchodů hlavní smyčkou

- Počet kroků zpět pro vyřešení blokace: 5 - 10 (náhodně) průchodů hlavní smyčkou
- Váha uzlů nad sektory s nepřáteli: $1 + 2 \times \sum V_i$, kde V_i jsou velikosti jednotlivých nepřátel
- Vzdálenost od dalšího uzlu předpočítané cesty pro přesun ukazatele na tento uzel: 1,5



Obrázek 7.5: Hledání cest - mapa s 5000 nepřáteli

7.2.2 Efektivita implementace grafiky

Jednoznačným úzkým hrdlem aplikace se nakonec stala realizace grafického zobrazení v enginu - především kvůli jedné konkrétní funkci implementované v grafických knihovnách Javy - vykreslování obrázků s vícebitovou alfou, především v kombinaci se změnou jejich velikosti.

Ve hře se jedná především o nepřátele a předměty rozmístěné v mapě a také umístěné v inventáři (ve stávající implementaci uživatelského rozhraní je inventář zobrazen po celou dobu zobrazení GameView) - s rostoucím počtem předmětů na obrazovce roste vytížení procesoru, v následujících podmínkách:

- OS Linux x64

- rozlišení 1680 x 1050, 60 FPS
- dvoujádrový procesor s frekvencí sníženou na 1GHz

bylo změřeno vytížení procesoru běžící aplikací, v závislosti na počtu zobrazených částečně průhledných obrázků dosahovaly hodnoty přibližně:

- 18% při zobrazení rozhraní pro tvorbu postavy
- 30% při zobrazení mapy s avatarem (bez nepřátel a předmětů)
- 55% pro 50 předmětů a 25 nepřátel zobrazovaných v každém snímku

Pro vytížení $> 50\%$ je pak již subjektivně znát jisté zpomalení.

Jelikož hlavní příčinou problému je velký počet najednou zobrazovaných předmětů, je s ním spojené i možné řešení: samostatné zobrazení inventáře (zde již mírné snížení snímkové frekvence nijak vadit nebude) a vhodné nastavení množství předmětů generovaných poraženými nepřáteli.

V podmínkách s menším množstvím předmětů se již blížíme popsané ideální situaci, je třeba také vzít v úvahu, že u starších počítačů je možné očekávat menší rozlišení a s ním spojené nižší nároky na procesor.

Při vytváření spustitelného programu pro přiložené CD byl objeven další problém - zatímco při spuštění aplikace z vývojářského prostředí (konkrétně Eclipse) běžel systém zcela plynule (viz. výše), po exportu do spustitelného JAR archivu prudce vzrostlo zatížení procesoru a byl znát výrazný výkonový propad. Pomocí profilingu byla jako hlavní zdroj zátěže identifikována opět metoda knihoven Javy pro vykreslování obrázků - drawImage(). Zajímavým zjištěním bylo, že tento problém byl multiplatformní - projevoval se v Linuxu i ve Windows.

Příčina byla nakonec objevena - chybějící parametr "-Dsun.java2d.opengl=True", předávaný virtuálnímu stroji (v nastavení IDE nebo v dávkovém souboru) pro vynucení vykreslování 2D grafiky s pomocí OpenGL.

7.3 Platformy

Systém byl otestován pod různými operačními systémy a s různými implementacemi Javy:

- Linux x64, OpenJDK 1.6
- Linux x64, Sun Java 1.6.0_17
- Linux x86, OpenJDK 1.6
- Windows Vista x86, Sun Java 1.6.0_17

Na všech testovaných platformách je systém zcela funkční, bez nutných specifických úprav. Běží bez problémů (i když výrazně pomaleji) i ve virtualizovaném prostředí.

Kapitola 8

Zpětné srovnání s existujícími řešeními

Podle zadání byla vytvořena kostra systému se základní či testovací a v některých případech i poměrně pokročilou implementací jednotlivých částí.

Motivací bylo poskytnout systém, který z hledisek otevřenosti a konfigurovatelnosti překoná dostupné alternativy.

I když by implementovaný systém spadal spíše mezi "open-source řešení v různé fázi vývoje", smysl má srovnávat pouze s aplikací RPG Maker, kterou je jako jedinou možné použít jako vzor a příkad hotového a dobře navrženého systému.

8.1 Implementovaný systém a RPG Maker

Připomeňme si kritéria, podle kterých jsem se v úvodu rozhodoval, zda má smysl využívat existující alternativy, nebo bude-li přínosné navrhnout a implementovat vlastní řešení:

- software dostupný zdarma (pokud možno open-source)
- není po technické stránce zcela zastaralý
- všechny důležité aspekty herního systému/pravidel jdou změnit
- dostupný hráčům na všech hlavních platformách (Windows, Linux, Mac OS X)
- umožňuje vytvářet hry v češtině,

rovnou je možné definovat další kritéria, podle kterých bude možné systémy srovnat:

- otevřenost
- kvalita uživatelského rozhraní pro autory

Výsledkem úvodní analýzy bylo, že žádná dostupná aplikace není zcela vhodná - pojďme ale porovnat RPG Maker a implementovaný systém bod po bodu.

8.1.1 Dostupnost

RPG Maker je placený komerční produkt, cena je sice nízká, ale už samotná nutnost platby (navíc v dolarech) může být překážkou.

U implementovaného systému je zatím toto kritérium irelevantní, do budoucna by vydání pod některou z open-source licencí mohlo usnadnit zapojení dalších účastníků do projektu, ať již programátorů nebo grafiků. I uživatelská komunita bývá v podobných projektech velmi cenná.

8.1.2 Technická stránka

Za zcela zastaralé zpracování by se dala považovat například grafika v rozlišení 320×200 pixelů - u enginu aplikace RPG Maker není rozlišení výrazně vyšší, postavy jsou vždy umístěny na jediném poli mřížky a grafické zobrazení soubojů je z velké části statické - přesto je obojí důsledkem (pravděpodobně dlouho vyvažovaného) kompromisu mezi vizuálně přitažlivou a efektní grafickou stránkou a usnadněním práce autorům, včetně vytváření sad textur. I přes všechny vyjmenované vlastnosti působí grafické rozhraní (subjektivně) příjemně a svůj účel plní. Jakékoliv srovnání technické stánky her vytvořených v systému RPG Maker s komerčními hrami nemá smysl.

U implementovaného systému je situace srovnatelná - rozlišení je sice neomezené, pohyb postav plynulý, ale opět platí podobné principy a bylo nutné učinit některé kompromisy.

Velký rozdíl je v uživatelském rozhraní, zatímco u RPG Makeru je prakticky vše řešeno přes několik menu ovládaných klávesnicí, použití myši v implementovaném systému usnadňuje mnoho činností, ať již jde o přemisťování předmětů v inventáři nebo používání dovedností.

8.1.3 Konfigurovatelnost

Nastavení pravidel hry je u programu RPG Maker omezené, je možné měnit mnoho parametrů a upravovat, přidávat a mazat položky v databázi (předměty, nepřátele, dovednosti), základní principy jsou ale pevně dané, autor tedy nemůže ovlivnit například, jaké vlastnosti budou popisovat postavu a jaký bude jejich význam v logice hry.

V realizovaném systému je možné pravidla nastavit do mnohem větší míry, především v rámci implementovaného modulu RPG systému, v mnoha případech je možné zvolit také různé vzorce pro stejné výpočty. Možnost v budoucnosti doplnit zcela odlišné implementace některých součástí přidávají z pohledu konfigurovatelnosti ještě další rovinu.

8.1.4 Multiplatformnost

RPG Maker je vydáván pouze pro operační systém Windows, propojení je zvýrazněné i odvozováním jmen verzí aplikace od pojmenování verzí Windows. Předposlední verze (XP) byla s využitím aplikačního rozhraní WINE[10] s jistými omezeními použitelná i v linuxových operačních systémech, u poslední verze (VX) již toto řešení není prakticky možné.

Implementovaný systém je napsán čistě v Javě, čímž by měl být zajištěn bezproblémový běh na všech platformách, pro které je instalace Javy dostupná - pro OS Windows a Linux byl tento předpoklad otestován.

8.1.5 Možnosti lokalizace

RPG Maker disponuje uživatelským rozhraním pro vyplnění různých klíčových slov používaných ve hře, systém je tedy možné do velké míry lokalizovat. S češtinou by však autor narazil, některé znaky se vůbec nezobrazí.

Systém vytvořený v rámci práce má navržen jednoduchý systém lokalizace, který vzhledem k použití kódování znaků Unicode (UTF-8) podobnými problémy trpět nebude.

8.1.6 Otevřenost

RPG Maker je uzavřený systém s proprietárním formátem pro ukládání dat.

Implementovaný systém ukládá veškerá data do textových souborů nebo do binárních souborů s popsanou strukturou.

Pozn.: samotná uzavřenost nemusí být u her všeobecně na škodu, je tak možné zabránit hráčům pozměňovat pravidla a usnadnit si například některé úkoly, před kterými ve hře stojí - tedy de facto podvádět.

Na druhou stranu, u systému, kde je dostupný editor obsahu, mají hráči možnost udělat stejné zásahy, navíc v uživatelsky příjemném prostředí. Uzavřený formát souborů zde tedy nemá zjevné výhody.

8.1.7 Uživatelské rozhraní pro autory her

Uživatelské rozhraní pro autory her je nejsilnější stránkou RPG Makeru, umožňuje upravovat veškeré záznamy v databázi, tvořit mapy, propojovat události a poskytuje další nástroje pro specifické úkony - například automatický generátor podzemních prostor.

Implementovaný systém zatím ve většině případů spoléhá na ruční úpravy konfiguračních souborů, pomocí kterých je možné nastavit vše, co je v systému podporováno.

Výhody grafického rozhraní zde ale nejde upřít, především pro omezení možných chyb a zamezení nutnosti jejich odladění na běžící aplikaci. Možnosti doplnění dalších grafických nástrojů budou zmíněny v následující kapitole.

Samostatným tématem je editor map realizovaného systému, který je zcela použitelný, uživatelsky přívětivý a poskytuje některé praktické nástroje a vnitřní informace, které naopak chybí v editoru map RPG Makeru.

8.1.8 Shrnutí

Vzhledem k daným požadavkům, z části zaměřeným na to, aby autoři mohli pochopit vnitřní principy systému (zde je možné zdůraznit i jistou edukativní hodnotu), zčásti zaměřeným na zcela praktické aspekty, vychází ze srovnání implementovaný systém výrazně lépe.

8.2 Zhodnocení vlastního přínosu práce

V rámci práce byl tedy vytvořen funkční základ systému, který je v daném (i když nepříliš rozsáhlém) žánru aplikací pro tvorbu RPG her vyjímečný svou koncepcí otevřenosti a rozsáhlé konfigurovatelnosti.

Z implementovaných modulů si zvláštní pozornost zaslouží RPG systém, který umožňuje téměř bez omezení konfigurovat veškerou funkcionalitu, včetně volby jednoho z více rozdílných principů nebo použití různých vzorců v různých situacích. Takovéto možnosti opět žádná z dostupných aplikací ani zdaleka neposkytuje, pravidla RPG systémů jsou obvykle pevně daná.

Je zajímavé, že některé programy pro tvorbu her (spíše univerzálního zaměření, jako například aplikace Game Maker) vznikly na akademické půdě a je možné je úspěšně využívat na školách při výuce informatiky[11].

Realizovaný systém sice nemá podobné ambice, přesto práce s jeho otevřenou strukturou může být zajímavou zkušeností a nenásilným zdrojem poznatků o fungování her a programů obecně, řešení různých problémů při tvorbě her pak vyžaduje podobné logické uvažování a myšlenkové přístupy jako řešení informatických problémů obecně.

Kapitola 9

Zhodnocení splnění cílů práce

Podle zadání bylo úkolem vytvořit kostru systému, implementovat základní součásti a uživatelské rozhraní pro tvorbu her.

Vytvořený systém má nyní rozsah přes 20 000 řádek zdrojových kódů a okolo tisíce řádek konfiguračních souborů pro každé nastavení konkrétní hry, přesto je implementace stále na úrovni "kostry a základních modulů". Záměrem autora je v budoucnu v implementaci dále pokračovat, současné řešení je - i díky zpracování v rámci diplomové práce - pečlivě navržené, obsahuje všechny klíčové součásti a je dobrým výchozím bodem pro doplňování další funkcionality.

Do jaké míry ale byly splněny jednotlivé požadavky zadání?

9.1 Vyhodnocení splnění požadavků zadání

Zadání a splnění jednotlivých si projdeme postupně.

- *Navrhněte a implementujte kostru systému umožňujícího tvorbu různorodých RPG her,*

  Splněno, kostra byla vytvořena a i zpětně se návrh jeví jako dostatečně robustní (i z pohledu možnosti dalšího pokračování). Variabilita byla průběžně zohledňována jako jedno z hlavních kritérií.

- *který bude použitelný i pro autory bez znalosti programování.*

  Splněno, autoři se setkávají s grafickým editorem map a konfiguračními soubory s jednoduchou a intuitivně navrženou strukturou, záměrem je doplnit také další grafické nástroje pro práci s daty.

- *Aplikaci budou tvořit dvě hlavní části - herní engine a uživatelské rozhraní pro tvorbu obsahu her.*

  Splněno, větší důraz byl kladen na engine, ale také editor map na straně uživatelského rozhraní pro autory je zcela funkční. Doplnění dalších nástrojů bude rozebráno v následující podkapitole, při jasně definované struktuře konfiguračních souborů, do kterých jsou data ukládána, jde ale již o triviální problém.

- *Při návrhu kostry systému se zaměřte na variabilitu, autorům her bude umožněno široce nastavovat engine - včetně možnosti výběru různých implementací některých součástí (např. RPG systém, umělá inteligence, grafika), systém bude umožňovat snadné doplnění dalších implementací. U každého z těchto modulů implementujte alespoň jednu verzi s přinejmenším základní funkčnosti.*

  Splněno. Všechny moduly byly implementovány, u zmíněných je s využitím dědičnosti realizována možnost vytvořit více různých implementací.

- *Navrhněte efektivní a otevřenou strukturu datových souborů vytvářených her (celková nastavení, mapy, předměty, NPC apod.).*

  Splněno, veškerá data her jsou uložena buďto v podobě textových souborů nebo binárních souborů s popsanou strukturou. Textový formát byl navržen s důrazem na intuitivitu a úspornost.

- *Pro implementaci použijte jazyk Java, zvažte použití specializovaných knihoven pro jednotlivé součásti systému (grafika, UI).*

  Splněno - možné použití specializovaných knihoven je rozebráno v následující podkapitole.

9.1.1 Shrnutí

Všechny body zadání byly splněny, kostra systému i implementované verze jeho součástí jsou zcela funkční. Vzhledem k rozsahu programu však bude možné dále pokračovat v práci na mnoha dalších částech.

9.2 Možnosti dalšího pokračování práce

Následující podkapitola stručně nastíní další plánovanou práci na systému.

Některá témata již byla zmíněna v textu:

- souboje na kola
- NPC postavy

Dalšími tématy budou:

9.2.1 Více postav ovládaných hráčem

Jediná postava je typická spíše pro RPG hry zaměřené více na akci, v taktičtějších hrách hráči většinou ovládají více postav, obvykle 3 - 6.

Samotné přidání více postav by bylo relativně jednoduché - vytvoření třídy Party s polem instancí třídy Avatar, rozhraní pro tvorbu postavy při startu hry by dovolilo postupně nezávisle vytvořit všechny postavy.

Komplikovanější by byla situace na straně soubojů a grafiky. V RPG hrách s grafikou z pohledu první osoby je situace jednodušší - postavy jsou reprezentovány pouze svým portrétem, v systému je udržovaná jediná pozice. V implementovaném systému by musely být všechny postavy přítomné na mapě. Souboje by pak pravděpodobně musely probíhat na kola, aby byl hráč vůbec schopen všechny postavy ovládat. Další nutnou modifikací by byla umělá inteligence nepřátel, především kvůli potřebě určit, na kterou z postav útočit.

9.2.2 Pokročilejší umělá inteligence

Stávající implementace umělé inteligence se jeví jako dostatečná, použití specifických knihoven nebo pokročilejších algoritmů tedy nebude dále zvažováno.

Naopak zajímavou možností bude napojit některé nepřátele na více specifických událostí ve hře a popsat jejich chování komplexnějším konečným automatem s přechody definovanými pomocí jednoduchého skriptovacího systému.

9.2.3 Quests - úkoly jako motivační prostředek

Hlavním motivačním prostředkem a také prostředkem pro vyprávění příběhu jsou úkoly, které avatarovi zadávají různé postavy herního světa. Za jejich splnění je avatar (a jeho prostřednictvím také hráč) odměňován zkušenostmi, penězi nebo novými předměty.

V realizaci půjde o datovou strukturu, obsahující popis několika fází určených různými událostmi:

- rozhovor s danou vedlejší postavou
- poražení daného nepřítele
- získání nějakého předmětu

Bude tak možné vytvořit například dějovou linii, ve které vedlejší postava požádá avatara o nalezení ztraceného předmětu, série rozhovorů s dalšími NPC ho přivede na stopu, po porážce silného nepřítele předmět získá a po odevzdání první vedlejší postavě se dočká odměny.

9.2.4 Grafický editor RPG systému

Sada grafických nástrojů byla při srovnání systémů uvedena jako silná stránka programu RPG Maker, přinejmenším pro RPG systém (tedy vlastnosti, dovednosti, předměty) bude v budoucnu vhodné vytvořit grafické uživatelské rozhraní, ve kterém bude možné vyplnit všechny hodnoty, přidávat/měnit/mazat položky a především určit návaznosti položek mezi sebou. Programově pak půjde ošetřit, jak se vypořádat s chybami vzniklými například po smazání položky, na kterou již existují nějaké reference.

Při přímých úpravách konfiguračních souborů musí autor vše hlídat sám a velice snadno se dopustí chyby.

9.2.5 3D grafika

Současné 2D grafické rozhraní ve hrách by se mohlo stát výchozím zobrazením, především díky snadnému použití a přehlednému zobrazení.

3D grafika může systém výrazně zatraktivnit, přesto však bude muset stavět na stejném základu, tedy terénu omezeném čtvercovou mřížkou a spritech/billboardingu pro zobrazení většiny objektů ve hře, hlavním přínosem pak budou pravděpodobně různé speciální efekty a práce se světlem.

Proto se jeví jako zbytečné použít některý z kompletních enginů (jako OGRE 3D[6] nebo jMonkeyEngine[4]). I kvůli potřebě zapojení 3D zobrazení do již implementovaného uživatelského rozhraní napsaného s pomocí grafických knihoven Javy se jako nejlepší možnost jeví použití JOGL.

9.2.5.1 JOGL

JOGL[5], tedy rozhraní zpřístupňující v Javě funkce knihovny OpenGL, poskytuje kompletní funkcionalitu tohoto API, navíc pevně navázanou na strukturu tříd uživatelského rozhraní v Javě. Dostupná je i možnost programovat shadery v jazyce GLSL (OpenGL Shading Language).

JOGL by bylo možné využít také pro realizaci pokročilejšího 2D zobrazení.

Samotná realizace 3D zobrazení by pravděpodobně zahrnovala doplnění jednoduchého animačního systému a systému umístění světel do světa (například pro odlišení dne a noci).

Zobrazení mapy pak zahrnuje pouze terén (polygony), postavy a objekty (sprity - billboarding), předměty a efekty dovedností (opět billboarding nebo jednoduché texturované polygony).

V nastavení kamery bude vhodný pohled třetí osoby (kamera umístěna na souřadnicích odvozených od souřadnic avatara, ve větší výšce) a pravděpodobně určité omezení dohledu volbou úhlu pohledu - především pro zamezení zobrazení konce mapy.

Zajímavým doplňkem by mohl být jednoduchý částicový systém pro simulaci efektů počasí a kouře.

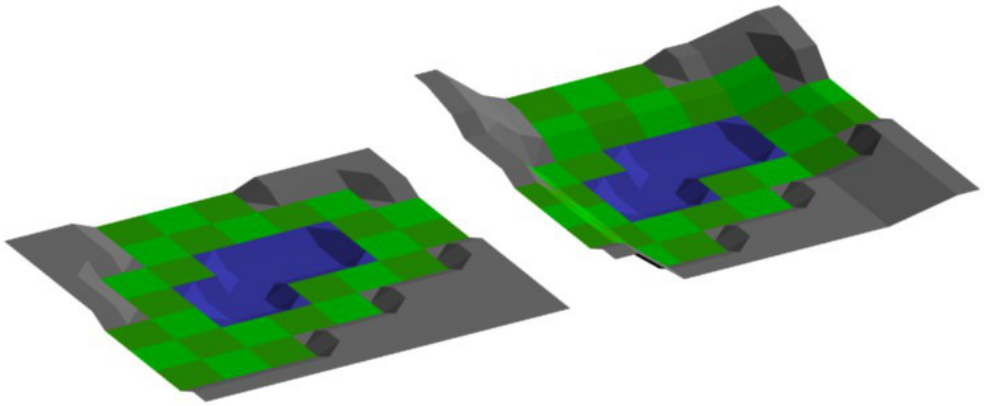
9.2.6 Mapy - výšková mapa a výška podle typu terénu

Při doplnění modulu pro 3D grafiku bude mít smysl přidat terénu třetí rozměr. Snadnou možností realizace bude výšková mapa, která navíc půjde snadno upravovat v editoru map.

Zajímavou možností je navázat reliéf terénu i na jeho jednotlivé druhy - například voda mírně snížená, cesta mírně vystouplá. K výškové mapě by pak stačilo jednotlivé koeficienty v daných sektorech přičíst.

9.2.7 Výchozí sada grafických dat

Pro skutečné použití systému bude nutné vytvořit dostatečně rozsáhlou sadu textur, obsahující několik sad terénů pro různá prostředí, textury objektů, sadu obličejů a spritů různých avatarů a nepřátel.



Obrázek 9.1: Reliéf terénu a kombinace s výškovou mapou

9.2.8 Zvuk a hudba

Zvukový systém nebyl součástí zadání - jednak v žánru RPG her není klíčový, jednak neovlivňuje ostatní části systému - pouze reaguje na určité události.

V konečné podobě by mohl stačit systém, přehrávající hudbu v MIDI formátu a přehrávající jednorázově zvuky v některých konkrétních situacích, například:

- použití dovednosti
- poražení nepřítele
- manipulace s předměty

Hudba by mohla být navázána na aktuální mapu (jako množina skladeb s náhodným výběrem), případně na výslovnou specifikaci některých událostí (zde by již byl potřebný skriptovací systém).

9.2.9 Shrnutí

Dá se očekávat, že se ještě objeví další témata k rozšíření. Důležité je, že zpracování v podobě diplomové práce bylo výraznou motivací důkladně se věnovat návrhu a využívat co nejsystematičtější a nejčistší řešení, díky čemuž vynikl systém, který je snadno možné dále rozšiřovat a několik modulů, které již nebudou potřebovat další úpravy.

Kapitola 10

Závěr

Práce popisuje návrh a realizaci specifického GCS (Game Creation System) zaměřeného na žánr RPG her.

Výsledkem analýzy existujících řešení je, že žádný z několika mála dostupných systémů tohoto zaměření nesplňuje zvolená kritéria, především pokud jde o otevřenost a konfigurovatelnost, proto bylo realizováno vlastní řešení.

V návrhu bylo specifikováno několik obecných principů, používaných při implementaci a po analýze dostupných možností byl navržen textový formát konfiguračních souborů pro ukládání nastavení a dat jednotlivých her v systému vytvářených.

Implementace se týkala dvou částí - první je herní engine, zahrnující funkcionalitu zpracovávající události v herním světě, chování nepřátel, grafické rozhraní a především RPG systém, realizovaný jako modul s rozsáhlými možnostmi nastavení pravidel pro vývoj postav, druhou uživatelské rozhraní pro autory her, kde byl hlavní důraz kladen na editor map, aplikaci umožňující s využitím grafických nástrojů snadno vytvářet herní prostředí.

Systém a jeho možnosti byly otestovány vytvořením dvojice her, tedy rozdílných sad pravidel a dat, pozornost byla věnována i analýze efektivity implementace důležitých součástí.

Implementovaný systém byl ještě podroben zpětnému srovnání s existujícími řešeními, ve kterém obstál.

I po dokončení práce je stále mnoho možností, jak v implementaci pokračovat, součástí textu je i popis nejdůležitějších z nich.

Literatura

[1] Game editor - web projektu. http://game-editor.com/Main_Page.

[2] Game maker - web projektu. http://www.yoyogames.com/make.

[3] jclassicrpg - web projektu. http://sourceforge.net/apps/mediawiki/javacrpg.

[4] jmonkey engine - web projektu. http://www.jmonkeyengine.com/.

[5] Jogl - web projektu. http://kenai.com/projects/jogl/pages/Home/.

[6] Ogre 3d - web projektu. http://www.ogre3d.org/.

[7] Rpg builder 3d - oficiální fórum. http://rpbuilder.org/forum/.

[8] Rpg maker - hlavní stránka projektu. http://tkool.jp/products/rpgvx/eng/index.html.

[9] Xml na webu w3c. http://www.w3.org/XML/.

[10] Wine - oficiální web. http://www.winehq.org/.

[11] Yoyo games - web. http://wiki.yoyogames.com/index.php/Information_For_Teachers.

Příloha A

Seznam použitých zkratek

API Application Programming Interface

GCS Game Creation System

GLSL OpenGL Shading Language

GUI Graphical User Interface, grafické uživatelské rozhraní

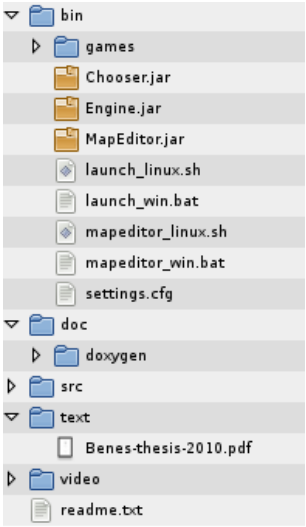
MIDI Musical Instrument Digital Interface

NPC Non-player Character

RPG Role-playing Games

Příloha B

Obsah přiloženého CD



Obrázek B.1: Obsah přiloženého CD

Adresář bin obsahuje spustitelné verze enginu a editoru s dávkovými soubory pro linux a windows.

V adresáři doc je umístěna dokumentace vytvořená systémem doxygen a vyexportovaná jako html.

Zdrojové kódy jsou umístěny v adresáři src, text práce v pdf v adresáři text.

Adresář video obsahuje několikaminutová videa z vytvořených testovacích her.

Vše (včetně instalace a ovládání) je ještě shrnuto v souboru readme.txt.