UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Rozšíření funkčnosti vybrané open source počítačové hry

Martin Holubec

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Holubec**
Osobní číslo:	**I10057**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Rozšíření funkčnosti vybrané open source počítačové hry**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je rozšíření serverové části vybrané open source počítačové hry. Daná hra bude doplněna o funkčnost, která významným způsobem rozšíří aktuální možnosti hry. Hra bude doplněna o další aktéry do hry, kteří budou disponovat umělou inteligencí se zaměřením na identifikace hráče ve hře. V návaznosti na toto rozšíření hry budou také doplněny další příkazy pro administrátory.
Pro realizaci cílů bakalářské práce bude využito vyššího programovacího jazyka Java a databázového systému MySQL. Dokumentace rozšiřujících prací v bakalářské práci bude provedena s využitím jazyka UML.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ECKEL B. Thinking in Java (4th Edition). Prentice Hall 2006. ISBN-13: 978-0131872486.
2. GILFILLAN I., Mastering MySQL 4. Sybex, 2003. ISBN 978-0782141627.

Vedoucí bakalářské práce: **Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2014

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Chrudimi dne 7. 4. 2014 Martin Holubec

Poděkování

Rád bych tímto poděkoval svému vedoucímu bakalářské práce Ing. Michaelu Bažantovi Ph.D. za vedení, rady a pomoc při řešení problémů, nejen při tvorbě této práce, ale i během studia.

Dále děkuji své přítelkyni Lucii Burešové, za podporu po celou dobu studia, za motivování do práce a pomoc při stylizaci. Následně děkuji celé své rodině za podporu morální, duševní i finanční po dobu mého studia, a že ve mne nepřestala věřit.

Anotace

Cílem bakalářské práce je rozšíření serverové části vybrané open source počítačové hry. Daná hra bude doplněna o funkčnost, která významným způsobem rozšíří aktuální možnosti hry. Hra bude doplněna o další aktéry do hry, kteří budou disponovat umělou inteligencí se zaměřením na identifikace hráče ve hře. V návaznosti na toto rozšíření hry budou také doplněny další příkazy pro administrátory.

Pro realizaci cílů bakalářské práce bude využito vyššího programovacího jazyka Java a databázového systému MySQL. Dokumentace rozšiřujících prací v bakalářské práci bude provedena s využitím jazyka UML.

Klíčová slova

Java, MySQL, Lineage, invaze města, programování, Eclipse, UML

Title

Extension of selected open source computer game

Annotation

The objective of the bachelor's theses is extension of selected open source computer game. Game will be extended for function, which will significantly improve current game options. Game will have added new actor objects, whose are going to have artificial intelligence (AI) with focus to identify players in game. In connection to that extension there will be added new commands for administrators and players.

To achieve objectives of the bachelor's theses will be used higher programming language Java and database system MySQL. Documentation of extension will be done in UML language.

Key words

Java, MySQL, Lineage, town invasion, programming, Eclipse, UML

Obsah

Seznam zkratek ... **9**
Seznam obrázků ... **10**
Seznam tabulek ... **10**
1 Úvod ... **11**
2 Požadavky na uživatele ... **12**
2.1 Informace o projektu popisovaném v této práci ... 12
3 O Lineage 2 ... **13**
3.1 O světě ... 13
3.2 Informace o hraní ... 13
3.3 Vybavení hráče ... 13
3.4 Hráčské statistiky a jejich ovlivnění ... 14
3.5 Rasy ... 15
3.6 Povolání postav ... 15
3.7 Olympiáda ... 16
3.8 Klan ... 16
3.9 Aliance ... 17
3.10 Události ... 17
3.11 Bossové ... 18
4 Architektura projektu a popis balíčků L2J projektu ... **19**
4.1 Rozdělení jádra ... 19
4.1.1 Obecné ... 19
4.1.2 Herní server (Game server) ... 20
4.1.3 Log ... 30
4.1.4 Log-in server ... 30
4.1.5 Status ... 30
4.1.6 Nástroje (tools) ... 30
4.1.7 Utility (util) ... 31
5 Popis tříd a souborů invaze města ... **32**
5.1 TownInvasionManager.java ... 32
5.2 TownInvasionTask.java ... 32
5.3 TownInvasionEvent.java ... 32
5.4 TownInvasionTown.java ... 33
5.5 TownInvasionWave.java ... 34
5.6 L2PcInstance.java ... 34
5.7 L2TIMonsterInstance.java ... 35
6 Konfigurace invaze ... **36**
7 Stav události ... **37**
8 UML Diagramy ... **38**
9 Use case diagram ... **42**
10 SQL tabulky ... **43**
11 Závěr ... **45**

Literatura **46**
A. Příloha – Zdrojový kód metody eventStart() **47**
B. Příloha – Zdrojový kód třídy TownInvasionTask **48**
C. Příloha – Stromy ras a jednotlivích povolání **49**

Seznam zkratek

AI	umělá inteligence
DD	damage dealer – ten, co dělá hodně poškození za krátký čas
GUI	grafic user interface – grafické uživatelské rozhrání
ID	identifikační číslo
H5	High Five – verze L2
HTML	hypertextový jazyk
MMORPG	massively multiplayer online role-playing game
	hromadná online hra na hrdiny pro více hráčů
NPC	z anglického non-player character, je to postava ovládaná počítačem
L2	Lineage 2
L2J	Lineage 2 Java
PvE	Player vs. Environment – hráč proti
PvP	Player vs. Player – hráč proti hráči
SQL	Structured Query Language – strukturovaný dotazovací jazyk
TI	Town Invasion – invaze města
TvT	Team vs. Team – tým proti týmu
UML	Unified Modeling Language – unifikovaný modelovací jazyk
XML	Extensible Markup Language – rozšiřitelný značkovací jazyk

Seznam obrázků

Obrázek 1 - Základní vlastnosti hráče – Zdroj[hra Lineage 2] 14
Obrázek 2 - Statistiky hráče a ikony efektů – Zdroj[hra Lineage 2] 15
Obrázek 3 – Rasy – Zdroj [2] 15
Obrázek 4 - Rozdělení herního serveru 20
Obrázek 5 - Dědičnost od L2Character 23
Obrázek 6 - Dědičnost pozice objektu 24
Obrázek 7 - Dědičnost od CharStat 24
Obrázek 8 - Dědičnost od CharStatus 25
Obrázek 9 - Dědičnost šablon 25
Obrázek 10 - Dědičnost od ItemContainer 26
Obrázek 11 - Třídy věcí a jejich dědičnost 27
Obrázek 12 - UML Olympiáda 28
Obrázek 13 - Seznam úkolů správce úloh 30
Obrázek 14 - UML diagram hlavní třídy události 38
Obrázek 15 - UML diagram plánovače události 40
Obrázek 16 - Uživatelský use case diagram 42
Obrázek 17 - Stromy povolání jednotlivých ras – Zdroj [1] 50

Seznam tabulek

Tabulka 1 - Příklad dat měst v databázi 43
Tabulka 2 - Příklad dat vln NPC v databázi 44

1 Úvod

L2J je open source projekt – určený k hostování korejské MMORPG hry Lineage 2. První verze se datuje na 24. června 2004 a byla napsána vývojářem s přezdívkou L2Chef. Od té doby je neustále kód aktualizován a opravován za přispění komunity. Stávající verze L2J odpovídá ve stabilním sestavení Lineage 2 High Five, ale ve vývoji je již novější verze Goddes of Destruction, která je nyní ve fázi beta testování.

Projekt se skládá ze dvou hlavních částí, první je jádro serveru a druhá datový balík. V jádru jsou obsaženy všechny metody, závislosti a jiné věci spojené s celým během serveru. V balíku najdeme úkoly, skripty, HTML stránky, které se ve hře zobrazují jako dialogy a XML soubory s nastavením věcí, charakterů, schopností a vlastností. Další data jsou uložena přímo v databázi. To jsou například údaje o hráčích, NPC, souřadnice teleportů, listy se souřadnicemi, na kterých se objevují jednotlivá NPC a oznámení ve hře.

Více informací a pomoc s instalací lze naleznout na oficiálních stránkách L2J serveru. Video s instalací lze nalézt ve více verzích pro různé programy, ve kterých se projekt kompiluje. Nejčastější je Eclipse (Zdroj: [5]), ale od roku 2013 se začalo využívat i NetBeans (Zdroj: [6]). Lineage 2 má i vlastní stránky wikipedie (Zdroj: [3]) s rozsáhlými informacemi o hře.

Tato práce se zabývá rozšířením herního jádra o událost invaze města, jejím naprogramováním, nastavením a popsáním jednotlivých částí kódu a průběhu. Invaze města je událost, která je velice komplexní a bude testovat, jak jsou hráči nezávisle na alianci a klanu schopni spolupracovat při obraně města. Událost celkově využívá velké množství již před programovaných metod z open source projektu, ale značné množství je také nově naprogramováno. Tyto metody budou blíže popsány, vysvětleny a následně také demonstrovány.

Všechny kódy aplikace jsou psané v anglickém jazyce, a proto rovněž nově doplněné metody, třídy a komentáře v aplikaci jsou kvůli dodržení konvencí psané anglicky. U důležitých metod jsou použity dokumentační komentáře, napovídající, jaké parametry bude daná metoda potřebovat a co bude vracet.

2 Požadavky na uživatele

Ke zprovoznění vlastního herního serveru budete potřebovat alespoň základní znalosti SQL, Java, HTML a XML. SQL je potřeba k instalaci databáze. Nejčastěji se používají volně dostupné verze MySQL. Java se potřebuje k sestavení serveru ze staženého kódu. Znalost HTML a XML je nutná na úpravy souborů v datovém balíku.

2.1 Informace o projektu popisovaném v této práci

- Použitá databáze: MySQL 5.6.13.
- Verze serveru: stabilní verze 5937.
- Verze data balíku: stabilní verze 9641.
- Java IDE: Eclipse Standard/SDK
    - Build id: 20130919-0819
- Java/JDK: Java SE Development Kit 7 Update 25 (64-bit)

3 O Lineage 2

V této kapitole bude blíže popsána hra Lineage 2. Popíší se hlavní části hry z pohledu hráče, aby se pochopilo, o čem hra je.

3.1 O světě

Cílem této hry je být co nejlepší, být v klanu a alianci a společně s ostatními hráči vládnout světu Adenu. Aden je pojmenován celý herní svět a také jedno město. Na každém serveru si hráčská komunita vytvoří své hlavní město, které je centrem obchodu a komunikace mezi hráči. Na vlastních upravených serverech se toto město dá přednastavit, umístěním důležitých NPC všech do administrátorem zvoleného města.

Ve světě Adenu nalezneme devět měst, u kterých je hrad. Tyto hrady lze s klanem a aliancí dobýt a poté bránit proti ostatním klanům a aliancím. Vlastnictví hradu přináší klanu prestiž a další věci, suroviny a peníze. Pán hradu, zároveň i vůdce klanu, nastavuje v celém území města taxy na nákup u NPC. Tím mu do klanové pokladnice jdou peníze z obchodů vykonaných v jeho městě. Města s hrady jsou Gludio, Dion, Giran, Oren, Aden, Goddard, Heine, Rune a Shuttgard. Další města, která lze nalézt ve světě Adenu jsou Gludin, Hunter's Village a začátečnické vesnice jednotlivých ras.

3.2 Informace o hraní

Jako hráč se objevíte bez vybavení na úrovni zkušeností jedna. Maximální úroveň v Lineage 2 High Five je 85. Opět lze toto snadno modifikovat v jádru herního serveru, pokud chceme upravený server. Vybavení si hráč kupuje u NPC nebo hráčů za zlaté mince nazývané Adeny, případně vyrábí u trpaslíků. Adeny a jiné materiály padají z monster, pokud je hráč zabije, ale také je lze získat plněním úkolů. Tyto úkoly nejsou stěžejní podmínkou k tomu, aby hráč mohl zvýšit svou úroveň nebo si vydělal dostatek prostředků na nové vybavení.

3.3 Vybavení hráče

Vybavení hráče se skládá ze základních věcí a doplňků. Mezi základní počítáme zbraň, brnění a šperky. Brnění se skládá z více kusů a to hrudní brnění nebo tunika pro mágy, kalhoty, boty, rukavice a helma nebo korunka, opět pro mágy. Mezi šperky řadíme prstýnky, náušnice a náhrdelník. Zatímco brnění nás chrání proti fyzickým útokům, šperky mají funkci obrany proti magii a zvyšují magickou odolnost.

Zbraně, jak každému dojde, znamenají zvýšení magického i fyzického útoku daného hráče. Najdeme zde více typů zbraní a každý z nich je dobrý pro jiné povolání. Povolání hráče bude probráno níže. Typy zbraní jsou následující: luk, dýka nebo duální dýka, meč nebo duální meč, palcát, obouruční meč i palcát, rapír, starověký meč, kuše a obrněné rukavice. Největší fyzický útok mají luk a kuše následované obouručními zbraněmi. Pokud je zbraň

duální, znamená to, že má hráč v každé své ruce jednu dýku nebo meč. K použití luku a kuše si hráč musí obstarávat také šípy.

Doplňky na oficiálním serveru slouží pouze jako doplňky vzhledu vašeho charakteru. Na vlastních upravených serverech se do těchto doplňků přidávají ještě jiné speciální vlastnosti. Například pokud si hráč nasadí masku na obličej, zvednou se mu životy o 20 procent. Pokud si nasadí opasek, zvedne se mu útok o 1000. Jednotlivé statistiky se dají multiplikovat, přičítat nebo se nastavuje základní hodnota.

3.4 Hráčské statistiky a jejich ovlivnění

Hráči mají šest základních vlastností, které se dají zvyšovat a snižovat určitými způsoby. Těmito vlastnostmi jsou síla, obratnost, konstituce, inteligence, mentalita a moudrost. Všechny tyto vlastnosti ovlivňují další statistiky hráče, což jsou fyzický a magický útok, fyzická a magická obrana, rychlost, rychlost útoku a sesílaní kouzel, životy a mana, míření a úhyb. Síla ovlivňuje fyzický útok a inteligence útok magický. Obratnost přidává rychlost, zvedá míření a úhyb a zvyšuje rychlost útoku. Moudrost zvyšuje šanci na kritický hit magií a rychlost sesílání kouzel. Konstituce přidává fyzickou obranu a odolnost proti fyzickým schopnostem (např. omračující střela). Mentalita hráče chrání proti magii a mentálním útokům (např. uspání, strach) a zvedá množství many, které hráč může použít na své schopnosti.

Základní vlastnosti se ovlivní za pomoci vhodného poskládání setu z brnění, nebo nakreslením barevných symbolů. Pokud si kreslíme symbol, tak se nám vždy jedna vlastnost zvýší a druhá se sníží. Proto si hráč musí dobře rozmyslet, jakou kombinaci si zvolí. Dají se kombinovat pouze fyzické vlastnosti mezi sebou a mentální mezi sebou. Takže si mág nemůže vzít sílu a přidat si inteligenci. Maximum přidané k jedné vlastnosti je plus pět, ale záporné můžeme mít i vyšší. Ukázka je na Obrázek 1.

Obrázek 1 - Základní vlastnosti hráče – Zdroj[hra Lineage 2]

Fyzický útok a ostatní statistiky se dají modifikovat pomocí speciálních schopností postav. Některé se dají používat pouze na sebe, jiné na partu, další na klan a alianci a jiné i na libovolné hráče. Po použití těchto schopností se do listu efektů na postavě přidá efekt dané schopnosti. U povolání prorok (prophet) můžeme nalézt schopnosti síla, štít, magická bariéra, bystrost a spoustu dalších, trvají většinou 20 minut a dají se použít na kohokoli. Všechny předchozí schopnosti násobí statistiku, kterou ovlivňují. Například síla nám zvedne na úrovni tři schopnosti náš fyzický útok o 15 procent.

Obrázek 2 - Statistiky hráče a ikony efektů – Zdroj[hra Lineage 2]

3.5 Rasy

Ve světě Adenu nalezneme šest různých ras, ze kterých si hráči při zakládání postavy mohou vybírat. Obratné elfy s lehkou konstitucí a velikou moudrostí, temné elfy s velikou obratností, silou a inteligencí, průměrného člověka vyváženého ve všech směrech, neohrabané orky s obrovskou silou a dobrou konstitucí, trpaslíky s dobrou odolností a silou, ale pomalé a moc neobratné a nakonec kamaely, rasu padlých andělů s jedním křídlem a používající starobylé zbraně.

Obrázek 3 – Rasy – Zdroj [2]

3.6 Povolání postav

Každý hráč si může zvolit své povolání již při tvorbě postavy. Záleží, jakou si vybere rasu, a následně povolání. Po dosažení úrovně 20, 40 a 76 si hráč zvolí, jaké další zaměření bude jeho postava mít. Pro změnu povolání se plní takzvané přestupové úkoly. Po jejich splnění si hráč u svého mistra své povolání změní. Celý strom lze vidět v příloze (C).

Základní rozdělení je bojovník a mág. Mágové jsou dále děleni na kouzelníky, podpůrné mágy, vyvolávače a léčitele. Kouzelníci dělají největší poškození. Vyvolávači používají vlastní vyvolané NPC, které jim pomáhá při boji. Toto NPC může mít funkci bojovou, ale také pomocnou a léčivou. Podpůrní mágové jsou charaktery se schopnostmi ovlivňujícími ostatní schopnosti. Léčitelé mají za úkol uzdravovat členy party nebo klanu, případně je oživovat. Někteří mágové mohou zastat více funkcí s nižšími schopnostmi (Overlord, Doomcryer), mohou léčit, dělat poškození, ale i zvyšovat statistiky. Jejich poškození je znatelně menší, než od kouzelníků.

Bojovníci mají dělení složitější, protože co povolání to jiný druh zbraně. Takže se v podstatě dá říci bojovník s dýkou, lučištník, tank – povolání používající těžké brnění a jedno ruční meč a štít. Další povolání jsou přiřazena ke svým zbraním následovně: dualista – duální meče, grand kvahatari – ork s obrněnými rukavicemi, titán – ork s obouručním mečem nebo palcátem, tanečník s meči (bladedancer) – duální meče, zpěvák s mečem (swordsinger) – jedno nebo dvou ruční meč, kušištník (trickster) – kamael s kuší, nositel zkázy (doombringer) – kamael se starověkým mečem, další je kamael s rapírem a poslední jsou trpaslíci – u těch můžeme používat většinu zbraní, ale zaměření mají na palcáty.

Během hry může hráče přestat bavit hrát za jeho povolání, ale pokud by nechtěl zakládat novou postavu, stačí mu splnit úkol, díky kterému si může vybrat vedlejší povolání. Toto povolání si zvolí ze seznamu povolání na úrovni 40. Ale problémem je, že na těchto vedlejších povoláních nefungují určité schopnosti spojené pouze s hlavním povoláním. Po změně povolání si kdykoli můžeme změnit povolání zpět nebo na další. Oficiálně může mít hráč až tři vedlejší povolání.

3.7 Olympiáda

Ve hře se nachází olympiáda k ověření schopností hráčů. Utkávají se spolu na kolbišti jeden na jednoho nebo v týmech tři na tři. Mohou si vybrat, zda budou bojovat pouze proti hráčům se stejným povoláním, jaké mají oni nebo zda jim je to jedno a mohou soutěžit třeba lučištník proti mágovi. Olympiády se lze účastnit pouze za hlavní povolání a hráči musejí mít status noblesy. Ten se získá splněním úkolu a požaduje se mít alespoň jedno vedlejší povolání na úrovni 60. Po uplynutí měsíce se vyhodnotí zápasy a hráči za své povolání s nejvíce body získají titul hrdina.

S titulem hrdina jsou spojeny extra schopnosti, které má hráč na hlavním povolání v záložce schopností. Hrdina září a má možnost vybrat si svou hrdinskou zbraň se speciálními vlastnostmi. Může používat globální chat pro hrdiny, to znamená, že kdekoli na mapě může napsat text a zobrazí se všem hráčům. Získání hrdiny přinese také reputační body pro klan.

3.8 Klan

Klan funguje na spolupráci jeho členů. Hráči se spolu účastní klanových akcí, vytvářejí si party na zabíjení monster v těžších lokacích, plní spolu úkoly a celkově se chovají jako

jeden celek. Mohou spolu komunikovat v klanovém chatu nebo používat komunikační programy umožňující spojení více lidí najednou (Ventrilo, Team Speak, Skype…). Tyto programy se nejvíce uplatní při dobývání hradů a hromadných nájezdech na bosse, urychlí to komunikaci a hráči ihned vědí, co mají dělat, aby pomohli co nejvíce.

V klanu je jeden vůdce, ten určuje práva ostatních hráčů a spravuje celý klan. Může si zvolit svého zástupce, ale tato funkce je pouze mezi hráči, ve hře ji nenaleznete. Vůdce učí klanové schopnosti za reputační body, vybírá, jaký znak bude jeho klan mít a zve, případně vyhazuje, hráče do klanu. Také vyhlašuje a ukončuje klanové války, na aukci může přihazovat na klanové haly, a pokud vlastní hrad, tak jezdit na wyverně.

Reputační body se získávají více způsoby. První a nejjednodušší jsou klanové války. Za zabití nepřítele z klanu, se kterým má klan hráče válku, se přičte jeden reputační bod. Další body lze získat za to, když hráč získá v daném měsíci hrdinu, když klan dobije hrad nebo plní úkoly pro získání reputace.

Vůdce klanu je nepostradatelná součást dobývání hradu, aby mohl jeho klan hrad dobýt, musí jeho vůdce použít schopnost na artefakt v hradu. Členi klanu tuto schopnost nemají a bez vůdce si mohou pouze užít souboje.

3.9 Aliance

Aliance je sdružení více klanů do jednoho celku. Členové jednotlivých klanů spolu mohou komunikovat pomocí aliančního chatu. Vůdce aliance volí ikonu celé aliance, která je umístěna vedle ikony klanu u jména hráče.

3.10 Události

Ve hře jsou základní sezónní události jako velikonoční, vánoční, novoroční, při kterých ze zabitých monster padají speciální věci, které hráči mohou vyměňovat za odměny. Další herní události jsou například otevírání truhliček s pokladem, závod, při kterém jsou hráči proměněni v žábu nebo i v této práci řešená invaze města.

Dalším druhem událostí jsou vedeny pomocníkem administrátora nebo administrátorem samotným. Tyto události nemají prakticky žádná omezení. Administrátor si sám vymyslí, co dá za odměnu a co musí hráči udělat a oznámí jim to přímo ve hře pomocí oznámení. Mezi tyto události by se dalo zařadit schovávaná, při které se administrátor schová někde na mapě, dá hráčům pár nápověd a ti ho poté hledají. Jednodušší je házení kostek anebo ruská ruleta.

Pokud by hráči měli stále plnit jen úkoly anebo zabíjet monstra tak by je hra přestala velice rychle bavit. Proto zde jsou tyto události, olympiáda a bitvy o hrady. Mezi nejlepší hráč proti hráči (PvP) patří souboje během epických bossů. Ty se objevují jednou za den až týden a proto jsou věci, které z nich padají, velice vzácné a prestižní.

3.11 Bossové

Ve hře se nacházejí silnější monstra nazývaná bossové. Mají tak veliké statistiky, že na ně chodí většinou parta lidí a trvá jí několik minut, a někdy i hodin nežli bosse zabijí. Proto z bossů padá větší množství věcí s větší šancí na vypadnutí, než je šance a množství z monster normálních. Mají také delší interval znovuzrození (respawn).

Epičtí bossové jsou nadřazeni bossům normálním. Chodí na ně celé klany a aliance. Mají miliony životů a poškození, které jim maximálně hráči dělají, se pohybuje v řádech tisíců. Boss sám dělá poškození tak velké, že je potřeba několik léčitelů. Někteří bossové mají dokonce různá kouzla, kterými přimějí hráče na útěk nebo je rovnou zabijí.

4 Architektura projektu a popis balíčků L2J projektu

Celkový projekt se skládá z více než 1600 Java souborů v jádru a přes 60 000 souborů v datovém balíku. Soubory v jádru se dělí podle své funkce do balíčků, kde se sdružují soubory podobné si vzájemně svou funkcí v projektu. V datovém balíku jsou soubory rozděleny také do balíků, ale některé typy souborů (HTML, XML) jsou rozděleny do adresářů a v projektu se neupravují, pokud neděláte vlastní server s vlastním nastavením vybavení, schopností a dalších možností. V datovém balíku je jeden balíček pro každý úkol, je jich přes 500 a najdeme zde i manipulátory, ve kterých jsou příkazy a by-pasy z herních dialogů nebo příkazů.

4.1 Rozdělení jádra

Celé jádro lze rozdělit do hlavních oddílů:

- Obecné,
- Herní server,
- Log-in server,
- Log,
- Status,
- Nástroje (tools),
- Utility (utils).

V každém se nacházejí balíčky a soubory spojené s daným balíčkem.

4.1.1 Obecné

V této části projektu lze nalézt pouze tři soubory a to Config.java, L2DatabaseFactory.java a Server.java. Všechny jsou společné pro většinu balíčků a používají se v nich přes příkaz import. Config.java obsahuje všechna nastavení serveru, včetně defaultních hodnot. Pomocí tohoto souboru se načtou do mezipaměti serveru všechny proměnné používané na serveru, spojené s nastavením. Nastavení se po rozbalení zkompilovaného projektu nachází vždy ve složce *Config*. Log-in server i game server mají vlastní konfigurační adresáře.

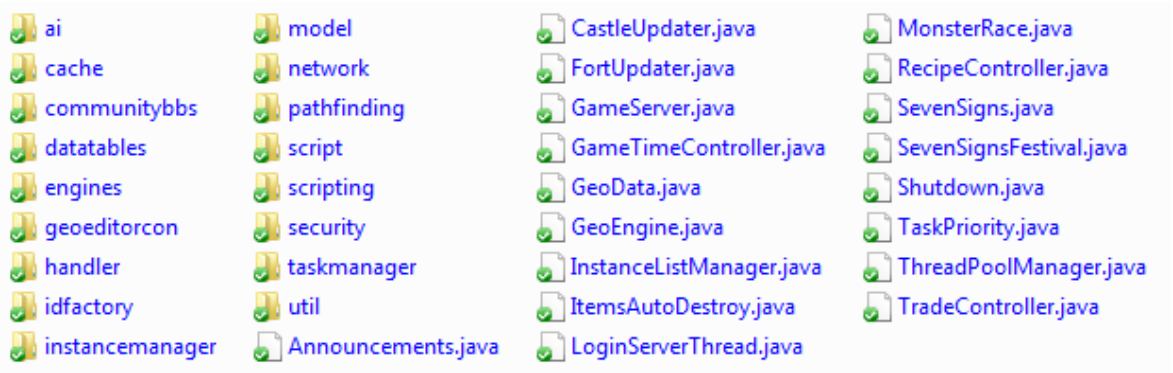
Nastavení serveru lze provézt pohodlněji pomocí *L2J_Configurator.jar*, což je mini program co načte všechny soubory ze složky *Config* a předá je uživateli pomocí jednoduchého GUI. Uživatel zde místo psaní *pravda* nebo *nepravda* jednoduše zaškrtne políčko a tím je daná hodnota změněna. Pokud by se jednalo o textové pole, doplní dle vzoru uživatel hodnoty, které potřebuje.

Server.java slouží již pouze pro uchování typu serveru. Dříve to byl spouštěcí soubor pro celý server, ale od rozdělení na herní a log-in server se již nepoužívá.

L2DatabaseFactory.java je nepostradatelný soubor, obsahující ovládání databáze a metody s tím spojené. Nastavení připojení je převzato z Config.java a využívá se ve všech balíčkách nebo souborech, které potřebují připojení do databáze.

4.1.2 Herní server (Game server)

Herní server si lze opět rozdělit na menší celky, které se budou lépe popisovat zvlášť. Jednotlivé balíčky lze vidět na následujícím obrázku, kde jsou zobrazené jako adresáře. Soubory s koncovkou *java* jsou přímé soubory v balíčku herního serveru. Některé důležité budou popsány následně.



Obrázek 4 - Rozdělení herního serveru

Announcements.java slouží ke zprostředkování textu a informací mezi hráči a herním serverem. Pokud server něco oznámí hráčům ve hře, objeví se jim v konverzačním okně světle zeleno-modrý, případně tmavě zeleno-modrý text. Tmavý je používán pro takzvaná kritická oznámení, zatímco světlé pro normální systémové hlášení.

Jako další, nejdůležitější soubor, je zde GameServer.java. V něm nalezneme použití všeho, co na serveru běží, a pokud to není zde, tak na to odkazuje nějaká instance, kterou v sobě herní server má. Obsahuje spustitelnou metodu *main()* a v konzoli po spuštění vidíme následné načítání serveru po sekcích. Načtou se databázové položky, věci, obchodní listy, schopnosti, pozice NPC, na které se mají jednotlivá NPC objevit, data olympiády, vedlejší události jako tým proti týmu, invaze města, manipulátory a příkazové by-pasy, spustí se plánovač vláken a kontrolér času herního serveru.

LoginServerThread.java slouží ke spojení herního serveru s log-in serverem. Pomocí vygenerovaného hexadecimálního ID se identifikuje herní server do vědomí log-in serveru a udržují spojení, dokud ho jeden z nich neukončí. Herní server slouží pouze ke hraní, a tudíž není potřeba ho ještě více komplikovat přihlašováním uživatelů a ověřováním s databází. Přes vlákno tohoto souboru se přihlášený uživatel dostane ke svým herním postavám a následně je může přihlásit do hry.

Shutdown.java ukončuje a plánuje ukončení právě běžícího serveru. Pokud administrátor ve hře zadá příkaz restartování nebo ukončení, server se automaticky po zadaném čase provede zadanou úlohu. Ukončí se všechny instance, uloží se data zpět do databáze a jako poslední se zavře spojení s databází.

Již dle názvu lze říci, že TaskPriority.java slouží k nastavení priority jednotlivých úkolů, plánovaných vláknovým manažerem. Nutno podotknout, že se jedná o *enum*, tedy výčtový typ.

ThreadPoolManager.java, soubor bez kterého by byl server pomalý, protože by se vždy čekalo, až se provede daná úloha a nedala by se odložit o nějaký čas. Pomocí vláknového manažeru jasně určíme, kdy se jaký úkol provede, a pokud by mělo nastat jeho odložení, tak se nebude na procesoru čekat, ale začne se provádět úkol následující a stávající se odsune do plánovače na předem určený čas s danou prioritou. Pomocí plánovače se provádějí ukládání do databáze, update listu známých objektů, regenerační metody a všechny asynchronní události na serveru.

4.1.2.1 Herní server – AI

Balíček AI se zabývá problematikou umělé inteligence objektů v herním světě. Každý objekt, který by měl vykonávat nějakou činnost ve hře, má vlastní AI. Vlastní AI se nemyslí, že každý objekt má naprogramováno myšlení samostatně, ale skupiny objektů s podobným nebo stejným myšlením sdílejí stejné AI. AI mají i hráči na serveru, aby nemuseli všechny úkony vykonávat sami. Pokud hráč útočí na NPC a použije schopnost, tak ihned po dokončení schopnosti opět zaútočí, protože mu to říká jeho AI.

V herním světě jsou za objekty považované i lodě, všechna NPC, ale i vzducholodě. AI pro lodě a vzducholodě plánuje další trasu a udává hráčům pozici, kde se právě nacházejí, i když se oni vzhledem k lodi nepohybují, ale stojí na místě.

4.1.2.2 Herní server – mezipaměť (cache)

Mezipaměť serveru obsahuje data o HTML stránkách, podle nastavení konfiguračního souboru, lze při spuštění serveru načítat do mezipaměti všechny soubory, nebo použít druhé nastavení (častější) a do mezipaměti daný soubor nahrát až při prvním použití hráčem na serveru. Další mezipaměť je vyhrazena pro sklad věcí hráčů a klanů a nakonec i ikon klanů. Ikona klanu se zobrazuje vedle jména hráče.

4.1.2.3 Herní server – komunita (communitybbs)

V balíku nalezneme soubory související s tabulkou komunity ve hře. Ke kompletnímu spuštění této tabulky je potřeba mít zprovozněn i komunitní server, obsahující přátele hráče a herní mail klient. Na většině serverů je komunita vypnutá, nebo slouží pouze pro zobrazení přihlášených hráčů.

4.1.2.4 Herní server – tabulky dat (datatables)

Pokud načítáme data z databáze, a víme, že tato data budeme potřebovat používat neustále, je lepší si je uložit do tabulky dat v herním serveru, než pokaždé prohledávat databázi a brát potřebné údaje. Z databáze se plní tabulky klanů, hráčů, šablon hráčů, lokace a

statistiky NPC, souřadnice teleportů, na které nás přemisťují by-pasy strážců bran (Gatekeeper) nebo příkazy administrátorů a jiná další data.

Do těchto tabulek také ukládáme roztříděná data z XML souborů. Ty se třídí pomocí souborů s příponou *xsd*, ve kterých je udáván počet jednotlivých parametrů u daného XML souboru. Základním parametrem je *SEZNAM (LIST),* do kterého jsou jako další vkládány další parametry jako například *schopnost (skill)* nebo *věc (item)*. Ty se dělí dále podle vlastností každého prvku zvlášť.

4.1.2.5 Herní server – nástroje (engines)

V tomto balíku lze nalézt nástroje pro třídění dokumentů, zmíněných o kapitolu dříve. Určují, je-li soubor validní a následně ho zpracují.

4.1.2.6 Herní Server – geografický editor (geoeditorcon)

Určuje stávající polohu objektů, vůči sobě, geografický editor není obsažen v projektu L2J, proto se používají defaultní data.

4.1.2.7 Herní server – manipulátory (handlery)

Obsahuje *interface* jednotlivých manipulátorů z datového balíčku a také nějaké manipulátory vlastní. Manipulátory nám určují, jak přijatá data zpracovat a bude se s nimi manipulovat. Hlavni je *IHandler.java*, který se implementuje nebo rozšiřuje ostatní.

4.1.2.8 Herní server – továrna ID (idfactory)

Třídy v balíku se starají o udržování databáze, promazávají nebo ukládají potřebná data, v závislosti na ID objektu, uloženém v databázi. V případě tvorby objektu se postará, aby jeho ID bylo jedinečné a nemohl ho mít žádný jiný objekt na serveru.

4.1.2.9 Herní server – manažeři instancí (instancemanager)

Ve hře se spouštějí události a ovládají se těmito manažery. Pokud máme epického bosse, tak celý jeho průběh je ovládán odtud pomocí *GrandBossManager*. Dobývání hradů, přihlašování a jiné s tím spojené se nachází uvnitř *TownManager*. Další manažeři spravují úkoly, prokleté zbraně, petice, maily, ostrov Hellbound, aukce, bitvy o klanové haly a jiné.

4.1.2.10 Herní server – model

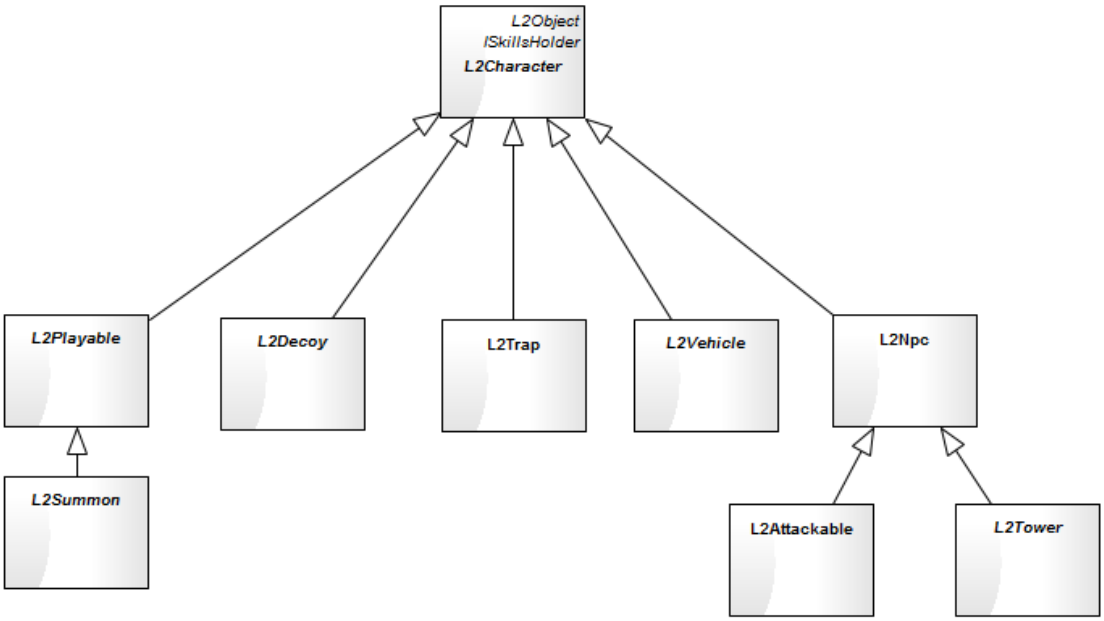
Nejkomplexnější balík ze všech, proto bude rozdělen a jeho balíky zvlášť popsané. Obsahuje nejvíce informací, nastavení a druhů entit.

Model - hlavní balík

Ve virtuálním světě serveru jsou objekty, a každý druh těchto objektů, má vlastní třídu s vlastnostmi, metodami a podtřídami. Jako objekt je brán i klan nebo člen klanu, ale přitom člen klanu není brán jako hráč. Model jednotlivého objektu je přesněji specifikován v balících podřízených tomuto.

Model – představitel (actor)

Abstraktní třída *L2Character* je zde rozšířena a rozdělena pro všechny pohyblivé objekty. Třída *L2Npc* a *L2Playable* rozšiřují tuto třídu přímo, další modely rozšiřují tyto dva. Jsou to vlastní třída pro vyvolané NPC pomocí schopnosti, reagující na povely hráče, *L2Attackable*, které určí, že na daný model lze (či nelze) útočit a styl, jakým získává, nebo dává poškození. *L2Trap, L2Vehicle* a *L2Decoy* jsou reprezentanti pastí, vozidel a zmizení charakteru ze světa.

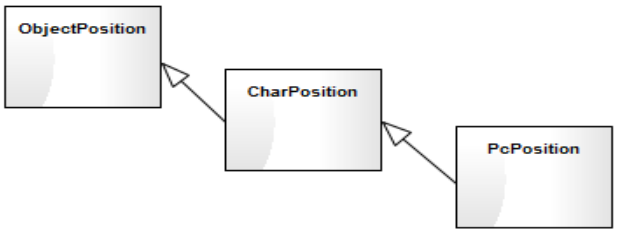


Obrázek 5 - Dědičnost od L2Character

Jako sub balík je zde vzhled, ve kterém je definováno, jaký vzhled se bude hráči zobrazovat. Vzhled se tvoří pouze pro entity *L2PcInstance*, ve které je jako parametr.

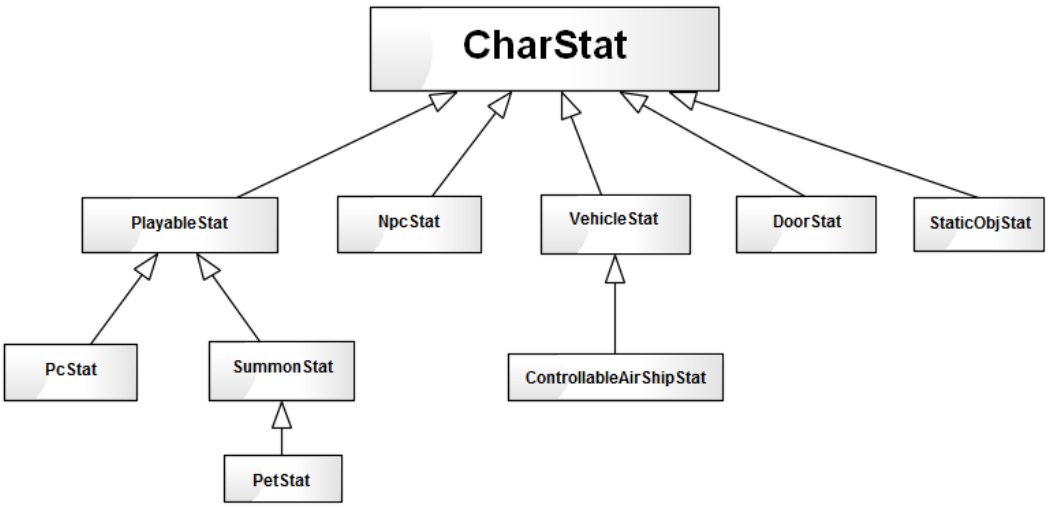
Nyní konkrétní rozdělení představitelů jednotlivých objektů. Naleznete je v balíku instance, ve kterém jsou umístěny i soubory instance invaze města (*L2TIMonsterInstance* a *L2TIRegNpcInstance*). Hráči pro své vlastnosti a sety schopností využívají *L2PcInstance,* ta rozšiřuje třídu *L2Playable*. Při tvorbě každé entity *L2PcInsance* se z databáze načtou všechna potřebná data pro daného hráče. Mezi další důležité patří *L2MonsterInstance,* prezentující každé NPC útočící na hráče, nebo *L2RaidBossInstance* rozšiřující *L2MonsterInstance,* a která existuje pouze jediná pro jedno ID bosse. Další třídy se většinou neupravují, ale jsou neméně důležité jako všechny předchozí.

Pozice objektu ve světě Adenu je určována třídami v balíku pozice (position). Hlavní je *ObjectPosition* a následně ji rozšiřuje pozice charakteru (nemusí být hráč). Ta je následně rozšířena pozicí hráče, ale ta obsahuje však pouze validaci, že je hráč ještě v mezích námi vyhrazeného virtuálního světa. Pokud by nebyl, teleportuje ho na nulové souřadnice a ohlásí, aby kontaktoval administrátora.



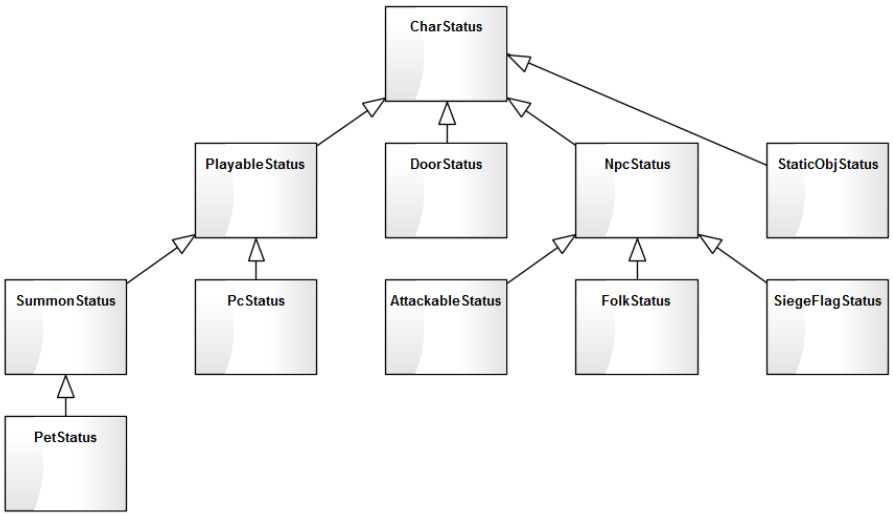
Obrázek 6 - Dědičnost pozice objektu

Statistiky a vlastnosti jednotlivých objektů (životy, rychlost, útočná síla…) se sdružují ve třídách statistik, které rozšiřují hlavní třídu se statistikami *CharStat.java*. Pro hráče se používá *PcStat*, která rozšiřuje *PlayableStat*, jež je potomkem *CharStat*. *NpcStat*, na rozdíl od *PlayableStat*, nepotřebuje metody spojené se zkušenostmi, protože NPC nemění svou úroveň. Další rozdíl je, že hráč má navíc životní energii (vitality), která se používá při získávání zkušeností jako bonusový multiplikátor. Jaká je dědičnost, lze názorně vidět na Obrázek 7.



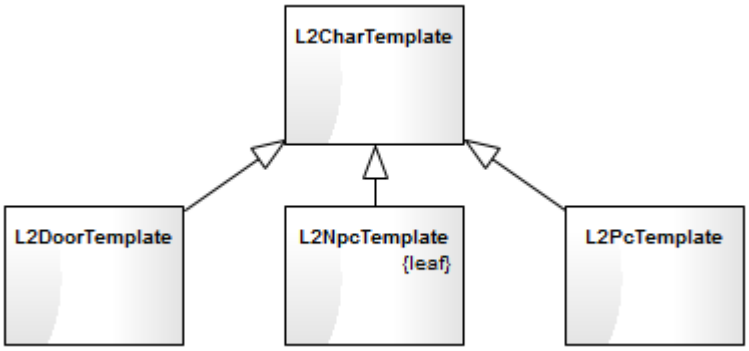
Obrázek 7 - Dědičnost od CharStat

Ve hře mají objekty vlastní status, určující jaký mají stávající počet životních bodů (HP), v případě hráče i bojových bodů (CP). Status soubory obsahují metody spojené s redukcí těchto bodů, případně jejich regenerací. Nejsložitější metody jsou ve třídě související s hráči, protože se zde bere v potaz nejvíce okolností, spojených s množstvím HP nebo CP, které mají být objektu odebrané. Dědičnost tříd od třídy CharStatus.java je znázorněn na následujícím obrázku (Obrázek 8 - Dědičnost od CharStatus).



Obrázek 8 - Dědičnost od CharStatus

Šablony charakterů se inicializují pomocí konstruktoru. Mají vlastní privátní atributy a každý tento atribut presentuje jednu vlastnost charakteru (síla, obrana proti vodě, rychlost, obrana proti magii…). Je nutné říci, že zmíněné atributy jsou základní a následně upravované a ovlivňované, jak bylo popsáno v kapitole 3.4.



Obrázek 9 - Dědičnost šablon

Model – base

Balík obsahující informace o povolání hráčů, jejich pohlaví, rase a vedlejších povoláních. Informace o povolání jsou držena pomocí výčtu *ClassId*. *ClassLevel* je výčet určující, jaké přestupové úkoly jsou již hotové. Třída *PlayerClass* obsahuje úrovně jednotlivých povolání, jejich název a typ. Dále si ukládá sety jednotlivých povolání, pro pozdější kontrolu, aby hráč nemohl mít dvě povolání stejného typu.

Model – efekty (effects)

Balík obsahuje soubory, které říkají, jak je daný objekt ovlivněný. Jsou zde použity výčty, které obsahují hexadecimální kódy, které se propojují s herním klientem, tudíž je možno v herním klientu zobrazit, že je hráč například paralyzován nebo omráčen. *L2EffectTemplate* šablona efektu se používá pro uložení daného efektu ke konkrétní

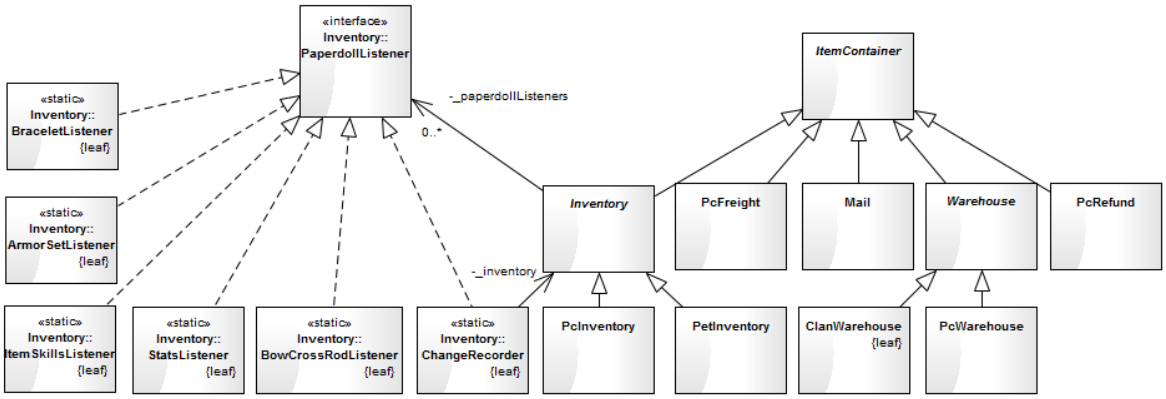
schopnosti. Všechny efekty jsou uložené ve výčtech a při načítání schopností, během nabíhání serveru, se k dané schopnosti přiřadí správný efekt.

Model – entity

Jednotlivé entity objektů v herním světě, které nejsou charaktery a představují pouze nějakou činnost nebo jsou nositele dat, nalezneme v tomto balíku. Některé z těchto činností jsou například bitva o hrad, tým proti týmu, duel mezi hráči. Převážná většina má svou vlastní vnořenou třídu, která implementuje *Runnable*. Potřebují to při plánování sledu událostí a spouštění metod v potřebný čas. Do tohoto balíku je zařazen sub balík se soubory obsahujícími kód bakalářské práce s invazí města. Celá invaze města je podrobně popsána včetně metod v kapitole 5 (Popis tříd a souborů invaze města).

Model – kontejnery věcí (itemconteiner)

Ve hře se nacházejí různé druhy vybavení, materiálů, nápojů, a ty jsou všechny udržovány a rozděleny podle místa svého uložení. Hráč má svůj vlastní inventář, ale také sklad kam si může nějaké věci schovat. Podobný sklad věcí má i každý klan, s tím že do něj může vkládat věci kdokoli, ale vybírat je smí pouze vůdce klanu. Další místo, kde mohou věci být uložené, je mezisklad, tam se věci dávají, pokud hráč potřebuje předat jiné postavě na stejném účtu své věci. Poslední kontejner je mail, protože ve hře lze posílat jako přílohu v mailu i věci. Všechny druhy kontejnerů dědí od třídy *ItemContainer* a rozšiřují ji. Třída inventář v sobě obsahuje vnitřní třídu s naslouchači (listenery), aby se dalo reagovat na změny vybavení inventáře. Když hráč vezme nebo odloží zbraň z ruky, zavolá se naslouchač k tomu určený a zjistí, zda měla zbraň nějaké speciální vlastnosti a ty následně odebere ze seznamu schopností hráče.



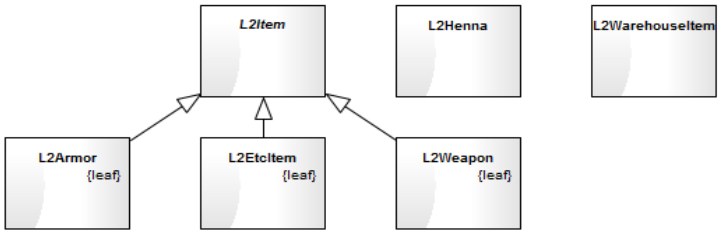
Obrázek 10 - Dědičnost od ItemContainer

Model – věci (items)

Věci v L2 jsou rozděleny podle více kritérií. Dělí se podle toho, zda je daná věc určena k oblečení, užívání nebo pro speciální užití s NPC. Věci určené k oblékání a užívání jsou definované ve vlastních třídách, rozdělených dle typu. Tyto třídy rozšiřují abstraktní třídu *L2Item* a dědí od ní metody. V abstraktní třídě jsou nadefinované hexadecimální tvary určující druhy, typy, materiály a úroveň věcí. Věci se dělí na zbraně, brnění a ostatní. Ve

zbraních je definován jaké speciální schopnosti mají, stejně jako jejich typ. U brnění je to jednoduší, zde se pouze ověří pozice, na kterou se v inventáři umísťuje a nastaví se typ. Ostatní jsou všechny věci, které lze mít v inventáři, ale neoblékají se na sebe. Jsou to materiály, kapsle, svitky, ale také věci zabalené do balíčků nebo sáčků. U těchto se nastavuje opět pouze typ věci.

Typ věcí se nastavuje z podbalíčku věcí. Obsahuje výčty jednotlivých typů zbraní, brnění a ostatních věcí. Specializované výčty implementují obecný výčet *L2ItemType*.



Obrázek 11 - Třídy věcí a jejich dědičnost

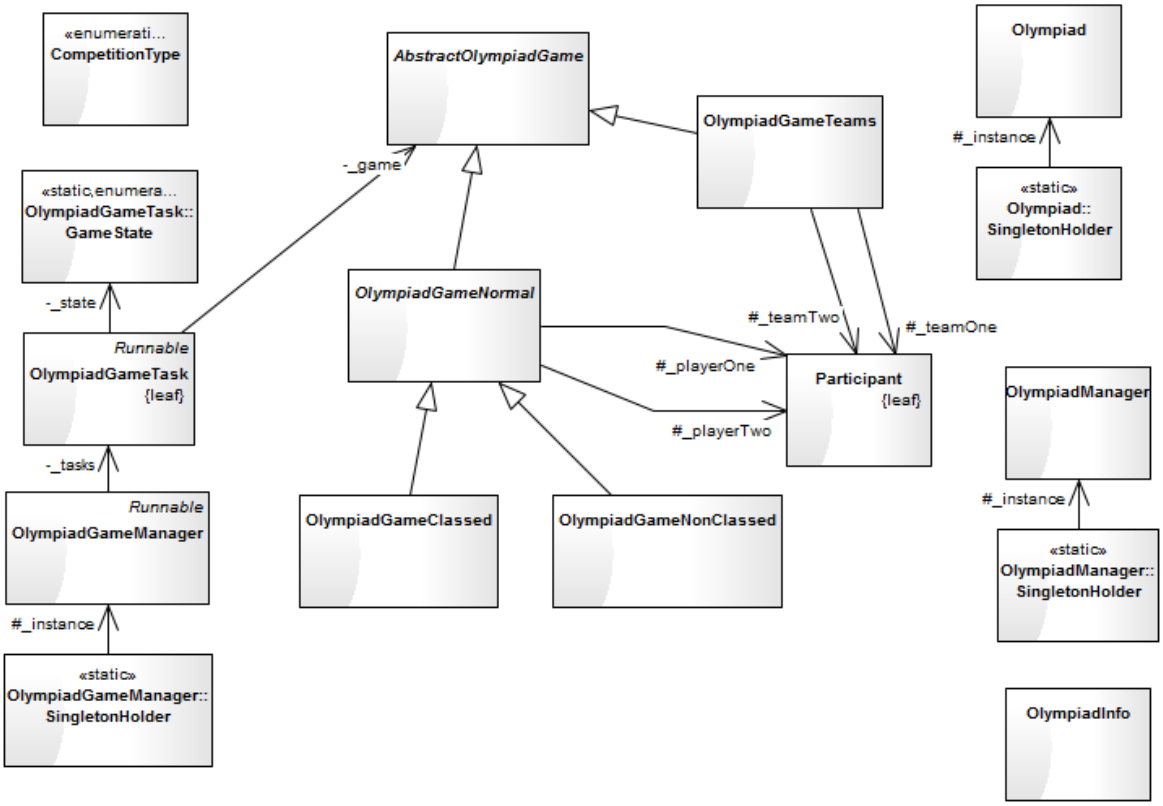
Model – olympiáda (olympiad)

Olympiáda je komplikovanější událost zabudovaná do herního jádra. Plánuje týdenní přidávání bodů hráčům, měsíční periodu, ale také jednotlivé souboje. Souboje na olympiádě jsou hráč proti hráči, nebo tým proti týmu. Oba typy soubojů mají vlastní třídu, dědící od abstraktní třídy *AbstractOlympiadGame*. Hráč proti hráči se dále dělí na souboje stejných nebo rozdílných povolání.

Přesnější závislosti mezi jednotlivými třídami jsou znázorněny v následujícím diagramu.

Účastník olympiády (participant) se dle typu hry přidělí do PvP nebo TvT a nastaví se strana, na které bude bojovat. Účastník se vytváří pomocí konstruktoru a to podle ID objektu ve hře, nebo pomocí instance *L2PcInstance*, reprezentujícího daného hráče.

Informace olympiády (OlympiadInfo) slouží pro uchování dat souboje. Tato data se při ukončení serveru, nebo při průběžném ukládání dat, vloží do databáze a na jejich základě se na konci měsíce rozhodne, jaký hráč bude za své povolání zvolen jako hrdina.



Obrázek 12 - UML Olympiáda

Model – úkol (quest)

V balíku jsou skriptovací třídy, založené na abstraktní třídě *ManagedScript*. Užívají se k uchování dat o jednotlivých úkolech, ale také se využívají při tvorbě událostí. Úkoly mohou být opakovatelné, jednorázové nebo denní. Tento údaj určuje třída *QuestState*, která je použita jako metoda v *Quest*.

Model – statistiky (stat)

V balíku se nalézá pět souborů. Dva jsou výčtového typu, jeden drží základní vlastnosti a druhý jednotlivé statistiky. Soubor s formulemi obsahuje metody pro výpočet poškození, šance na efekt, šance na kritický zásah, šance na smrtelný zásah, poškození elementární, výpočet úhybu před útokem a jiné. Finální třída *Calculator* je komparátor, starající se o porovnávání statistik, dle různých kritérií. Třída *Env* na druhou stranu předává hodnoty od instancí (*L2PcInstance, L2ItemInstance…*) do kalkulátoru, a pomocí metod *set, divide, multiply, add* a *sub* upravuje jejich hodnoty. Pokud přidáváme změny statistik do XML souborů, je nutno dodržovat pořadí výpočtu, aby se nestalo, že násobení bude před sčítáním.

Model – zóny (zone)

Svět Lineage 2 využívá různé druhy zón. Města jsou nastavena na zónu mírovou, to znamená, že v nich nelze útočit na jiné hráče. Je nutno říci, že všechna nastavení zón se dají upravit nebo přepsat, pokud má hráč potřebné povolení. Administrátor může nastavení zón obejít pomocí přepsání parametrů. Vyskytne-li se administrátor v zóně, kde nemá co

dělat, tak ho metoda v ověření zóny neteleportuje pryč, ale díky zjištění, že je administrátor, se z metody vrátí a další část ověřování neprobíhá. Mezi další druhy zón patří vězení, bojová zóna, zóna epických bossů, zóna bez obchodů, kde se nedají otevřít soukromé obchody, zóna se zákazem odlogování nebo restartování, bažina a voda. V základním L2J projektu je definováno 22 druhů těchto zón. Jednotlivé zóny se mohou navzájem překrývat. Nevylučuje se zóna mírová například se zónou vody. A pokud by se překrývala zóna mírová s bojovou, tak se bude brát v potaz zóna s vyšší prioritou.

4.1.2.11 Herní server – síť (network)

Veškerá výměna informací mezi hráčem a serverem probíhá přes zasílání paketů, s požadovanými vlastnostmi. V pravidelných intervalech se zjišťuje stávající poloha hráče, a dle potřeby se tento hráč přidá nebo odebere z listu známých objektů ostatních objektů. Při provedení akce na serveru se odešle z herního klienta paket s požadavkem na danou akci, a server po přijetí paketu, následně zpracuje a vyhodnotí, jak danou akci od hráče provede. Po vyhodnocení odešle zpět paket potvrzující provedení akce, a jak se má daná akce provézt. Jedná-li se o schopnost, hráči se zobrazí animace dané schopnosti, protože v paketu bylo zaslán příkaz pro použití schopnosti v klientu.

Obchoduje-li hráč s hráčem, tak neobchodují přímo mezi sebou, ale každý z nich obchoduje se serverem, a server poté distribuuje vyměněné věci mezi hráče, dle jejich dohody.

Lze říci, že bez tohoto balíku by se nedalo server pustit pro hráče. Nikdo by se nemohl připojit ani by nemohl nic dělat. Je zde i propojení s log-in serverem, vzájemně si zasílají informace o úspěšném přihlášení hráče, nebo herní server může odeslat i požadavek na zablokování hráče, nebo dokonce jeho celého účtu.

Herní server si ověřuje pakety zaslané hráči, aby se nestalo, že použijí hackovací nástroje. Dříve než se toto v projektu ošetřilo, stávalo se, že hráči mohli duplikovat věci, zasíláním klamných paketů. Server to nezjistil a odeslal hráči paket s přijetím věcí a vytvořil je na serveru. To je nyní ošetřeno pomocí třídy *BlowFishKeygen*. Jeho hexadecimální číslo je vkládáno do dvourozměrného pole o dvaceti řádcích. Vytváří se při spuštění serveru. Prvních 8 znaků každého řádku je náhodných a posledních 8 znaků je statických.

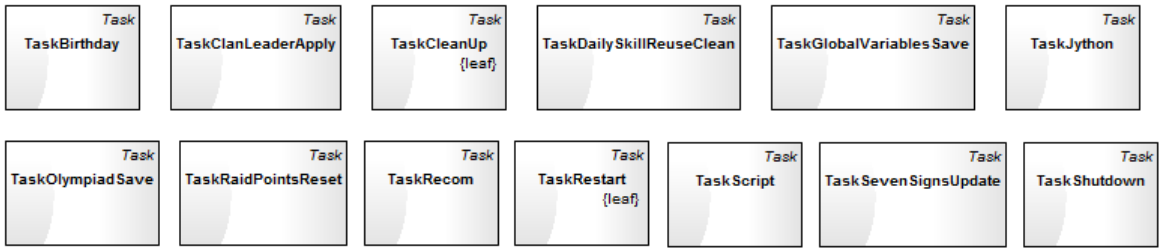
Pakety komunitního serveru zde nejsou rozebrány, protože komunitní server není součástí projektu a nejsou využívané.

4.1.2.12 Herní server – skripty (script a scripting)

Skriptovací třídy slouží jako prostředníci ke zpracování událostí. Mezi skriptovací soubory jsou zařazeny také naslouchače (listenery). Události ve hře mají své vlastní třídy, říkající jak se budou ovládat. Skripty se využívají pro běh událostí na serveru. Pokud hráč sebere věc ze země, použije se skript pro přidání této věci do inventáře. Všechny druhy těchto skriptů implementují rozhraní L2Event.

4.1.2.13 Herní server – manažer úloh (taskmanager)

Manažer úloh vkládá do plánovače úloh jednotlivé úlohy, dle přednastavených časů. Typy jednotlivých úloh jsou na obrázku 13. Správců úloh je více typů. Jeden z nich je správce listu známých objektů. Plánuje své obnovení dle konfiguračního souboru, kde se tato hodnota nastavuje v milisekundách. Kdyby se nastavila tato hodnota příliš nízká, obnovoval by se tento list příliš často a mohlo by dojít k poškození či selhání procesoru systému, kde server běží. Naopak příliš dlouhý interval by zapříčinil, že by o sobě jednotlivé objekty nevěděli, dokud by nestáli přímo u sebe nebo by nebyli již někde jinde.



Obrázek 13 - Seznam úkolů správce úloh

4.1.3 Log

Veškeré úkony serveru, které je potřeba sledovat se zapisují pomocí logovací instance do určených souborů. Třída loggeru je využívána ve všech souborech, ze kterých chceme získávat data do log souborů.

4.1.4 Log-in server

Log-in server je založen na ověřování uživatelského hesla a jména s databází. Pokud hráč uspěje, jsou mu odeslány pakety o úspěšném přihlášení a na herní server se odešle požadavek, na vpuštění hráče na jeho účet a odešlou se mu data charakterů vytvořených na účtu. Po vybrání postavy se vytvoří instance hráče na serveru a do té se vloží aktuální spojení vytvořené hráčem.

Log-in server je jednoduší konzolový program než herní server. Obsahuje pouze několik tříd, převážně spojených s komunikací a síťovými operacemi. Při přihlašování hráčů ověřuje, zda je účet založen. Pokud je povoleno automatické vytváření účtů a účet s daným jménem neexistuje, vytvoří se a uloží do databáze s heslem, které hráč použil při prvním přihlášení.

4.1.5 Status

Je-li herní server spojen s log-in serverem, udržuje se spojení pomocí vláken s nejvyšší prioritou. Přeruší-li se spojení, herní server vyhledává log-in server v pravidelných intervalech. Třída *Status* rozšiřuje vlákno (Thread). Obsahuje metodu pro vytvoření hesla, díky kterému se identifikují herní a log-in server vzájemně.

4.1.6 Nástroje (tools)

Jako nástroje pro ovládání, nebo nastavování serveru slouží instalátor databáze, konfigurátor nastavení serveru a správce účtů. Konfigurátor byl již popsán v kapitole 4.1.1.

Instalátor databáze se používá při instalaci serveru. Do databáze vytvoří potřebné tabulky a naplní je daty z SQL souborů.

4.1.7 Utility (util)

Utility pomáhají lepší správě kódu, jednoduššímu využívání tříd, nebo správě dat a log souborů. Najdeme zde nástroje, které upravují řetězce, místo enteru vyplňují „\n“, nástroje co čtou IP adresy a převádějí je do IPv4. Jsou zde filtry souborů, použité při načítání souborů s daty. Jsou zde rozšíření pro datové typy map, listů a náhodné třídy. Jako využívaný nastroj je zde i detektor zamrznutí serveru (deadlockdetector). Při zamrznutí serveru, se automaticky restartuje. Každý soubor zde obsahuje jeden nástroj, nebo rozšíření.

5 Popis tříd a souborů invaze města

5.1 TownInvasionManager.java

Tato třída slouží k plánování a celkovému spuštění události na serveru. Zhodnotí, zda je v konfiguraci povolena událost invaze města a podle toho se rozhoduje, má-li se vůbec událost plánovat. Byla-li by zakázána, událost se vypne a server se o ni již nezajímá a pouze se do log souboru zapíše informační sdělení, že je událost vypnuta.

Pokud by se Town Invasion měl spustit, tak se vezme instance třídy *TownInvasionEvent* a pomocí singleton holderu se inicializuje konstruktorem. Při nenačtení potřebných informací z databáze by se manažer rozhodl ukončit plánování a vypnul by událost na serveru.

Plánování se provádí voláním metody *ScheduleTownInvasion()*. Pomocí instance kalendáře se rozhodne o následujícím spuštění události. Ověřuje se, zda není již další den a plánuje se ve 24 hodinovém formátu. Při chybě zastaví plánování, vyhodí výjimku a zapíše chybu do logu.

5.2 TownInvasionTask.java

Třída implementující rozhraní *Runnable*, určená k načasování událostí ve správném sledu. Ovládá třídu *TownInvasionEvent* a určením stávající fáze Town Invasion volá potřebné metody v požadovaný čas. Je potřeba načasovat registraci na event, spawn jednotlivých vln a znovuspuštění události.

Při zavolání *TownInvasionTask ThreadPoolManagerem* se porovná stávající čas s naplánovaným časem události, pokud není tak se naplánuje znovu zavolání.

Dle *TownInvasionState* se volají metody *startReg(), eventStart(), eventStop(), spawnWave()* z instance *TownInvasionEvent*.

5.3 TownInvasionEvent.java

Třída ovládající celou událost od načtení potřebných dat z databáze, spouštění registrace a události po ukončení události a nastavení dalšího cyklu.

Celá instance je volána pomocí singleton holderu a při inicializaci ověřuje načtení dat z databáze. Při úspěšném načtení se nastaví stav události na *INACTIVE*, v opačném případě na *NOTLOADED*.

Stávající stav události určují boolean metody *isLoaded(), isRegistration(), isInActive(), isWaving(), isLast()*. Poslední stav je *END*, který se nastavuje při vytvoření poslední vlny NPC, jinak kdyby se nastavil jakýkoli jiný stav tak by se neprošlo *TownInvasionTask* až do poslední větve *else*, ale znovu by se spustila například registrace.

Po zavolání metody *startReg()* se spustí registrace, nastaví se stav na *REGISTRATION*, všem hráčům online se oznámí začátek události a čas, po který se mohou registrovat do události. Zároveň se vytvoří registrační NPC, které je jedna ze dvou možností jak se přihlásit. Druhá možnost je tzv. „*hlasovým*“ příkazem, začínajícím tečkou.

Po uplynutí registračního času se zavolá metoda *teleportPlayers()*, která ověří, zda jsou nějací hráči přihlášení do události a následně je teleportuje na místo, kam poběží všechna monstra z vytvořených vln. Toto místo je uložené ve třídě *TownInvasionTown*, a město je vybráno náhodně z listu měst načtených z databáze. Jakmile jsou hráči teleportováni, vytvoří se NPC, ke kterému budou hráči bránit přístupu ostatních NPC útočících na město. Po minutě se nastaví stav události na *WAVING* a první vlna monster je následně vytvořena. Tato minuta po teleportaci slouží hráčům k času na načtení potřebných dat hry a zobrazení objektů kolem nich, například budovy, ostatní hráči, stromy, sochy a jiné.

Vždy se ověřuje počet životů hráčů, a pokud by se stalo, že dosáhl nuly, tak se běh celé události zastaví, oznámí hráčům neúspěch a teleportuje je zpět do města. V případě ukončení invaze, a když zbývají hráčům životy, dostanou přednastavenou odměnu z konfiguračního souboru.

Při každé vlně se ověřuje, zda máme ještě následující vlnu a v případě poslední se nastaví stav na *LASTWAVE* a provede se vytvoření poslední vlny po oznámení hráčům. Pokud je vlna složená z *BOSS* monster tak se vytváří jeden boss na deset hráčů, naopak se na jednoho hráče vytvoří přednastavený počet monster. Během události nezáleží, zda hráč zabije pouze jedno nebo všechna monstra, ale důležité je ubránit město před invazí zabitím pokud možno všech vytvořených monster.

Monstra se chovají dle přednastavené umělé inteligence. Po vytvoření se začnou přesouvat směrem do vybraného města. Během cesty ověřují, zda nejsou v okolí charaktery, na které by mohly útočit. V případě útoku ověřují, zda se monstrum přesouvá od města nebo k městu a podle toho mění pozici. V případě tažení monstra směrem od města - hráč se jej snaží odvést dále od města, aniž by se ho snažil zabít - se odebere život a monstrum zmizí. Pokud zabije monstrum hráče, pokračuje dále v cestě do města, nemá-li další cíl.

Mrtví hráči mají nastavené znovuzrození, v případě smrti se oživí zpět ve městě po uplynutí nastaveného intervalu. Tento interval je nastaven v konfiguračním souboru spolu s dalšími možnostmi.

5.4 TownInvasionTown.java

Třída představující objekt města, do kterého budou proudit vlny monster a budou ho bránit hráči. Třída je po inicializaci finální a tudíž se její proměnné pouze získávají pomocí getrů. Proměnné se nastavují konstruktorem a jsou privátní. Nejsou statické, protože každé město má jiné proměnné, a tudíž je nesdílejí.

Jednotlivé proměnné v sobě uchovávají místo, kam hráči budou přemístěni při začátku události, místo, kde se objevují monstra a vyráží směrem k městu a název města, kde se invaze odehrává. Poslední je uchováno pro oznamovací účely ve hře a zápis do logu.

5.5 TownInvasionWave.java

Jednotlivé vlny jsou prezentovány touto třídou. Obsahuje data o monstru, jaké je pořadí vlny a zda vlna bude složená s monster nebo bossů.

Monstra nebo bossové jsou uvedení pouze jako číslo ID, pomocí kterého se dohledá příslušný objekt tabulce NPC. Do této tabulky se zapisují všechna NPC z databáze a poté se za pomocí ID dají dohledat a vytvořit.

5.6 L2PcInstance.java

Nejdůležitější třída, co se hráčů týká. Obsahuje všechna data o hráči a také všechny metody s hráčem spojené. Server a hráč používají většinu těchto metod ke komunikaci a ověřování, zda hráč je autorizován k vykonání činnosti ve hře. Zde budou popsány pouze metody a proměnné použité v práci.

Ve třídě L2PcInstance každého hráče se při inicializaci třídy vytvoří privátní boolean hodnota *_isInTownInvasion*, která nám určuje, zda je daný hráč přihlášen do invaze nebo ne. Pokud se hráč přihlásí, tak se pomocí setru nastaví hodnota na *pravda*. Odhlásí-li se, je hodnota nastavena na *nepravda*. Getr se používá pro určení HTML dialogu, který bude hráči zaslán a pro určení, zda má být hráč přidán nebo odebrán z události.

Jednotlivé HTML dialogy jsou zasílány jako pakety přímo hráčům. Pokud jsou v dialogu nějaké odkazy, tak se používají metody by-pasů k určení akce spojené s odkazem. U invaze města se používá by-pas na registraci a odhlášení hráče z události.

Je zde nastaven proces, co dělat když hráč zemře. Naplánuje se jeho oživení na místě, kam útočí monstra po nastaveném čase. Při úmrtí hráče se volá metoda *doDie()*, a pokud je hráč na události přihlášen, tak se zastaví tato metoda ihned po naplánování oživení. Kdyby se toto neudělalo, hráči by se zobrazil dialog s možností jít do města, hradu, klanové místnosti a jiné možnosti, podle toho, co vše hráčův klan vlastní, a mohl by jít okamžitě znovu bojovat.

Oživení se naplánuje do *ThreadPoolManageru* zavoláním metody *reviveToLocation()*, kde parametrem je lokace bodu, kam míří všechna monstra a kam jsou hráči při začátku teleportováni. Hráč se na toto místo přemístí a následně oživí.

5.7 L2TIMonsterInstance.java

Instance objektu jednotlivých objektů, prezentujících monstra. Třída rozšiřuje *L2MonsterInstance* o důležité metody určené pouze pro invazi města, jako třeba odměnění hráče, který se účastní události za své zabití a hlavně plánování chování monstra cestou do města.

V hlavním úkolu (*task)* se naplánuje přesun monstra do města a ověření, zda není v jeho okolí žádná instance hráče, na kterého by mohl zaútočit. V případě nenalezení hráče se začne pohybovat dále do města ke svému cíli.

V případě útoku na hráče se zjišťuje a ukládá momentální vzdálenost od cíle. Dosáhne-li vzdálenost větší vzdálenosti nežli je původní vzdálenost od cíle, tak se zavolá metoda *deleteMe()* k odstranění monstra a odebere se hráčům život. Toto je opatření proti tažení monster směrem od města aniž by hráči museli bojovat.

Hlavní úkol implementuje *Runnable*, a každý objekt který se objeví na serveru, se udržuje za pomoci podobných, naplánovaných úkolů v *ThreadPoolManageru*, volajících sebe sama a znovu naplánování v určeném intervalu.

Monstra berou všechna nastavení normálních monster a používají svá vlastní rozšíření, právě k účelům invaze města. Většina se bere pomocí konstruktoru, další z *override* metod *onSpawn()* nebo *doDie()*.

Jedna určuje provádění metod po objevení monstra a druhá následující akci, pokud je monstrum zabito. Po objevení se monstra se spustí nejprve ta samá metoda u rodiče a až poté se začnou provádět metody spojené s invazí města.

Konstruktor nastavuje typ monstra na *L2TIMonsterInstace*, vypíná na monstru možnost zasazení smrtelného úderu, zapne vidění monster i skrze tichý pohyb (Silent Move), které dělá hráče neviditelné pro agresivní monstra. Dále se nastaví, že monstrum používá běh místo normální chůze a vypne návrat na místo zrození, aby se nestalo, že se zasekne na půl cesty k cíli a nezačalo se vracet. Hlavním parametrem, který se nastavuje, je lokace, kam bude daná entita směřovat, a je brána z daného města pod útokem.

Chráněná metoda *moveTowardLoc()* slouží ke zjištění vzdálenosti od cíle a určení další akce. Pokud je monstrum již v okruhu kolem svého cíle, tak se zavolá mazací metoda a odebere se následně hráčům život. Když se nachází dále, tak se zjišťují hráči v okolí. Nenajde-li žádné, naplánuje další pohyb směrem do města, najde-li, tak začne útočit na nejbližší cíl.

Při zabití monstra se ověří, zda není mrtvý ten, kdo jej zabil. Pokud útočník mrtvý není, použije se metoda *giveReward()* a odmění hráče přednastavenými odměnami z nastavení.

6 Konfigurace invaze

V konfiguračním souboru *towninvasion.*properties se nastavují všechny možnosti události bez potřeby kódovat tato nastavení trvale v jádru. Je to výhodné pro rychlejší správu serveru bez nutnosti kompilace nového jádra. Ve hře je také příkaz pro administrátora ke znovunačtení nastavení, tudíž se dají tato nastavení měnit i za běhu serveru.

EnableTownInvasion

- Umožní zapnutí a vypnutí načtení Town Invasion při spuštění serveru.

TownInvasionLives

- Nastaví životy hráčů, po ubrání těchto životů událost končí neúspěchem.

TIKillReward

- Odměna za zabití jednoho monstra z vlny.

TIMobPerPlayer

- Počet monster vytvořených na jednoho hráče v jedné vlně.

TIRegistrationNpc

- Id číslo NPC, které se objeví pro registraci hráčů na event.

TIRegistrationTime

- Čas, po který se mohou hráči registrovat.

TIDeadPlayerRespawn

- Znovuzrození hráčů ve městě po uplynutém čase v sekundách.

TIRewards

- Odměny pro hráče pokud úspěšně ubrání město.

TIScheduleTimes

- Časové naplánování události. Jednotlivé časy jsou oddělené středníkem a psané ve 24 hodinovém formátu.

TIWaveTime

- Každá vlna se objevuje po časovém intervalu nastaveném zde. Čas je v minutách.

7 Stav události

Událost je rozdělena na různé stavy pomocí výčtové třídy TownInvasionState. Co jednotlivé stavy znamenají je popsáno níže.

INACTIVE

- nastavuje se v případě, že je událost neaktivní, ale načtená a připravená ke spuštění

REGISTRATION

- Aktivní během registrace.
- Pokud je nastaveno tak se u registračního NPC zobrazí dialog pro přihlášení na událost.
- Ověřuje se při použití hlasového příkazu (voice command).

WAVES

- Při vyvolávání vln.
- Říká TownInvasionTask, že se objevují vlny monster, kromě poslední.

LASTWAVE

- Při vyvolání poslední vlny.
- Určí, že po této vlně se má spustit ukončení události.

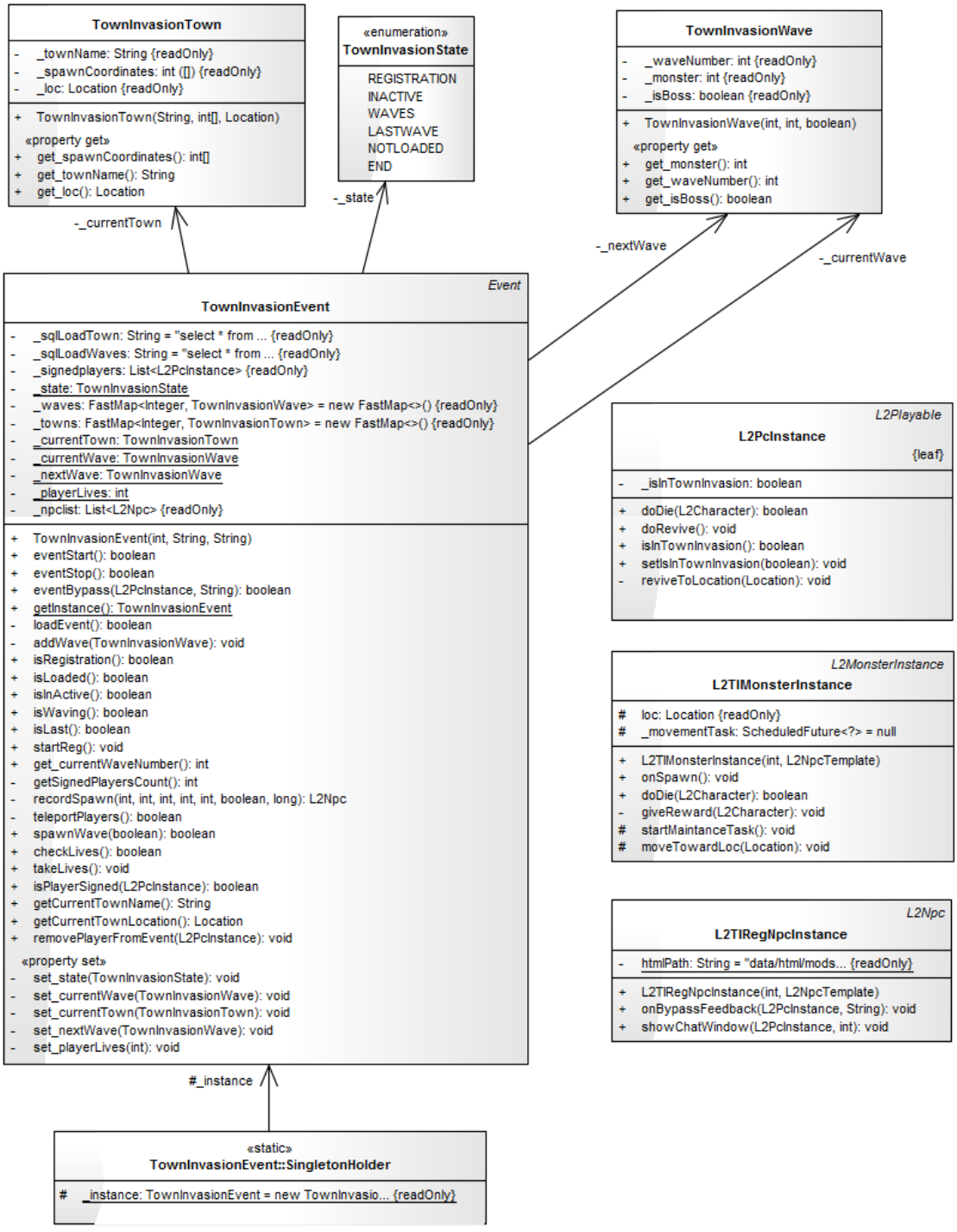
END

- Určuje ukončení události po poslední vlně.
- Použit je pouze k iteraci skrze rozhodování v TownInvasionTask.

NOTLOADED

- Inicializuje se při chybném načtení dat z databáze.

8 UML Diagramy



Obrázek 14 - UML diagram hlavní třídy události

Z obrázku výše (Obrázek 14) lze vidět, že máme pouze jednu instanci třídy *TownInvasionEvent*, která je uložena v *SingletonHoledru*. Tato třída má jako své atributy nastaveny dvě vlny (*_currenWave* a *_nextWave*), proto zde jsou dvě vazby na

TownInvasionWave. A jedno město ze třídy *TownInvasionTown*, ve kterém jsou údaje o právě obléhaném městě.

Dalšími atributy jsou dvě *String* proměnné použité k nastavení SQL příkazů. Tyto příkazy jsou vkládány do předpřipravených výpisů z databáze a následně prováděny, aby se uložily města a vlny do svých map.

Mapy měst a vln slouží k uchování všech načtených a roztříděných dat z databáze. U mapy měst je jako klíč vkládáno pořadí města, jak bylo načítáno z databáze a u mapy vln se jako klíč udává číslo dané vlny. Jako hodnota se do mapy vkládá celá třída *TownInvasionTown* nebo *TownInvasionWave*, podle toho jaká data právě zpracováváme.

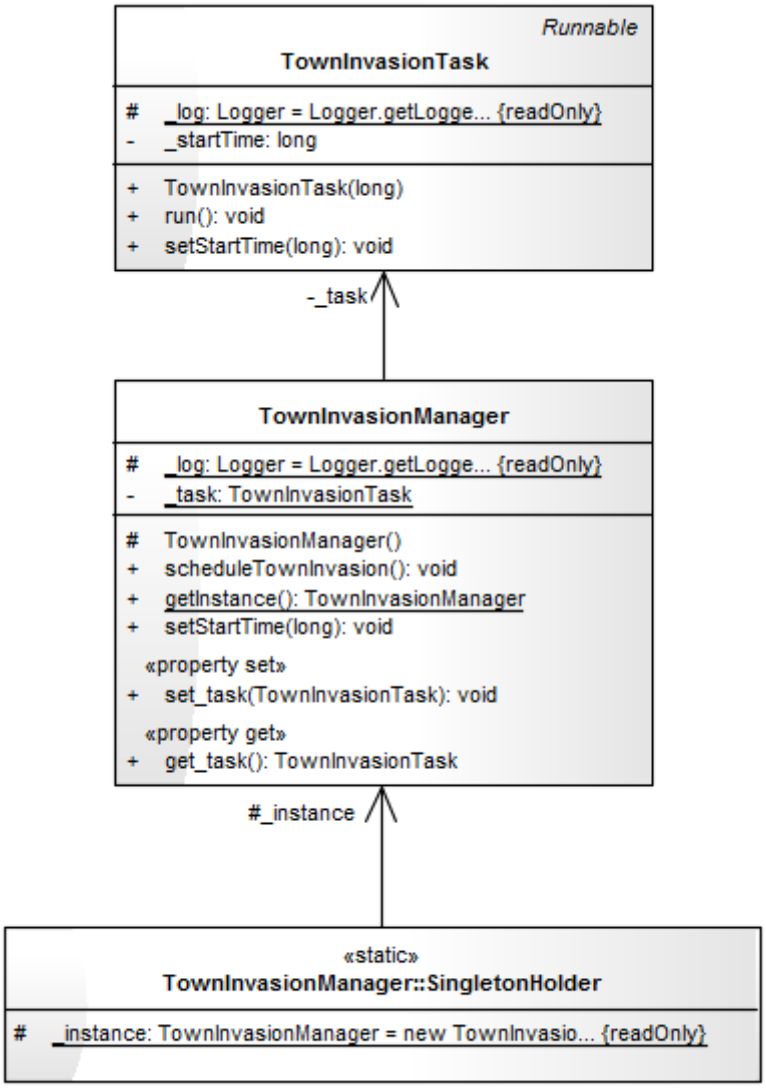
Pro uchování přihlášených hráčů je použit list entit *L2PcInstance*, kde každá položka představuje jednoho hráče. Do tohoto listu se přidávají hráči sami pomocí hlasového příkazu nebo prostřednictvím registračního NPC. Z tohoto seznamu se mohou následně i vymazat. Po ukončení události se tento list vyprazdňuje, aby byl na další spuštění události prázdný, protože, kdyby nám v něm zůstal hráč, který již třeba není ve hře, došlo by ke kolizi dat a nastala by chyba. Hráči se z události odebírají pomocí metody *removePlayerFromEvent()*, kde parametrem je odebíraný hráč. V této metodě se odebere hráč z listu přihlášených hráčů a nastaví se hodnota atributu *_isInTownInvasion* ve třídě *L2PcInstance* na *nepravda*. Hráč je rovněž z události smazán při odhlášení ze hry, k tomu dochází při provádění metody *logout()*, která postupně vymaže nebo vypne entitu a její návaznosti na ostatní součásti celého systému.

Každé NPC vytvořené během události se ukládá do listu NPC. Touto evidencí se dají po ukončení beze zbytku smazat zbývající NPC, která hráči nezabili, nebo také registrační NPC spolu s NPC, na které útočí monstra. Jednotlivá NPC se vkládají do listu pomocí metoda *recordSpawn()*, kam se vyplňují jako parametry ID daného NPC, souřadnice kde se vytvoří, směr, kterým po vytvoření hledí, a zda se má objevit náhodně kolem zadaných souřadnic nebo přímo na nich. K vytváření vln se používají náhodné souřadnice, aby se zvětšil prostor, ze kterého monstra útočí.

Stav události je nastaven hodnotou *TownInvasionState* podle momentální fáze. Je celkem šest možných stavů a každý, kromě posledního *END* si lze zjistit pomocí metody k němu patřící.

Množství životů, které hráčům stále ještě zbývá, udává *_playerLives*. Maximální hodnota se nastavuje z konfiguračního souboru (TownInvasionLives) a dokud není menší nebo rovna nule, tak se událost nezastaví. Když se monstrum dostane do blízkosti NPC, ke kterému během útoku na město útočí, odebere za pomoci metody *takeLives()* jeden život a ihned ještě v této metodě zavolá další metodu *checkLives()* a v případě odebrání posledního života se zastaví událost a hráčům se napíše, že neuspěli.

Jako vedlejší se ještě ukládá NPC, ke kterému se monstra blíží. Je zde uloženo kvůli ověření lokace monster oproti tomuto NPC, a pokud jsou již blízko, zavolá se již zmíněná metoda *takeLives*.



Obrázek 15 - UML diagram plánovače události

Z dalšího diagramu (Obrázek 15) je patrné, že manažer události si svou instanci drží také pomocí *SingletonHolderu*. Tím se zabrání, abychom na serveru měli více než jednu instanci této třídy a nemůže se stát, že by plánovaly událost dva nezávislé manažery.

Manažer v sobě drží jednu entitu třídy *TownInvasionTask*, se kterou následně pracuje a pomocí které plánuje celou událost.

Jako další tu máme vlastní log entitu, abychom mohli při chybě nebo oznámení zapsat potřebné informace do logu nebo konzole. Žádná jiná třída tento log nevyužije, protože v něm je zakódován název třídy manažera a informace psané do logu by nebyly pravdivé.

Jako veřejnou metodu využívá nastavení začátku události. Tento čas se nastavuje přímo do parametru *TownInvasionTask* ve třídě manažeru. Metoda se používá při manuálních příkazech administrátora nebo jeho pomocníka ve hře. Nastavením času na „nyní“ se obejde plánovač a spustí se událost okamžitě. Pokud bychom toto nenastavili, pouze by se ověřilo, zda již nenastal čas události a spuštění by se odložilo na správný čas.

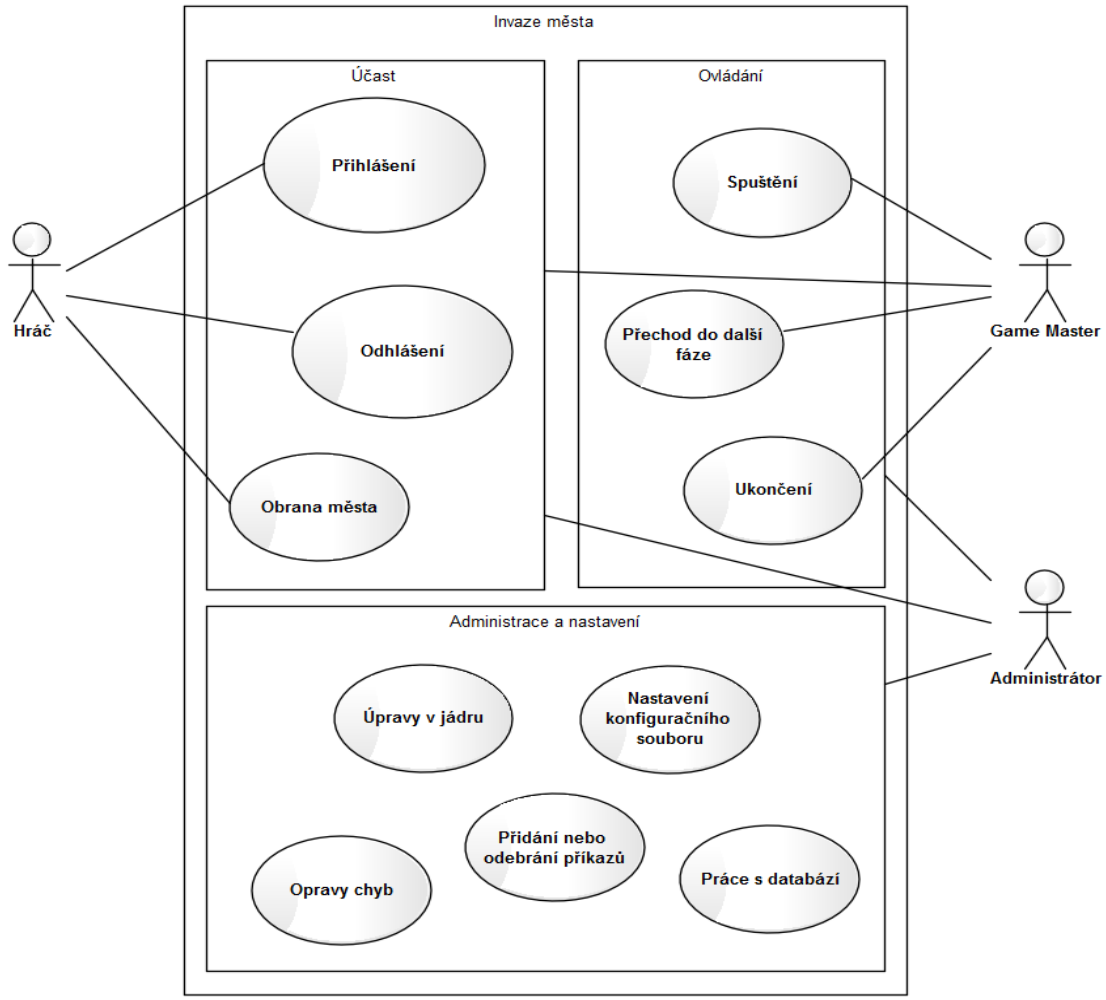
9 Use case diagram

Následující diagram (Obrázek 16) znázorňuje jednotlivé úkony náležící uživatelům dle jejich role na serveru.

Lze jasně říci, že nejnižší uživatel (*Hráč*) má pouze práva na účastnění se události, přihlášení a odhlášení se.

Pomocník administrátora (*Game master*) se stará o běh události ve hře a její případné ukončení. Všechny tyto úkony dělá přímo ze hry pomocí přednastavených příkazů. Ty nastavuje administrátor, včetně úrovně přístupu potřebné k jejich použití. Pokud by administrátor nechtěl, aby měl pomocník práva na užívání těchto příkazů, stačí mu nastavit potřebnou úroveň k užívání na vyšší, než je přístupová úroveň daného pomocníka. Defaultně je nastavena úroveň osm, což je administrátor.

Administrátor má přístup ke všem nastavením, konfiguračním souborům, databázi a kódu programu samotného. Stará se o plynulý běh události a odstraňuje případné chyby (bugy).



Obrázek 16 - Uživatelský use case diagram

10 SQL tabulky

Následující tabulky ukazují, jak se data ukládají do databázi MySQL. Hlavička tabulky udává typ a proměnou, jakou daný sloupec představuje.

V tabulce měst (Tabulka 1 - Příklad dat měst v databázi) lze vidět, že se nějaká města opakují. Je to z důvodu, že města mají ve většině případů více vstupů, a NPC mohou útočit z jakéhokoli směru. První sloupec je pouze k orientaci v databázi, ve hře nemá význam a je autoinkrementován.

Název města je zde určen pro jednodušší dohledání právě obléhaného města. Tento *String* se používá v oznámeních pro hráče a pro zápis do log souboru.

Impakt x, y a z jsou souřadnice NPC a místa kam míří vyvolané vlny nepřátel. Do třídy *TownInvasionTown* se ukládají jako *Location*, což je třída uchovávající souřadnice, směr kam se daný objekt natáčí čelem, ale i jiná, pro nás nedůležitá data.

Poslední tři sloupce slouží k uchování pozice, v okolí které se následně vytvářejí NPC útočící na město. Všechny tři hodnoty jsou uloženy do pole *Integerů* ve třídě *TownInvasionTown*.

Tabulka 1 - Příklad dat měst v databázi

town_id (int)	town_name (String)	impact_x (int)	impact_y (int)	impact_z (int)	spawn_x (int)	spawn_y (int)	spawn_z (int)
0	Aden	147462	27949	-2268	146996	36522	-3263
1	Goddard	147715	-55229	-2734	154336	-60190	-2676
2	Shuttgard	87345	-142851	-1316	87341	-136222	-2269
3	Goddard	147715	-55229	-2734	147547	-63678	-3435

Na další tabulce (Tabulka 2) bude popsán postup načtení jednotlivých vln z databáze do paměti hlavní instance události *TownInvasionEvent*. Jak lze vidět, v tabulce jsou tři sloupce. Jednotlivé sloupce jsou popsány v následujících odstavcích.

První sloupec reprezentuje pořadí vlny na události. Nejčastěji se nastavuje obtížnost jednotlivých vln od nejlehčích po nejtěžší, aby i slabí hráči měli šanci se zúčastnit a vyhrát nějaké odměny, alespoň za zabití monster. Je důležité, aby nebylo mezi čísly jednotlivých vln nějaké číslo přeskočeno, jinak by se nastavila vlna označená před touto mezerou jako poslední. Opět by bylo nejlepší řešení nastavit si v databázi autoinkrementaci tohoto sloupce.

Druhý sloupec ukazuje pomocí čísla, jaká instance NPC se má vzít z tabulky NPC. Toto NPC bude následně vytvářeno při jednotlivých vlnách útoku. Tabulka NPC je další tabulka obsahující všechna data o konkrétním NPC, jako například životy, síla, rychlost a jiné atributy, stejně jako udání typu NPC. V tomto konkrétním případě jsem v tabulce vlastních NPC vytvořil kopie vlků, u kterých je typ instance nastaven na *L2TIMonsterInstance*. Díky

tomu je jasné, jaké zvyklosti a chování bude tento vlk mít při vytvoření, zabití nebo doražení ke svému cíli.

Poslední je neméně důležitý. Udává počet jednotlivých NPC ve vlně, protože pokud je hodnota nastavena na „1“ tak je daná vlna označena jako *isBoss*, což vede k vytvoření pouze jednoho NPC na deset hráčů, minimálně však jednoho, kdyby bylo přihlášeno méně než deset hráčů. Opačný příklad vytváří monster tolik, kolik máme nastaveno v konfiguračním souboru (viz. TIMobPerPlayer) krát počet hráčů.

Tabulka 2 - Příklad dat vln NPC v databázi

wave_number (int)	monsterId (int)	is_boss (int)
1	50001	0
2	50002	0
3	50003	1
4	50004	0
5	50005	0
6	50006	1

11 Závěr

Cílem zadané bakalářské práce bylo implementovat do hry kompletní událost invaze města. Událost měla za úkol vytvořit NPC s vlastní AI, které budou útočit na město. Útočníci se měli objevovat ve vlnách po stanoveném intervalu a po dorazení na místo útoku se měli smazat a odebrat život hráčům, účastnících se invaze města. Dále se měli vytvořit SQL tabulky pro data invaze města v databázi serveru. Všechny události spojené s invazí města se měly vypisovat do konzole a log souboru. Událost má oznamovat hráčům svůj průběh a stav.

Naprogramované entity, třídy a jejich metody odpovídají zadání. Program vypisuje do souboru s logem potřebné údaje o průběhu události. Během události se ověřuje její stav a plánuje se další krok do manažeru vláken. Při objevení vln se dané objekty dají do pohybu a v případě, že se v jejich okolí objeví hráč, zaútočí na něho. Událost je plně nastavitelná pomocí konfiguračního souboru. Tímto se splnilo zadání práce.

Událost nebyla testována na normálním serveru, ale pouze v lokálním počítači za postavu administrátora. Testování proběhlo bez chyb a do logu se vypisovali všechny potřebné údaje. Administrátor ve hře mohl používat příkazy, připravené pro tuto událost, vizuálně mohl potvrdit oznámení pro hráče a přihlašoval se do události pomocí registračního NPC. Při ukončení události nebo zabití NPC invaze obdržel nastavenou odměnu.

V porovnání s ostatními událostmi je tato více komplikovaná, ale dala by se dále vyvíjet. Lze zabudovat více možností invazí, například invazi více měst najednou, invazi města z více směrů. Z nedostatku času se nestihlo práci plně otestovat a vylepšit.

Bakalářská práce mi pomohla hlouběji proniknout do programovacích technik jazyka Java, rozšířil jsem své znalosti s tímto jazykem a seznámil jsem se více s projektem L2J. Díky složitosti tohoto projektu jsem otestoval své schopnosti, orientovat se v kódu, který je na vysoké úrovni a v tak velikém rozsahu. Na této práci jsem si otestoval práci s vlákny a jejich plánováním.

Literatura

[1] L2World: Lidská povolání v L2. *L2World* [online]. 2008 [cit. 2014-05-13]. Dostupné z: http://www.l2world.com.pl/ludzie-2/

[2] Lineage II Wiki: Races. *Lineage II Wiki* [online]. 2011 [cit. 2014-05-13]. Dostupné z: http://lineageii.wikia.com/wiki/Races

[3] Lineage 2 Encyclopedia. *Lineage 2 Encyclopedia* [online]. 2013 [cit. 2014-05-13]. Dostupné z: http://l2wiki.com/

[4] L2J Server Project. *L2J Server Project* [online]. © 2004-2009 [cit. 2014-05-13]. Dostupné z: http://www.l2jserver.com/

[5] L2J Server Project: Eclipse and SVN getting started guide. *L2J Server Project* [online]. © 2004-2009 [cit. 2014-05-13]. Dostupné z: http://www.l2jserver.com/2013/04/eclipse-and-svn-getting-started-guide/

[6] L2J Server Project: Netbeans for L2J - Getting started tutorial. L2J Server Project [online]. © 2004-2009 [cit. 2014-05-13]. Dostupné z: http://www.l2jserver.com/2013/08/netbeans-for-l2j-getting-started-tutorial/

[7] ECKEL, Bruce. Thinking in Java. 4th ed. Upper Saddle River, NJ: Prentice Hall, c2006, 1482 p. ISBN 978-013-1872-486.

[8] LAFORE, Robert a Robert LAFORE. Data structures. 2nd ed. Indianapolis, Ind.: Sams, c2003, xix, 776 p. ISBN 06-723-2453-9.

[9] GILFILLAN, Ian a Robert LAFORE. Mastering MySQL 4. 2nd ed. London: Sybex, c2003, xxxiii, 729 p. ISBN 07-821-4162-5.

A. Příloha – Zdrojový kód metody eventStart()

```
@Override
public boolean eventStart()
{
 if (!isLoaded())
 {
  return false;
 }

 set_currentTown(_towns.get(Rnd.get(_towns.size())));
 if (_currentTown == null)
 {
  eventStop();
  return false;
 }

 if (!_waves.isEmpty())
 {
  set_currentWave(_waves.get(1));
  set_nextWave(_waves.get(2));
 }
 else
 {
  eventStop();
  return false;
 }

 _log.info("Choosen " + getCurrentTownName());

 set_playerLives(Config.TI_LIVES);

 if (teleportPlayers())
 {
  _protecting = recordSpawn(Config.TI_IMPACT_NPC_ID, _currentTown.get_loc().getX(),
_currentTown.get_loc().getY(), _currentTown.get_loc().getZ(), 0, false, 0);

  Announcements.getInstance().announceToAll("Town Invasion: Teleporting signed players to "
+ getCurrentTownName() + ".", true);

  Announcements.getInstance().announceToAll("Town Invasion: 1st wave will come in 1 minute.
Prepare yourself!", true);

  set_state(TownInvasionState.WAVES);
 }
 else
 {
  Announcements.getInstance().announceToAll("Town Invasion: Nobody signed, event has been
canceled.", true);
  eventStop();
  return false;
 }

 _log.log(Level.INFO, "Town Invasion: Event started.");
ThreadPoolManager.getInstance().scheduleGeneral(TownInvasionManager.getInstance().get_task(
), 60000);
 return true;
}
```

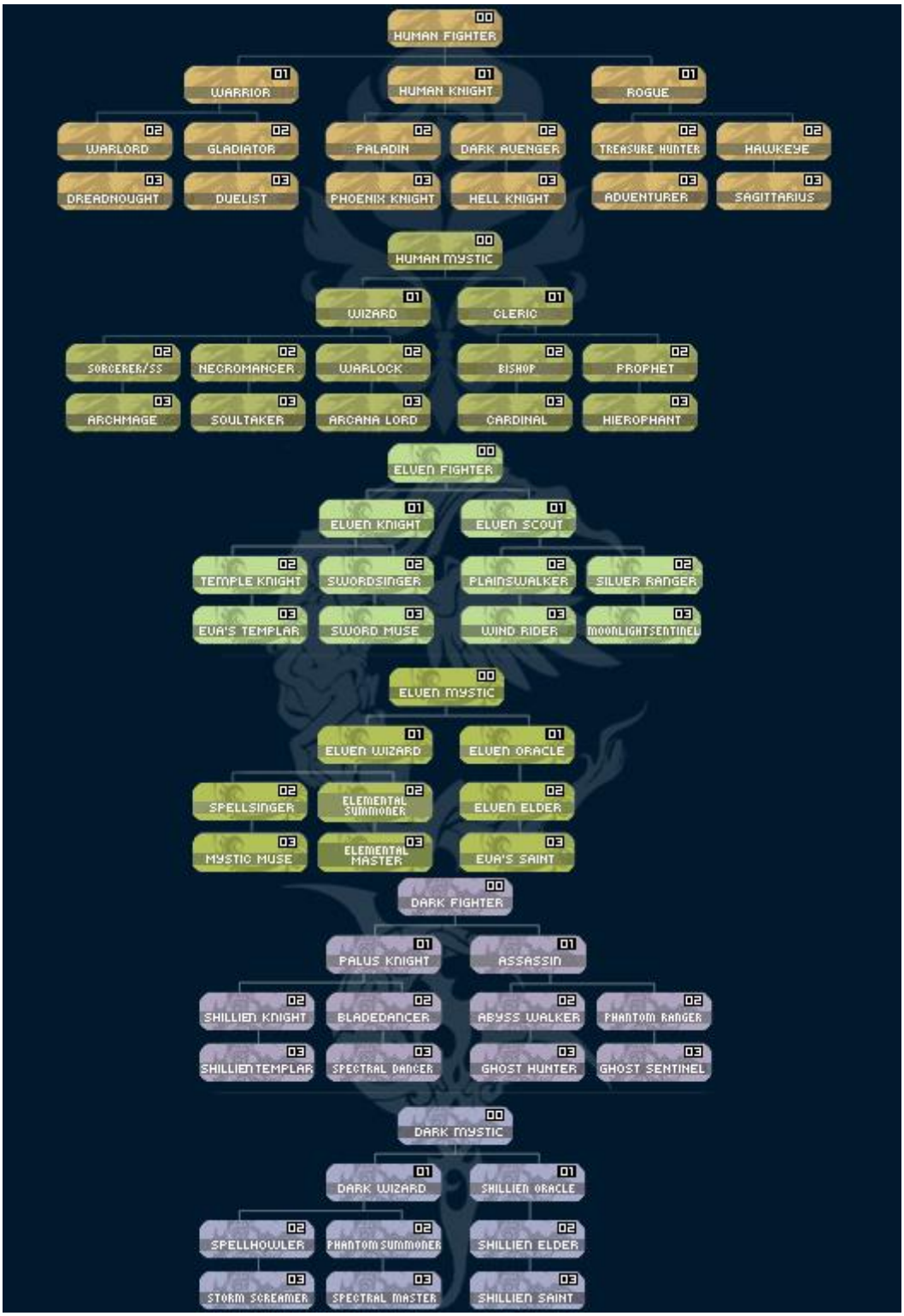
B. Příloha – Zdrojový kód třídy TownInvasionTask

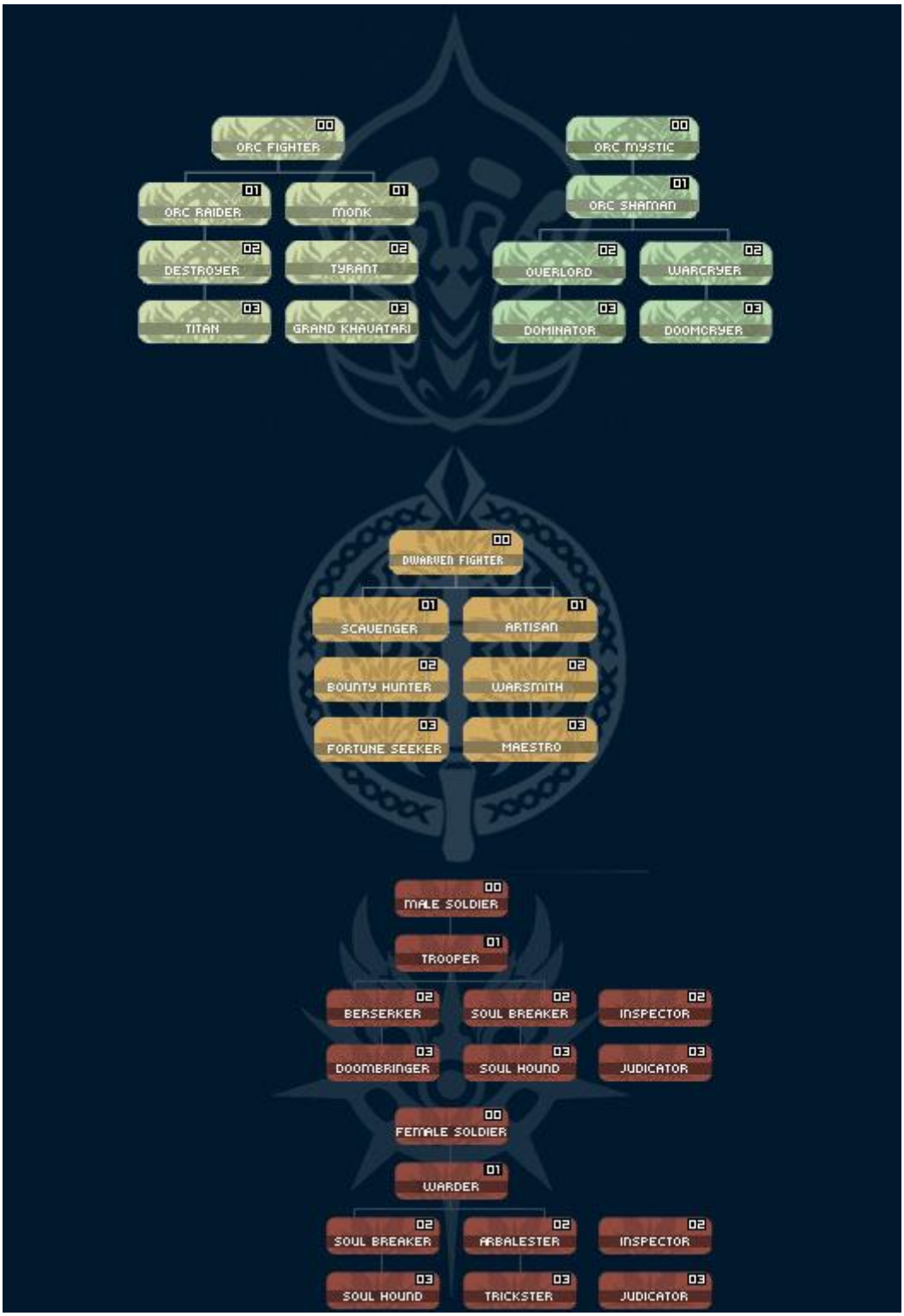
```
public class TownInvasionTask implements Runnable
{
protected static final Logger _log=Logger.getLogger(TownInvasionTask.class.getName());
 private long _startTime;

 public TownInvasionTask(long time)
 {
  _startTime = time;
 }
 @Override
 public void run()
 {
  int delay = Math.round((_startTime - System.currentTimeMillis()));

  if ((delay > 0) && !ThreadPoolManager.getInstance().equals(this))
  {
   ThreadPoolManager.getInstance().scheduleGeneral(this, delay);
  }
  else
  {
   // start registration
   if (TownInvasionEvent.getInstance().isLoaded() &&
TownInvasionEvent.getInstance().isInActive())
   {
    TownInvasionEvent.getInstance().startReg();
   }
   // start event
   else if (TownInvasionEvent.getInstance().isRegistration())
   {
    TownInvasionEvent.getInstance().eventStart();
   }
   // spawn wave
   else if (TownInvasionEvent.getInstance().isWaving() &&
TownInvasionEvent.getInstance().checkLives())
   {
    TownInvasionEvent.getInstance().spawnWave(false);
   }
   // spawn last wave (most likely boss)
   else if (TownInvasionEvent.getInstance().isLast() &&
TownInvasionEvent.getInstance().checkLives())
   {
    TownInvasionEvent.getInstance().spawnWave(true);
   }
   // end event and schedule next start
   else
   {
    TownInvasionEvent.getInstance().eventStop();
   }
  }
 }
 public void setStartTime(long time)
 {
  _startTime = time;
 }
}
```

C. Příloha – Stromy ras a jednotlivích povolání





Obrázek 17 - Stromy povolání jednotlivých ras – Zdroj [1]