VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Strategická hra pro více hráčů v prostředí Internetu

Multiplayer Strategic Game over Internet

2015 Patrik Klement

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Patrik Klement**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Strategická hra pro více hráčů v prostředí Internetu
Multiplayer Strategic Game over Internet

Zásady pro vypracování:

Cílem práce je vytvořit hru souboje osad zasazeného do historického světa oplývajícího magií. Hra bude založena na principu strategické hry v reálném čase v prostředí Internetu. Hra bude vytvořena v jazyce Java a umožní výstavbu budov, vytváření více druhů bojových jednotek, hru více hráčů.

Hra bude umožňovat:
1. Editaci herní mapy.
2. Výstavbu budov, zlepšování bojeschopnosti jednotek.
3. Vytváření více druhů bojových jednotek, které umožní boj na dálku, z blízka a využití magie.
4. Hru více hráčů v prostředí Internetu.
5. Bojové jednotky budou podporovat jednoduché autonomní příkazy (útoč, hlídkuj, ...).
6. Hru proti jednoduché umělé inteligenci.
7. Program umožní připojení do současně vyvíjeného portálu her v jazyce Java.

Práce bude obsahovat:
1. Implementaci výše popsané strategické hry.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2015 .....

Rád bych na tomto místě poděkoval vedoucímu Ing. Davidu Ježkovi Ph.D. za jeho cenné rady a pomoc při realizaci této práce. Dále bych chtěl poděkovat rodině, přítelkyni a všem přátelům za jejich podporu.

Abstrakt

Cílem této bakalářské práce je vytvořit Strategickou hru, která je zařazena do historického světa s pomocí kouzel. Hra je vytvořena v programovacím jazyce Java pomocí využití 3D herního enginu s názvem jMonkeyEngine 3.0. Umožňuje hru dvou hráčů proti sobě a také v případě odpojení jednoho z nich hru proti lehké umělé inteligenci. První část této práce obsahuje úvod do tématu, technologie použité při tvorbě, popis hry a další důležité informace, které byly potřebné k úspěšnému dokončení této hry. Druhá část je zaměřena na průběh tvorby této hry, její implementaci a další informace spojené s touto prací.

Klíčová slova: strategická hra, jMonkeyEngine 3.0, Java, historický svět, magie, více hráčů

Abstract

The aim of this thesis is to create a strategic game which is included into historical world with magic. The game is created by Java programming language with using 3D game engine called jMonkeyEngine 3.0. The game allows to play two players against each other and also the game between easy artificial intelligence if one of the players will disconnect. The first part contains an introduction to the topic, the technology used in the creation of the game, description of the game and other important information which were necessary to complete this game successfully. The second part is focused on the process of making this work, its implementation and other information related to this work.

Keywords: strategic game, jMonkeyEngine 3.0, Java, historical world, magic, multi-player

Seznam použitých zkratek a symbolů

BSD	–	Berkeley Software Distribution - licence pro svobodný software
LWJGL	–	Lightweight Java Game Library
GUI	–	Graphical User Interface - Grafické uživatelské rozhraní
jME3	–	jMonkeyEngine 3.0
XML	–	Extensible Markup Language - obecný značkovací jazyk
API	–	Application Programming Interface - programové rozhraní aplikace

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Cílem této práce je vytvořit real-time strategickou hru, která je založena na principu historického světa s využitím kouzel. Ve hře se vyskytuje více typů jednotek a budov. Celá hra spočívá v tom, že je potřeba zničit základnu protivníka, která je v našem případě hrad. Při úspěšném zničení této hlavní budovy dojde k vítězství a ukončení hry. Pro zdárné vítězství je k dispozici ve hře několik vylepšení, ale je potřeba i vymýšlet úspěšnou taktiku a předpovídat tahy protivníka, aby nedošlo ke zničení naší základny. Hra se odvíjí v prostředí Internetu, kde hrají dva hráči proti sobě. V případě odpojení jednoho z hráčů, je tento hráč nahrazen umělou inteligencí, která se pokusí o zdárné zničení protivníka.

Inspirací k této hře mi byly úspěšné strategické hry, které obsahovaly základní tématiku a stejný princip celého průběhu hry. Tímto tématem se budeme zabývat později, kde rozvinu podobnost této hry, její princip a další faktory, které jsou stejné či podobné.

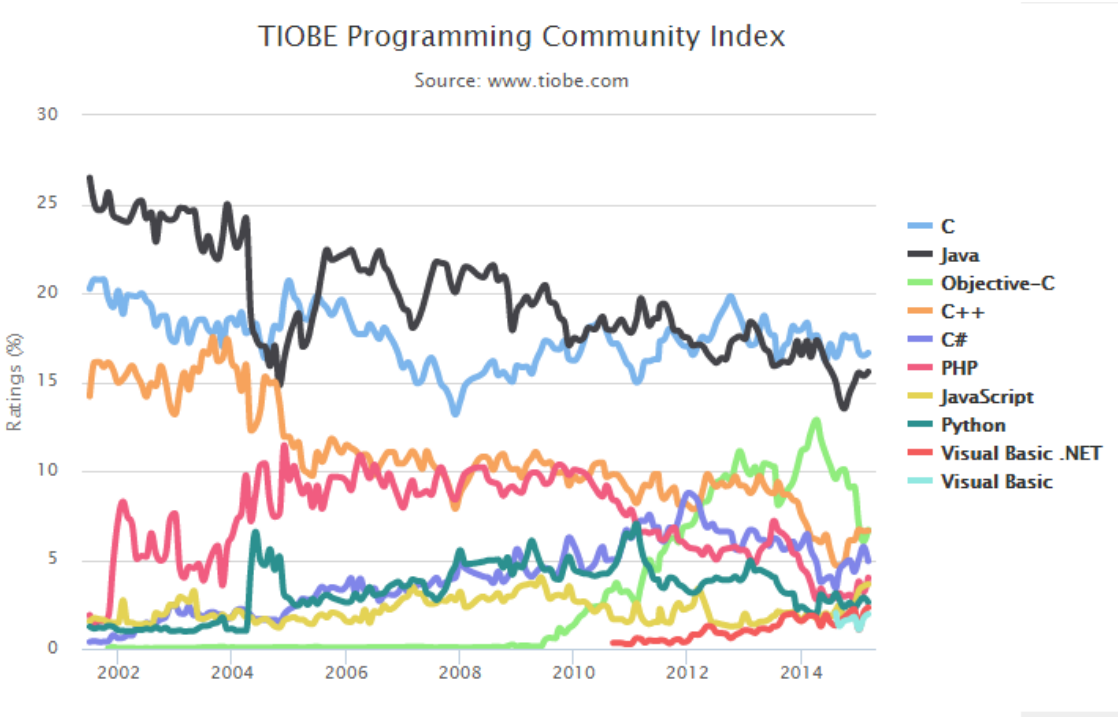
Celá práce je vytvářená v jazyce Java pomocí využití herního enginu jMonkeyEngine 3.0. Celá hra je rozdělená do dvou části. První část obsahuje klienta a druhá část je server.

V první části celého dokumentu se budeme zabývat popisem celé hry. Ukážeme si typy jednotek, budov, různá vylepšení a dále si ukážeme také jejich rozdílné schopnosti. Vysvětlíme si jak hru spustit, popíšeme si její pravidla, které jsou potřebné ke zdárnému dokončení hry a k jejímu vítězství. Hra obsahuje také editor map, který můžeme použít k vytvoření vlastní mapy.

2 Základní informace

2.1 Programovací jazyk Java

Java je objektově orientovaný programovací jazyk, vyvinut firmou *Sun Microsystems*. Jedná se o druhý nejpoužívanější programovací jazyk světa podle **Tiobe indexu** [1]. Jednotlivé druhy a pozice programovacích jazyků jsou zobrazeny na Obrázku 1. Java vyniká svou přenositelnosti, která dosahuje vysokého rozsahu práce na různých systémech.



Obrázek 1: Nejpoužívanější programovací jazyky světa

První oficiální verze jazyka vyšla 26. srpna 1996. Současně nejnovější verze vyšla v roce 2014 pod názvem **Java SE 8**.Více informací v [2].

Java používá automatickou správu paměti. V případě, že na daný objekt neukazují žádné odkazy, se paměť uvolní. Programátor určuje kdy bude objekt vytvořen a o uvolnění paměti se stará **garbage collector**.

V současné době je tedy **Java** jedním z nejpoužívanějších jazyků k tvorbě programů a her. Pro tento jazyk existuje spoustu frameworku a herních enginů.

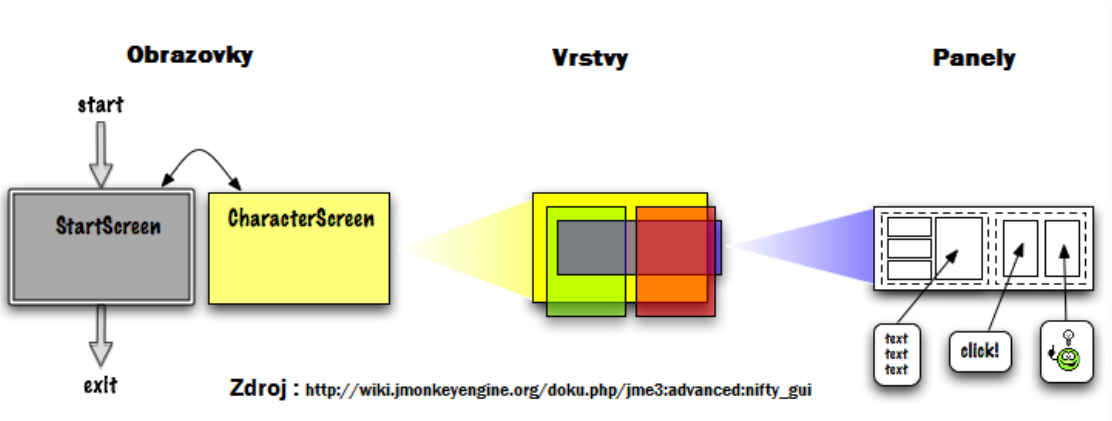
2.2 Herní 3D engine jMonkeyEngine 3.0

jMonkeyEngine [3] je výkonné, robustní rozhraní pro programování Java aplikací. Autoři tohoto enginu se inspirovali knihou **3D Game Engine Design** od *Davida Eberlyho*. Aktuálně je nejnovější verze 3.0. Celý tento projekt je s otevřeným zdrojovým kódem (

Open Source), který je distribuován pod BSD licencí. V současné době je používán vykreslovací systém LWJGL. LWJGL je knihovna, která je napsaná v jazyce Java. Využívá knihovny OpenGL a OpenAL. Podporuje také široké spektrum grafických efektů. Obsahuje vlastní editor, který slouží k vytváření prostředí, kde si můžeme volitelně nastavit terén, vytvářet pohoří, přidávat vodu, oheň, stíny, odlesky a další speciální efekty, které nám tento engine nabízí.

2.3 Nifty GUI

Nifty GUI [4] je knihovna, která je naprogramovaná v jazyce Java a slouží k vytváření uživatelského rozhraní. Hlavní využití spočívá v tom, kdy chcete vytvořit tlačítka, textová pole, seznamy a další prvky potřebné ke tvorbě GUI. Nifty GUI (*de.lessvoid.nifty*) je velmi dobře integrován do jMonkeyEngine 3.0 přes balíček s názvem *com.jme3.niftygui*. Celé GUI může být definováno pomocí XML a kontrola probíhá dynamicky pomocí kódu. Všechny potřebné knihovny a materiály jsou již obsaženy v jME3.



Obrázek 2: Nifty GUI koncept

Na Obrázku 2 lze vidět, že se Nifty GUI skládá ze tří základních částí :

1. **Obrazovky** - skládá se z jedné nebo více obrazovek. V současnou chvíli může být zobrazena pouze jedna obrazovka. Tyto obrazovky jsou ovládány třídou.

2. **Vrstvy** - skládá se taktéž z jedné nebo více vrstev. Vrstvy jsou kontejnery, které zahrnují zarovnání obsahu (*vertikálně, horizontálně nebo střed*) . Vrstvy se mohou překrývat, ale nesmějí být do sebe vnořeny.

3. **Panely** - vrstvy obsahují panely. Panely jsou opět kontejnery, které se stejně jako vrstvy starají o rozmístění prvků. Nemohou se překrývat, ale mohou být do sebe vnořeny. Panely tedy zahrnují obrázky, text nebo například ovládání (tlačítka atd.) .

Jak jsem již zmínil, samotný návrh je uložen v XML souboru. O jednotlivé akce se poté stará programátorem napsaný kód, který na základě stisknutí tlačítka vykoná akci. Nejedná se pouze o ovládání prvků, ale také samozřejmě o zobrazení jednotlivých obrazovek a další práci s GUI. Tato technologie samozřejmě taktéž poskytuje tvorbu HUD. Více informací detailně popsáno vč. ukázky kódu v [5].

```
<?xml version="1.0" encoding="UTF-8"?>
<nifty  xmlns="http:// nifty -gui.sourceforge.net/ nifty -1.3.xsd" xmlns:xsi="http :// www.w3.org/2001/
      XMLSchema-instance" xsi:schemaLocation="http://nifty-gui.sourceforge.net/nifty-1.3.xsd␣
      http://nifty-gui.sourceforge.net/nifty-1.3.xsd">
  <screen id="start">
    <!-- ... -->
  </screen>
  <screen id="hud">
    <!-- ... -->
  </screen>
</ nifty >
```

Výpis 1: Ukázka XML souboru

3 Popis hry

Tato kapitola popisuje hru, její pravidla, princip a další věci, které jsou ve hře použity.

Hra je tedy RTS (Real-time strategy), což znamená, že veškeré hráčové úkony a počítačem řízené funkce probíhají v reálném čase. Jelikož se jedná o strategickou hru, ovládáme zde větší skupinu objektů a manipulujeme s nimi po hrací ploše. Důležitou strategii k vítězství je myšlení a plánování akcí. Celá hra je vytvořená ve **3D**. Na Obrázku 3 lze vidět vzhled hry. Můžeme vidět hráčovu vesnici, která už je v pokročilé fázi budování.

Obrázek 3: Ukázka hráčovy vesnice

3.1 Akce ve hře

Hra se skládá z několika typů akcí jako je například **výstavba budov, tvorba jednotek a vylepšení bojeschopnosti jednotek.**

3.1.1 Výstavba budov

Hráč má k dispozici několik druhů budov, přičemž každá z nich má jiný typ jednotek a vylepšení. Ve hře se nacházejí tyto budovy :

- **Altar** - základní budova, která se nachází na Obrázku 3 vpravo nahoře. Má za úkol vytvářet bojovou jednotku s názvem Soldier. Budova má 500 jednotek životů a její výstavba stojí 500 zlatých.

- **Castle** - hlavní budova celé hry. Tato budova má za úkol vytvářet bojovou jednotku pod názvem Zombie. Dále je úkolem této budovy vytvářet jednotky, které jsou určené pro těžbu zlata a přijímat jejich vytěžené zlato. Tyto jednotky se nazývají pracovníci (Worker). Pokud hráč přijde o tuto budovu, prohraje. Budova má 1500 jednotek životů a hráč si ji obstará za 1000 zlatých. Nachází se na Obrázku 3 vlevo nahoře.

- **Range** - budova, která slouží k vytváření pokročilé jednotky s názvem Ranger. Hráče vyjde na 750 zlatých a životnost budovy je 800 jednotek. Budova je postavena na Obrázku 3 vlevo dole.

- **GoldTrend** - tato malá budova hraje ve hře významnou roli. Bez této budovy se hráč neobejde a to z toho důvodu, že slouží k těžbě zlata. Hráč si jí musí postavit poblíž zlatého dolu a k této budově poté chodí pracovníci, kteří těží zlato. Bez této budovy se zlato těžit nebude a výroba pracovníku bude zbytečná. Stojí pouhých 250 zlatých a životnost budovy je 250. Na Obrázku 3 je budova postavena poblíž zlatého dolu.

- **Laboratory** - pokročilá budova, která má za úkol vylepšovat jednotky. Hráč si ji může postavit za 5000 zlatých a její životnost je 2000 jednotek. Laboratoř má za úkol zvýšit poškození útoku daného typu jednotek. Vylepšení vypadá takto :

  1. Vylepšení jednotky Soldier - zvýšení útoku o 5.
  2. Vylepšení jednotky Zombie - zvýšení útoku o 10.
  3. Vylepšení jednotky Ranger - zvýšení útoku o 20.

  Tato budova je postavena na Obrázku 3 vpravo dole.

3.1.2 Tvorba jednotek

Ve hře je několik druhů bojových jednotek a to jak na útok na blízku tak i na dálku. Každá jednotka se liší počtem životů, poškozením a dalšími vlastnostmi. Jedná se o tyto typy vojáků :

- **Soldier** - základní jednotka na blízku. Disponuje nízkým útočným poškozením a to 10 jednotek za útok. Cena jednotky je 100 zlatých a životnost taktéž 100 jednotek životů.

- **Zombie** - lepší, ale taktéž dražší jednotka pro útok na blízku. Její útočné poškození činí 25 jednotek za útok a životnost je 300. Tuto jednotku lze pořídit za 200 zlatých.

- **Ranger** - nejvyspělejší jednotka, která má útok na dálku. Má také možnost vyvolat kouzlo, které způsobí zranění v okolí od vyvolání a taktéž efekt zvaný odstrašení, při kterém dojde k pohybu jednotek od středu vyvolání kouzla. Tato jednotka stojí 250 zlatých a má 500 jednotek životů.

- **Worker** - pracovník, je slabá jednotka, která má velmi nízký útok a životnost. Slouží pouze k těžbě zlata. Tato jednotka unese 100 zlatých a nosí je zpět do hlavní budovy. Jeho životnost je pouhých 50 jednotek a útočné poškození 2 jednotky za útok. Cena této jednotky je 50 zlatých.

Každá z jednotek má tři typy akcí :

1. **Jdi** - základní příkaz pro přesun jednotky z pozice na pozici.
2. **Zaútoč** - příkaz, který se vykoná po kliknutí na nepřátelskou jednotku.
3. **Hlídkuj** - po zvolení tohoto příkazu si hráč vybere cílový bod a jednotka se bude pohybovat mezi svým počátečním a koncovým bodem.

3.1.3 Editace mapy

Tato hra samozřejmě taktéž obsahuje editor mapy, při kterém lze do terénu přidávat libovolný počet stromů, zlatých dolů a kamenů. Hráč si může zvolit buď tvorbu své vlastní mapy, nebo zvolí již vytvořenou mapu.

3.1.4 Porovnání

Princip této hry se nijak zvlášť neliší od ostatních strategických her. Jako v každé hře je hlavním cílem zničit hlavní budovu, v našem případě Castle, abychom se stali vítězem. Lehkou inspirací mi byla hra **Age of Mythology**, která je taktéž zařazena do historického světa. Obsahuje větší počet jednotek, kde každá jednotka je něčím specifická. Ve hře jsou obsaženy i prvky magie. Samozřejmě je hra ve 3D a byla vydána v roce 2002 a v roce 2003 se stala RTS hrou roku.

4 Programátorská dokumentace

4.1 Rozbor serverové části a komunikace

Nyní přecházíme do programátorské části, kde si popíšeme nejdůležitější prvky implementace. V této části si taktéž popíšeme princip komunikace mezi klientem a serverem.

4.1.1 SpiderMonkey API

Při vytváření serveru a komunikace mezi klientem jsem využil SpiderMonkey API [6], které bylo vytvořeno pro jMonkeyEngine a slouží ke zjednodušení implementace serveru. Princip spočívá v tom, že se několik klientů připojí na jeden server.

- **Centrální server** - je to základní jednoduchá aplikace (headless SimpleApplication), která koordinuje hru na pozadí bez grafického vzhledu.
- **Herní klient** - každý hráč má svého herního klienta (standard SimpleApplication) a připojí se k centrálnímu serveru.

Celý princip tedy spočívá v tom, že klient udržuje spojení se serverem a informuje ho o svých pohybech a akcích, které následně server rozesílá do všech připojených klientů. Díky tomuto můžou všichni herní klienti sdílet stejný herní svět. Hra se poté zobrazuje klientovi z jeho herního pohledu.

SpiderMonkey API je sada rozhraní a pomocných tříd, které se nachází v *"com.jme3.network"* balíčku. Pro většinu uživatelů je tento balíček vše, co potřebují. Máme dva typy balíčků :

- **com.jme3.network.Client** - tento balíček slouží k implementaci na straně klienta.
- **com.jme3.network.HostedConnection** - balíček, který se používá na straně serveru.

Můžeme si zaregistrovat několik typů listenerů, které nás budou informovat o změnách. Jedná se o tyto typy :

- **MessageListeners** - tento typ listeneru se nachází na obou stranách. Upozorňuje server i klienta na to, jak přijde nová zpráva. Pomocí tohoto listeneru můžeme být informování pouze o určitém typu zpráv.
- **ClientStateListeners** - slouží k informování klienta o změnách jeho stavu připojení. Příkladem může být například kopnutí ze serveru a další.
- **ConnectionListeners** - informuje server o tom, jakmile se připojí nebo odpojí klient.
- **ErrorListeners** - informuje klienta o problémech v síti. Například dojde-li k chybě připojení k serveru, klient vyhodí výjimku, která může být různě ošetřena.

4.2 Implementace serveru

Jak jsem již výše zmínil, server je typu **headless**. Znamená to, že se server spustí normálně, ale neotevře se žádné okno a aplikace nepřijímá žádný uživatelský vstup. Toto je typické chování pro serverovou aplikaci.

```
public static void main(String[] args) {
        Main app = new Main();
        app.start(JmeContext.Type.Headless); // headless type for servers!
    }
```

Výpis 2: Spuštění serveru

4.2.1 Vytváření a přijímání zpráv

Každá zpráva reprezentuje data, která chceme posílat mezi klientem a serverem. Pro každý typ zprávy je potřeba vytvořit třídu, která rozšiřuje třídu
com.jme3.network.AbstractMessage. Dále je potřeba vytvořit anotaci **@Serializable** ze třídy *com.jme3.network.serializing.Serializable* a poté vytvořit prázdný konstruktor. Dále si vytvoříme vlastní konstruktory, metody a obsahy zpráv.

```
@Serializable
public class HelloMessage extends AbstractMessage {
  private String hello;          // vlastni libovolna data
  public HelloMessage() {}       // prazdny konstruktor
  public HelloMessage(String s) { hello = s; } // vlastni konstruktor
}
```

Výpis 3: Vytvoření zprávy

Samozřejmě je potřeba, aby se tato třída nacházela na obou stranách tzn. jak na serveru, tak u klienta, a taky je potřeba, aby každá zpráva byla zaregistrována do
com.jme3.network.serializing.Serializer taktéž na obou stranách. Více informací v [6].

```
Serializer.registerClass(HelloMessage.class);
```

Výpis 4: Registrace zprávy

Ve svém serveru mám použito těchto tříd více. Jak jsem výše uvedl, pro jiný typ dat je potřeba vždy vytvořit novou třídu. Na serverové části se nachází tyto typy zpráv :

- **EditMessage.java** - tato třída slouží k posílání zpráv pro editaci mapy. V této třídě se nám posílá 5 proměnných (co, pozici x, pozice z, šířka a výška). Pro dané parametry jsou zde funkce **GET** a **SET**. Dále se zde nachází list zpráv. Do tohoto listu ukládám všechny zprávy a následně je najednou pošlu.
- **NameMapMessage.java** - tato třída slouží pouze k posílání názvu map ze serveru na klienta, aby klient věděl, kterou mapu si zvolit.
- **HelloMessage.java** - třída, která se stará o jednotky, jejich pohyb, aktualizace života a další.

- **PlayerInfoMessage.java** - jednoduchá třída, která se stará pouze o posílání stavu zlata z klienta na server a naopak.
- **ShootMessage.java** - stará se o posílání střely jednotek na klienta, aby klient věděl kdy střelu vykreslit a kde.
- **SoldierBuildingMessage.java** - tato třída se stará o posílaní zpráv, které se týkají budov. Například klient chce vytvořit budovu, tak se pošle zpráva s pozicí, jaká budova se má vytvořit, životy a další. Taktéž se jedná o posílání zpráv ze serveru na klienta, aby věděl, zdali není budova zničená a kolik má životů.
- **SpellMessage.java** - má na starost odeslání vytvoření kouzla na server, aby věděl kde má způsobit zranění a které jednotky ovlivnit.
- **UpdateMessage.java** - jednoduchá třída, která se stará pouze o odeslání informace na server, která jednotka má být vylepšená.
- **StatusMessage.java** - zpráva, která posílá připravenost hráče na server.

Přijde mi zbytečné každou třídu popisovat zvlášť, protože mezi nimi je minimální rozdíl. Vzor těchto tříd je vždy stejný, avšak obsahují pouze jiná data k odeslání.

Takto vytvořené zprávy poté odesíláme buď klientovi nebo na server. Abychom tyto zprávy mohli přijímat, je třeba nejdříve vytvořit listener, který bude na tyto zprávy reagovat.

```
try {
        myServer = Network.createServer(6143);
        Serializer .registerClass(NameMapMessage.class);
        Serializer .registerClass(StatusMessage.class);
        Serializer .registerClass(EditMessage.class);
    ......
    ......
        myServer.addMessageListener(control, NameMapMessage.class);
        myServer.addMessageListener(this, EditMessage.class);
        myServer.addMessageListener(this, StatusMessage.class);
    ......
    ......
        myServer.addConnectionListener(this);
    ......
    }    ......
```

Výpis 5: Registrace zpráv a listenerů

Na Výpisu 5 lze vidět kus kódu, který ukazuje, jak registrujeme jednotlivé třídy a listenery. Toto musíme provést pro všechny třídy, které chceme posílat a přijímat. Tato registrace musí být provedena jak u klienta tak i na serveru ve **stejném** pořadí. Pokud toto bude provedeno rozdílně, dojde k chybnému překládání, protože každá registrovaná třída bude mít jiné identifikační číslo (ID). V případě, že budeme mít na serveru jiné ID a pošleme zprávu na klienta, kde bude mít stejná třída jiné ID, dojde k přijetí zprávy, ale její deserializace bude neúspěšná.

Abychom mohli tyto zprávy úspěšné serializovat a zpracovávat, má každá ze zpráv vytvořený svůj kontrolér. V tomto kontroléru se nachází funkce, které jsou potřebné k tomu, abychom mohli zprávu přečíst a následně ji bez problému vykonat. Dále také obsahuje funkce pro odeslání zpráv daného typu.

Popíšeme si například třídu **BuildingMessageController.java**, která obsahuje tyto funkce :

- **checkSoldierBuildingMessage** - tato funkce se stará o zpracování příchozích zpráv od klienta. Nejprve zprávu přetypujeme z Message na SoldierBuildingMessage, abychom mohli následně získat všechny data. Jelikož se jedná o vytváření budovy tak si vytvoříme novou budovu a do konstruktoru dáme data, která získáme ze zprávy od klienta. Následně si z parametru typu zjistíme o jakou budovu se jedná a podle toho budeme pokračovat a vytvoříme příslušnou budovu daného typu. Dále budovu vložíme do listu daného hráče, který si jí vytvořil. Budova se taktéž musí přidat do mřížky, aby se jí jednotky mohli později vyhýbat.

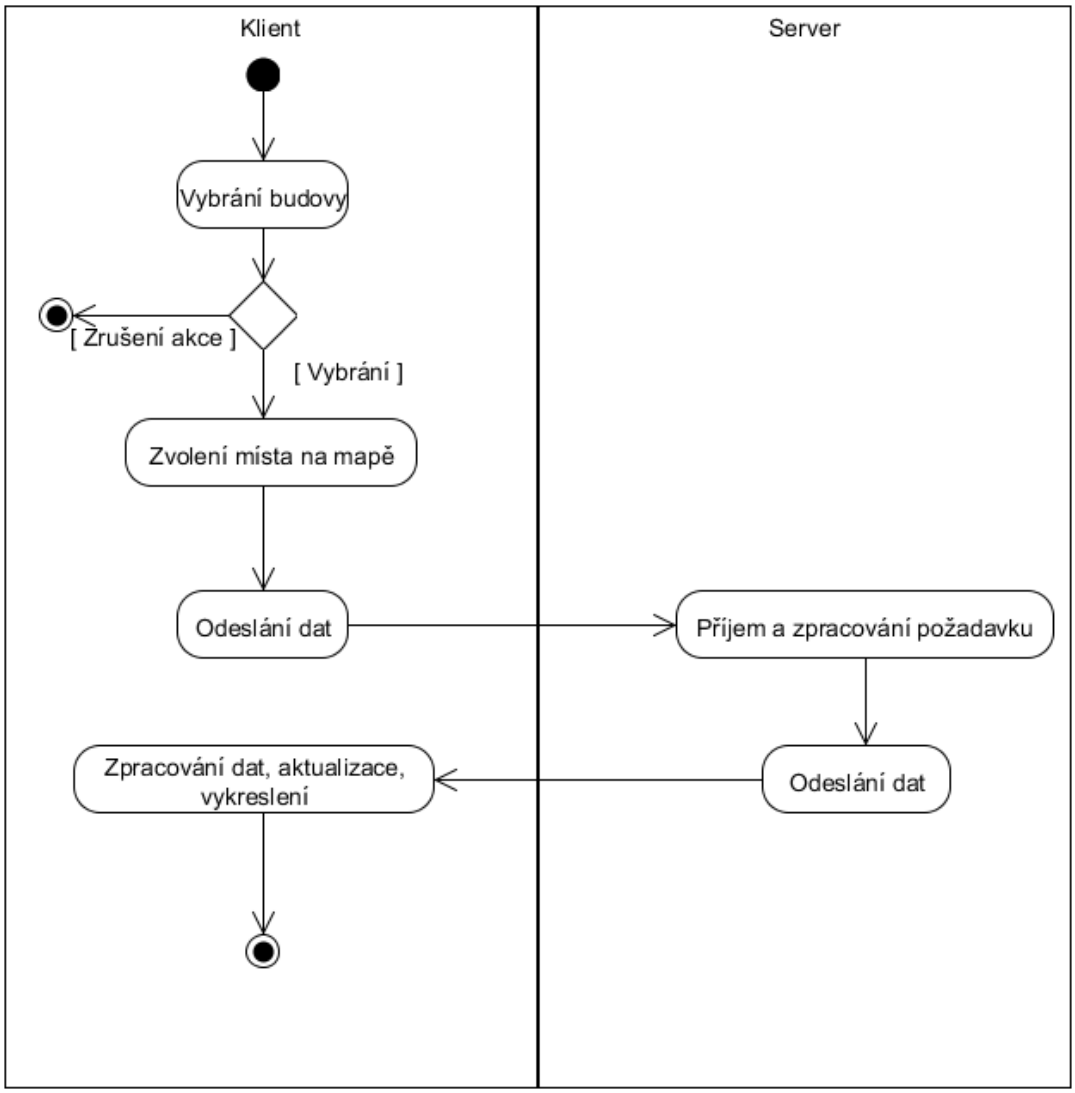
- **sendBuilding** - cílem této funkce je odeslat vytvořenou budovu i protihráči, aby si ji mohl vložit do mapy, vykreslit a útočit na ní.

- **updateBuilding** - zde je cílem informovat oba hráče o stavu budovy. V případě, že budově budou odebrány životy je třeba aby se o této změně dozvěděli oba hráči. Ti si následně aktualizují data.

Všechny takto vytvořené třídy, které se starají o zpracování zpráv, pracují. Vždy je zde funkce, která se zavolá v případě přijetí zprávy od klienta. Zjistí se typ zprávy a následně provede příslušná část kódu. Jedná-li se o stavbu nebo vytvoření jednotky je vždy potřeba, aby o změně byl informován druhý hráč a samozřejmě co se týče aktualizací, tak o té musí být informování oba hráči.

Na Obrázku 4 lze vidět znázornění komunikace mezi serverem a klientem. Pro příklad jsem dal stavbu budovy klientem. Jako první akce je potřeba, aby si klient vybral budovu, kterou chce postavit a následně zvolil místo na mapě nebo akci zrušil. V případě zvolení a potvrzení stavby dojde k odeslání dat na server. Na serveru se zpráva zpracuje a následně odešle aktualizace. Klient danou zprávu zpracuje a na základě ní upraví vykreslení, aktualizuje či vykoná jinou akci.

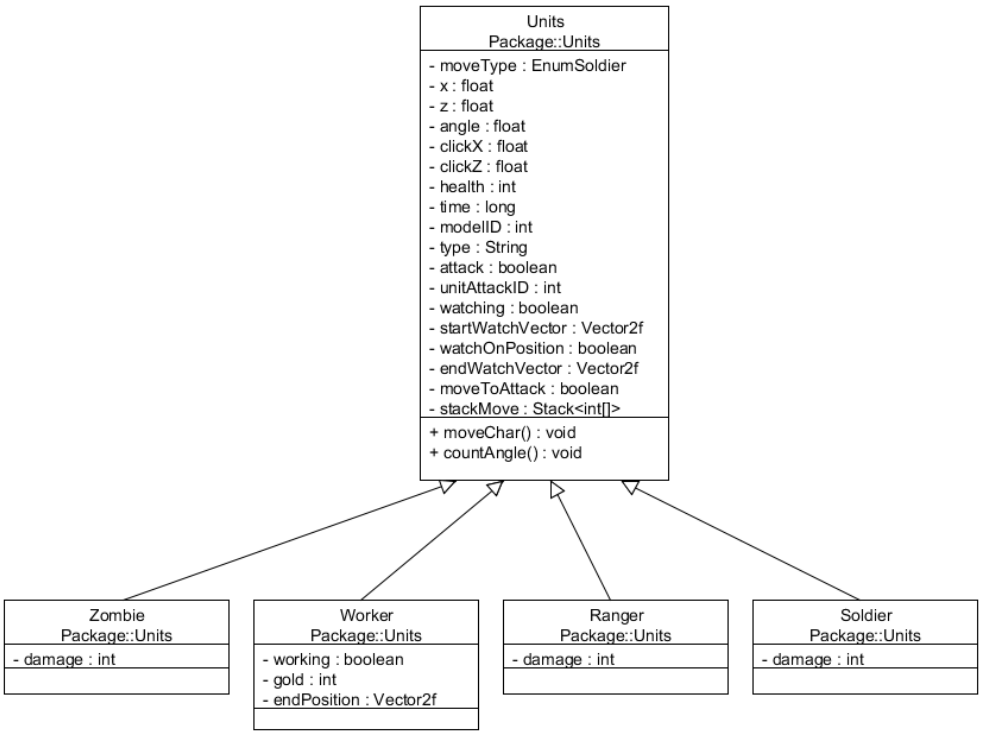
Takovýto princip je stále stejný, kdy klient vybere akci a odešle data na server. Server je následně zpracuje a na základě toho odešle zpět nové informace, které jsou pro klienta důležité a o které zažádal.



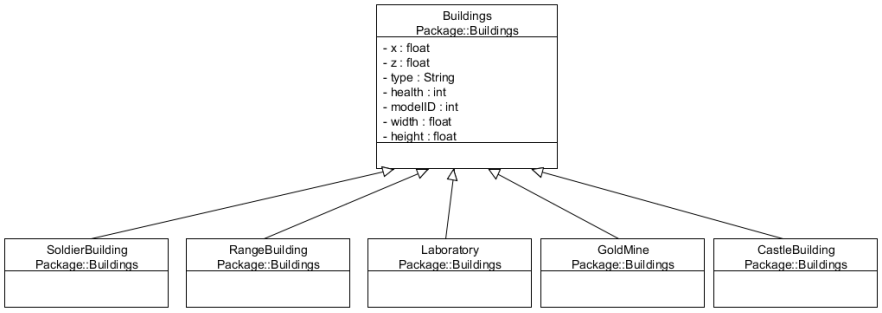
Obrázek 4: Znázornění komunikace server - klient

4.2.2 Jednotky a budovy

Na serveru se také nachází jednotlivé třídy pro budovy i jednotky. Vždy je vytvořen jen rodič a následně ostatní jsou potomci. Tento jev nazýváme **dědičnost**. Tuto hierarchii si názorně zobrazíme na Obrázku 5 a na Obrázku 6.



Obrázek 5: Dědičnost jednotek



Obrázek 6: Dědičnost budov

4.2.3 Mapa

Na serveru se také nachází implementace pro načítání a tvorbu mapy. Nachází se zde několik funkcí, které slouží k ukládání, vkládání objektů do mřížky (grid) a další důležité funkce. V balíčku **Map** se nachází třída **FunctionMap**. Tato třída obsahuje funkci **loadAndSendNameMap**, která se nám postará o to, aby hráč, který se připojí jako první, obdržel názvy map, které jsou dostupné a ze kterých si může vybrat. Dále ve třídě **Con-**

troller.java se nachází tyto funkce :

- **editMapToGrid** - tato funkce se stará o vkládání věcí do listu a následného vložení do mřížky. Do mřížky je vkládáme proto, abychom se mohli následně pomocí algoritmu při pohybu těmto věcem vyhýbat.
- **saveMapToFile** - zde ukládáme jednotlivé věci do souboru, abychom je mohli načíst najednou jako celou mapu.
- **loadMap** - tato funkce slouží k načtení mapy ze souboru. Pokud si klient nezvolí mapu, načte se základní mapa s názvem **map1**.
- **sendEditedMap** - v neposlední řadě je potřeba, abychom mapu odeslali oběma hráčům. Tato funkce se postará o načtení listu a oběma hráčům pošle všechny objekty, které si má vložit do mapy a následně vykreslit.

4.2.4 Realizace útoku na jednotky a budovy

V této kapitole si projedeme balíček **Attack**, ve kterém se nachází realizace útoku na jednotky, budovy a také realizace útoku na dálku. Do tohoto balíčku patří tyto třídy :

- **AnimationOfRange.java** - tato třída obsahuje pouze proměnné, funkci k pohybu (moveShoot()) a funkci k výpočtu směru pro střelu (countAngle()).
- **BasicAttack.java** - třída obstarává úspěšný útok na jednotku pomocí kliknutí. Nejedná se tedy o automatický útok. Nachází se zde dvě funkce a to :
  - **checkDistance(Units soldier, Units soldier2, Grid grid)** - tato funkce se volá v hlavním vláknu, které si popíšeme až nakonec. Volá se pouze tehdy, pokud má jednotka, která se kontroluje, ID od cizí jednotky na kterou má zaútočit. Na začátku této funkce se vypočte vzdálenost mezi dvěma objekty a to proto, abychom věděli, jestli může jednotka už útočit nebo se musí nejprve přiblížit. Pokud je vzdálenost větší než 15, uloží se nám do jednotky, se kterou chceme pohnout, pozice jednotky na kterou chceme zaútočit a následně se spustí algoritmus pro vyhledání cesty. Vyhledá se cesta a jednotka se přiblíží k nepřátelské jednotce. Jakmile je vzdálenost menší nebo rovná 15, jednotka dostane příkaz k zastavení a zavolá se nyní druhá funkce s názvem **Attack(Units soldier, Units soldier2)**.
  - **Attack(Units soldier, Units soldier2)** - tato funkce nám obstarává samotný útok. Nejprve se zkontroluje, zdali není jednotka typu Ranger. Pokud ano, ještě před samotným ubráním životů je potřeba vytvořit střelu, která nám bude naznačovat útok na dálku. Následně se vezmou dvě jednotky, kterých se tato akce týká. Jedná se o naší a nepřátelskou a to z toho důvodu, abychom věděli, jaké jednotce máme ubrat životy a kolik. Jakmile bude čas od útoku více než 3s, čas se vynuluje a jednotka může zaútočit znovu.

- **BasicAttackBuilding.java** - tato třída je téměř stejná jako třída **BasicAttack.java**. Zde pouze pracujeme s naší jednotkou a s budovou nepřítele. Funkce mají tedy místo druhého parametru *Units soldier2* tento **Buildings building**.
- **AutoBasicAttack.java** - třída, která se stará o automatický útok. Její funkce se volají ve smyčce v hlavním vlákně. Zde se tedy stále kontroluje, jestli není cizí jednotka poblíž na kterou bychom zaútočili automaticky. Princip v této třídě je téměř stejný jako u předchozích. Kontroluje se tedy vzdálenost, která musí být pro automatický útok nižší než 20. V tomto případě se jednotka pomocí algoritmu přesune automaticky blíž na pozici, na které se nachází jednotka a zaútočí.
- **AutoBasicAttackBuilding.java** - třída, která se liší od automatického útoku na jednotky pouze ve vzdálenosti pro přiblížení a typem druhého parametru. Opět zde kontrolujeme jako druhý parametr budovu a s ní taktéž pracujeme.

4.2.5 Vyhledání cesty a tvorba formace

Tato část se bude zabývat vyhledáním cesty pro jednotky mezi překážkami. Jedná se o algoritmus, který se stará o to, aby jednotka měla přesnou a zároveň co nejkratší cestu na místo.

4.2.5.1 A* Algoritmus

A* (A star) [8] je počítačový algoritmus, který se používá pro vyhledávání optimálních cest. Byl vytvořen v roce 1968 Peterem Hartem, Nilsem Nilssonem a Bertramem Raphaelem.

A* používá tzv. hladový princip pro nalezení optimální cesty z počátečního do koncového bodu. Touto cestou se rozumí ta, která je nejkratší, nejrychlejší a další. K vytvoření algoritmu jsem použil pseudokód na Obrázku 7.

Nejprve si tedy vytvoříme listy pro otevřené uzly a uzavřené uzly. Do otevřeného listu vložíme jako první počáteční uzel. Nyní následuje smyčka ve které máme na začátku vložit otevřený uzel s nejmenší cenou do současného uzlu, se kterým budeme pracovat. Abychom měli jako první uzel s nejmenší cenou, použil jsem funkci **Collections.sort(openList, new CustomComparator())**, která se postará o to, aby vždy na první pozici byl uzel, jehož cena je nejmenší. Detailní popis algoritmu včetně praktické ukázky naleznete v [7]. Realizace komparátoru pak vypadá jako na Výpisu 6.

```
public static class CustomComparator implements Comparator<PathNode>{
        public int compare(PathNode o1, PathNode o2) {
            return o1.getCost() − o2.getCost();
        }
    }
```

Výpis 6: Vlastní komparátor pro seřazení uzlů

A* PATHFINDING TUTORIAL

```
OPEN //the set of nodes to be evaluated
CLOSED //the set of nodes already evaluated
add the start node to OPEN

loop
 current = node in OPEN with the lowest f_cost
 remove current from OPEN
 add current to CLOSED

 if current is the target node //path has been found
  return

 foreach neighbour of the current node
  if neighbour is not traversable or neighbour is in CLOSED
   skip to the next neighbour

  if new path to neighbour is shorter OR neighbour is not in OPEN
   set f_cost of neighbour
   set parent of neighbour to current
   if neighbour is not in OPEN
    add neighbour to OPEN
```

zdroj : https://www.youtube.com/user/Cercopithecan/

Obrázek 7: A* pseudokód

Nyní z otevřeného listu odstraníme uzel se kterým nyní pracujeme tzn. současný a následně vložíme do uzavřeného listu. Uzavřený list značí již navštívené uzly. Následuje podmínka, která kontroluje, jestli se nacházíme na konečné pozici. Pokud ano, smyčka se ukončí. Nyní musíme projít každého souseda pro současný uzel. Zde se nachází dvě podmínky. První z nich je, že pokud je soused již navštívený nebo na něj nemůžeme jít, přeskočíme jej a půjdeme na dalšího. V případě, že nová cesta k sousednímu uzlu je kratší uložíme si tuto novou cenu jako cenu souseda a nastavíme mu souseda současný uzel. V případě, že se soused nenachází v listu otevřených uzlů, vytvoříme jej a vložíme do listu.

Po ukončení vyhledání cesty procházíme od počátečního uzlu pomocí jeho sousedů až do konečného uzlu, který nemá žádného souseda. Pozice všech uzlů ukládám do zásobníku, který má jednotka se kterou budeme pohybovat. Tato jednotka si poté ze zásobníku vytahuje pozice na které má jít až dojde na konečné místo.

4.2.5.2 Formace

Princip formace není nijak zvlášť složitá věc. Jakmile přijde zpráva pro aktualizaci jednotek tak se pro každou jednotku volá metoda pro zajištění formace. Probíhá to tak, že ve třídě **Formation.java** se nachází dvě proměnné **i** a **j**, které se starají o posun jednotek.

Vše se vyhledává ve mřížce, ve které se nachází všechny objekty. Formace spočívá v tom, že se první jednotka uloží na místo a další se postaví vedle ní. V jedné řadě může stát maximálně 5 jednotek. V dalším případě se musí vytvořit nová řada. V případě, že se jednotka nemůže postavit hned vedle, posune se o další pozici, dokud se nenajde ta správná. Po nastavení formace pro všechny jednotky, kterých se aktualizace týkala, nastavíme proměnné **i** a **j** zpět na 0, aby formace byla použitelná pro další aktualizaci umístění jednotek. Příklad formace lze vidět na Obrázku 15.

4.2.6 Mřížka (Grid)

Mřížka nám slouží k tomu, abychom věděli kudy se pohybovat. Mřížku využívá A* algoritmus, který mezi objekty v mapě vyhledává cestu, aby se tyto objekty mohly vyhýbat překážkám. V mřížce se nachází budovy, stromy, kameny i zlaté bedny. Celá mřížka je tedy dvourozměrné pole typu String o šířce a výšce 256. Celá mřížka je zaplněna tečkami ("."). Je-li v mřížce tečka, znamená to, že je pozice volná a jednotka může projít. V případě, že se zde něco nachází tak mřížka obsahuje křížek ("x"). Celou funkci nám obstarávají tyto dvě funkce :

- **addToGrid** - tato funkce se stará o to, aby nám vložila do mřížky křížky. Tato funkce se volá například když stavíme budovu. Klient odešle informace o postavení budovy na server a server zprávu zpracuje, získá si pozici budovy, její šířku, výšku a na základě toho obsadí pozice v mřížce. Díky tomu se budou jednotky této budově vyhýbat a nebudou procházet přes ní.
- **deleteFromGrid** - téměř totožná funkce, akorát její funkce spočívá v odstranění objektu z mřížky. Tato funkce se volá v případě, že je budova zničená a odstraněná z mapy.

4.2.7 Hlavní vlákno (Controller)

Hlavní část celého programu se nachází v balíčku **Controller**. V tomto balíčku se nachází stejnojmenná třída, která se stará o chod celého serveru.

Při spuštění vlákna se nejprve spustí funkce **sendStartBroadCast()**. Ta se stará o odeslání veškerých potřebných informací pro hráče na začátku hry. Dále se odešle pro oba hráče mapa, která se skládá z různých objektů, které se nachází na mapě. Dále následuje smyčka **while**. V této smyčce se provádějí všechny potřebné funkce, aby byl zajištěn správný chod serveru. Na začátku je kontrolován pohyb jednotek a kontrola, zdali některá z jednotek nemá život menší nebo roven nule. Pokud ano, vloží se do listu jednotka, která má být smazána. Jako další je třeba taktéž kontrolovat životy budov, zdali nemají být odstraněny.

Dále následuje kontrola automatického útoku. První se spustí funkce s názvem **autoAndAttackunit()**, která se nachází ve třídě functions. V této funkci se nám kontrolují všechny jednotky obou hráčů. V případě, že jednotka má ID nepřátelské jednotky, spustí se funkce pro útok na základě kliknutí uživatele. V opačném případě následuje kontrola vzdálenosti mezi jednotkami. Jedná se o funkci **checkDistance()**, kterou jsme si popsali

výše. Abychom věděli, že nepřátelská jednotka je již mrtvá, následuje funkce **checkIfIdExists()**. Pokud jednotka ještě existuje, útok na ní stále pokračuje. V případě, že ID nebylo nalezeno se nastaví parametr u jednotky na 0, aby nedošlo k chybě a neútočila na jednotku, která je již mrtvá a neexistuje. Tento stejný postup použijeme i na kontrolu mezi budovami. Jedná se o funkci **autoAndAttackBuilding()**.

Jako další dochází ke kontrole, zdali je jednotka v módu hlídkování. Jedná se o funkce **setWatching(unit)** a **onWatchPosition(unit)**. Pokud má jednotka nastaven parametr na hlídkování spustí se tyto funkce. Následuje vypočtení cesty pomocí algoritmu na místo a následně se kontroluje, jestli je jednotka na místě a pokud ano, tak se spustí algoritmus na vyhledání cesty zpět. Toto se děje stále dokola, dokud nebude mít jednotka zrušen parametr hlídkování.

Nyní přicházíme ke kontrole vytváření střel a smazání jednotek a budov. Nejprve zkontrolujeme, jestli je potřeba odeslat vytvořenou střelu oběma klientům, aby si ji mohli vykreslit. Dále následuje kontrola listu pro smazání jednotek a budov. V případě, že se v listu nachází nějaká jednotka, odešle se zpráva klientům o jejich smazání. Následuje odeslání stavu jednotek a budov klientům a také kontrola zdali je potřeba provést vylepšení jednotek. Jako poslední se kontroluje práce pracovníků. Pracovníci jsou automaticky poslání na pozici, kde se nachází zlato a následně jsou poslání zpět ke hradu, kde dochází ke přičtení natěženého zlata.

V této třídě se také nacházejí listenery pro příjem zpráv, aby mohli být následně zpracovány viz. Výpis 7.

```
public void messageReceived(HostedConnection source, Message m) {
        if (m instanceof HelloMessage) {
            helloMessageController.checkHelloMessage(source, m);
        } else if (m instanceof SoldierBuildingMessage) {
            buildingMessageController.checkSoldierBuildingMessage(source, m);
        } else if (m instanceof PlayerInfoMessage) {
            playerInfoController .checkPlayerInfoMessage(source, m);
        } else if (m instanceof UpdateMessage) {
            updateController.checkUpdate(source, m);
        } else if (m instanceof SpellMessage) {
            spellController .checkSpell(source, m);
        } else if (m instanceof NameMapMessage) {
            NameMapMessage mess = (NameMapMessage) m;
            String [] name;
            name = mess.getName();
            try {
                loadMap(name[0]);
            } catch (FileNotFoundException ex) {
                Logger.getLogger(Controller.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Výpis 7: Listenery pro zprávy ve třídě controller

4.2.8 Umělá inteligence

V této hře je taktéž naimplementována umělá inteligence. Je to samostatné vlákno, které se spustí v případě, že se jeden z hráčů odpojí.

Na začátku se jako první zkontroluje, jestli hráč již postavil hlavní budovu a laboratoř, aby nedošlo k jejímu opětovnému postavení. Dále následuje opět smyčka **while**, která se stará o chod umělé inteligence. Celá smyčka se opakuje po 1,5 sekundě. Nejprve se zkontroluje, zdali hrad existuje. Pokud ne, je potřeba ho nejprve postavit.

Aby se počítačový hráč mohl bránit jsou zde funkce (**checkNearEnemyAtBuilding(), checkNearEnemyAtUnit()**) , které slouží ke kontrole jestli se nenachází nepřátelský hráč poblíž naší budovy nebo jednotky. V případě, že se jednotky nachází poblíž, jsou na ně odeslány automaticky jednotky počítače. V případě, že jednotky hráče utíkají, je zde také kontrola, aby počítačové jednotky neutekly až k základně hráče (**checkIfAway()**). Pokud se nachází daleko od základny, zastaví se. Dále je zde funkce, která se stará o zničení protivníka (**destroyEnemy()**). Pokud máme více než 15 jednotek a hráč má postavenou základnu jsou počítačové jednotky automaticky poslány na zničení základny.

Dále následuje **switch**, který na základě vygenerovaného čísla od 1 do 10 vybere akci počítače. Jedná se o tyto možnosti :

1. **createWork()** - funkce, která se stará o vytvoření potřebného počtu pracovníků.
2. **buildLaboratory()** - jestliže ještě není postavena laboratoř a počítač má na ní dostatek peněz, postaví se.
3. **createRanger()** - slouží k vytvoření jednotky typu Ranger.
4. **buildAltar()** - postavení budovy, která se stará o vytváření základních bojových jednotek.
5. **createSoldier()** - funkce, která vytvoří jednotku typu Soldier.
6. **buildRange()** - postaví budovu, která slouží pro vytváření jednotek typu Ranger.
7. **buildGoldmine()** - tato funkce se stará o postavení budovy pro těžbu zlata. Nejprve se zjistí pozice hlavní budovy a následně se vyhledává truhla zlata, která je nejblíže k základně. Jakmile se tato truhla vyhledá, postaví se v její blízkosti budova.
8. **createZombie()** - vytvoří jednotku typu Zombie.
9. **createUpdate()** - funkce se postará o vylepšení některé z jednotek.
10. **makeFormation()** - tato volba se postará o seskupení jednotek na určené pozici, aby byly pohromadě.

Maximální počet bojových jednotek pro počítač je **30**.

4.3 Implementace Klienta

Nyní následuje druhá část vytvořené hry a to klient. Aby mohl hráč hrát tuto hru, musí si nejdřív spustit klienta. Ten se stará o vykreslení jednotek, budov, objektů a dále o ovládání jednotlivých jednotek a další. Nyní si popíšeme implementaci klienta po částech.

4.3.1 Spuštění klienta (Třída Main.java)

Při počátečním spuštění klienta dojde nejprve k načtení scény pro hru a dále načtení všech modelů ve hře, aby nedocházelo k pozdějšímu sekání hry. Tuto činnost má na starost funkce **loadModels**, která se nachází ve třídě **models**. V této funkci se nám načtou modely do proměnných typu **Node**. Modely se načítají pomocí **assetManager** viz. Výpis 8. AssetManager slouží k načítání 3D objektů, optimalizaci a k připravení pro použití do hry. Na výpisu jde vidět, že lze nastavit i velikost a název modelu pro pozdější rozlišení mezi ostatními objekty.

```
public void loadModels() {
        skelet = (Node) assetManager.loadModel("Models/skelet/skelet.j3o");
        skelet.setLocalScale(0.08f);
        zombie = (Node) assetManager.loadModel("Models/soldier/soldier.j3o");
        zombie.setLocalScale(2f);
    ...
    ...
    ...
        stone = (Node) assetManager.loadModel("Models/Rock1/Rock1.j3o");
        stone.setName("Stone");
    }
```

Výpis 8: Načtení modelů

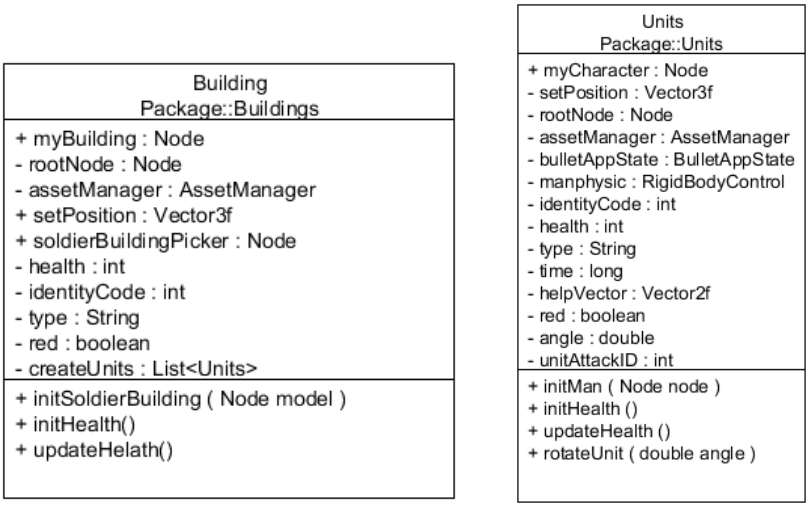
Následuje podobná registrace zpráv a listenerů pro tyto zprávy, jak tomu bylo u serveru. Dále je ještě potřeba vytvořit uživatelské menu, abychom mohli zobrazit úvodní stránku. Toto provádíme pomocí **hud**, který si popíšeme později.

4.3.2 Jednotky a budovy

Pro jednotky a budovy jsou vytvořené dva balíčky s názvy **Buildings** (budovy) a **Units** (jednotky) .

- **Buildings** - v tomto balíčku se nachází třída Building. Tato třída slouží k vytváření objektů budov, její inicializaci, inicializaci životů a obsahuje také funkci pro aktualizaci životů pro vykreslení.
- **Units** - v tomto balíčku se nachází více tříd. Je zde třída **GameData.java**, ve které se nachází důležité data ze hry. Dále je zde třída **RangeAttack.java**, která slouží k vytvoření objektu střely pro jednotky Ranger. V případě, že jednotka Ranger útočí a přijde ze serveru zpráva pro vytvoření střely, vytvoří se objekt této třídy, kde se nám načte kulička, která se následně pohybuje z vytvořené pozice na pozici cíle pomocí funkce **moveShoot()**.

Dále se v tomto balíčku nachází také třída **Thing.java**, která slouží k vytváření objektu pro objekty na mapě, tímto mám na mysli stromy, kameny a truhly na zlato. A v poslední řadě je zde třída **Units**. Má téměř stejnou strukturu jako třída pro budovy. Názorně jsou tyto dvě třídy níže zobrazeny pomocí diagramu (Obrázek 8).



Obrázek 8: Třída pro budovy a jednotky

4.3.3 Zprávy pro komunikaci

Jedná se o balíček **Messages**. Tuto část budu popisovat velmi zlehka, protože je totožná jako část na serveru. Jak jsem již zmínil, na obou stranách musí být tyto třídy totožné, aby komunikace probíhala bez chyb. Jedná se tedy o ty samé třídy, které jsou popsány v **Implementaci serveru** v kapitole **4.2.1**.

4.3.4 Hlavní část - kontrolér

Celá hra probíhá pomocí smyčky **simpleRender**, která se nachází ve třídě **Main.java**. Tato smyčka je volána stále dokola. Nachází se zde funkce ze třídy **Control.java**, které slouží k hlavnímu chodu celé hry. Popíšeme si funkce, které se zde nachází :

- **MoveCamera()** - tato funkce slouží k pohybu kamery po mapě. Na základě listenerů **OnAnalog()**,který se nachází v kontroléru, snímá každý pohyb pomocí myši a na základě pozice myši se rozhodne, jestli se bude kamera pohybovat a jakým směrem. Abychom mohli pohyb myši dokonale sledoval, je potřeba nejdříve registrovat listener pro jednotlivé akce myši.

- **CheckMessage()** - tato funkce spustí funkci **MessageCheck()**, která se nachází ve třídě messageControl. Tato funkce se stará o zpracování zpráv, které přicházejí ze serveru. Zprávy, které přijdou ze serveru se pomocí listenerů, který na základě příchozí zprávy zjistí, jakého typu zpráva je a vloží jí do příslušného listu zpráv. Pokud list není prázdný tak se provede funkce **MessageCheck()**. Ta v sobě obsahuje funkce, které slouží jak k vytvoření jednotek a budov tak i k její aktualizací. Dále je zde zpracování zprávy pro střelu, kdy dojde k vytvoření střely a následně ke vložení do scény a k jejímu zobrazení a pohybu. Tato funkce slouží taktéž ke zpracování zprávy pro odstranění jednotek a budov. Je to tedy kontrolér, který se stará o zpracování všech zpráv a o realizaci.

- **checkCreating()** - funkce, která obstarává kontrolu pro tvorbu jednotek z budov a také pro vytváření útoku pro jednotku Ranger. Tato funkce volá funkci **checkCreate** pro každou budovu hráče. V této funkci se nám nastaví příslušné menu pro budovu a kontroluje se zde čas pro vytváření jednotek. Jakmile dojde k uplynutí času, vytvoří se objekt daného typu, inicializuje se, vloží do scény a následně se přidá do listu hráče. Po vytvoření jednotky dojde k odeslání zprávy o vytvoření na server, aby server mohl zpracovat vytvoření jednotky a následně odeslat druhému hráči, aby si ji mohl vykreslit do scény.

4.3.5 Stavění budov, pohyb jednotek

V této kapitole si popíšeme průběh stavění budov a pohybu jednotek. Popíšeme zde, jak probíhá odchytávání pozice myši a dalších důležitých částí pro stavění a pohyb na mapě. Pro zjištění pozice myši jsem vytvořil funkci **getClickedPositionOnMap()** viz. Výpis 9. Tato funkce obstará přesnou pozici kliknutí na mapě, tedy ne na obrazovce.

```
public Vector3f getClickedPositionOnMap() {
        CollisionResults results = new CollisionResults();
        Vector2f click2d = inputManager.getCursorPosition();
        Vector3f click3d = cam.getWorldCoordinates(new Vector2f(click2d.x, click2d.y), 0f).clone()
            ;
        Vector3f dir = cam.getWorldCoordinates(new Vector2f(click2d.x, click2d.y), 1f).
            subtractLocal(click3d).normalizeLocal();
        Ray ray = new Ray(click3d, dir);
        rootNode.collideWith(ray, results);
        Vector3f pt = null;
        for (int i = 0; i < results.size(); i++) {
            pt = results.getCollision(i).getContactPoint();
        }
        return pt;
    }
```

Výpis 9: Získání pozice kliknutí

- **Stavění budov** - abychom mohli postavit budovu, je potřeba nejprve kliknout na tlačítko příslušné budovy v dolním menu. Jakmile klikneme na budovu, zobrazí se nám náhled budovy na mapě a pohybuje se nám na základě pohybu myši. Během

pohybu budovy po mapě se nám volá funkce **checkCollision**. Tato funkce se stará o zjištění kolize mezi naší budovou a jakýmkoliv jiným objektem na mapě. Pokud je počet kolizí větší než 0, funkce nám vrací **true**, což znamená, že budova je v kolizi s jiným objektem. V tomto případě budova zčervená a znemožní se její postavení. V opačném případě pokud můžeme budovu postavit, klikneme levým tlačítkem a tímto se budova postaví na pozici myši. Dále se odešle na server zpráva o postavení budovy, server tuto zprávu zpracuje a odešle druhému hráči.

```
public boolean checkCollision(Node a)
    {
        CollisionResults results = new CollisionResults();
        List<Spatial> spatial = rootNode.getChildren();
        for(Spatial spatial2 : spatial)
        {
            if(spatial2.getName().equals("Tree") || spatial2.getName().equals("Gold") ||
                spatial2.getName().equals("Stone") ||
                    spatial2.getName().equals("Units") || (spatial2.getName().equals("
                        Buildings"))
                    && (spatial2!=a)){
            boundingBox = (BoundingBox) spatial2.getWorldBound();
            a.collideWith(boundingBox, results);
            if(results.size()>0)
            {
                return true;
            }
            }
        }
        return false;
    }
```

Výpis 10: Detekce kolize

- **Akce s jednotkami** - pokud chceme pohybovat s jednotkou můžeme ji vybrat dvěma způsoby. V případě volby pouze jedné jednotky, stačí na ní najet myší a následně po kliknutí se provede funkce, která zjistí, zdali došlo ke kliknutí na nějaký objekt na mapě a podle toho se zavolá funkce pro vybrání jednotky. Dále lze jednotkou pohybovat pomocí pravého tlačítka myši, kdy po kliknutí na mapu získáme přesnou pozici a tuto pozici uložíme pro jednotku. Odešle se zpráva na server i s novou pozicí a server tuto zprávu zpracuje a spustí algoritmus pro vyhledání cesty a následně odesílá pozice, na které se má jednotka přemístit a tím dojde k pohybu.

4.3.6 Zobrazení menu a dalších informací

Jak jsem již dříve popsal, pro zobrazení různých informací a vytvoření menu jsem využil tzv. **NiftyGUI**. Pomocí XML souboru jsem nadefinoval různé obrazovky a panely, které slouží pro zobrazení menu pro jednotlivé budovy, jednotky a další. O celý tento proces se stará objekt třídy **HUD**. Zde se nachází funkce, které se provádějí na základě kliknutí na tlačítko na obrazovce. Každé tlačítko má své ID a po kliknutí se provádí funkce. Jsou zde funkce na postavení budov : **buildAltar, buildTrend, buildCastle, buildRangeB,**

buildLaboratory, ale taktéž funkce pro vytváření jednotek : **createSoldier, createZombie, createRanger, createWorker**. Jednoduše řečeno, nacházejí se zde funkce, které obstarávají akci pro všechny tlačítka, které lze ve hře vidět.

Pro přepínání jednotlivých typů menu jsem využil **switch**, který na základě parametru **i** volá příslušnou funkci pro načtení obrazovky z XML souboru.

```
public void setScreenBuilding(int i) {
        switch (i) {
            case 0:
                nifty.fromXml("Interface/Nifty/HelloJme.xml", "hud", this);
                break;
            case 1:
                nifty.fromXml("Interface/Nifty/HelloJme.xml", "asd", this);
                break;
            case 2:
                nifty.fromXml("Interface/Nifty/HelloJme.xml", "CastleHUD", this);
                break;
             ...
      ...
          ...
            case 9:
                nifty.fromXml("Interface/Nifty/HelloJme.xml", "endGame", this);
                break;
            case 10:
                nifty.fromXml("Interface/Nifty/HelloJme.xml", "exit", this);
                break;
        }
    }
```

Výpis 11: Přepínání obrazovek

5 Uživatelská dokumentace

V této kapitole se pokusím o vysvětlení aplikace z pohledu uživatele, tedy hráče. Popíšeme si, jaké jsou herní požadavky, spouštění aplikace, ovládání aplikace a další důležité faktory k úspěšnému hraní hry.

5.1 Herní požadavky

Herní požadavky na tuto aplikaci nejsou nikterak složité. Jedná se o aplikaci, která je vytvořena především pro počítačovou platformu.

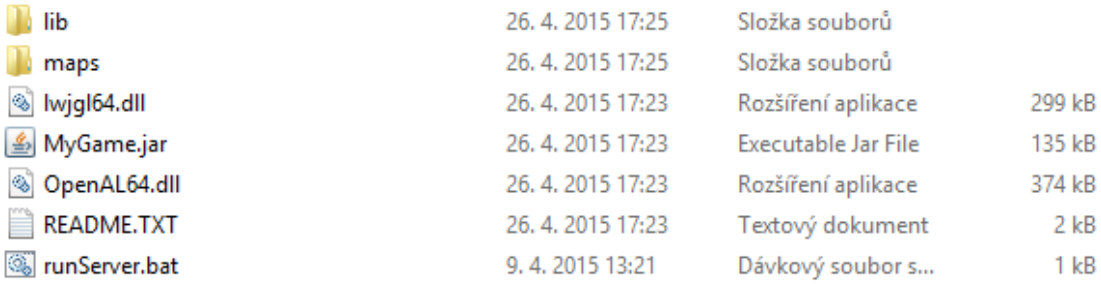
Ke spuštění aplikace je potřeba mít nainstalované Java rozhraní (pokud možno nejnovější). Velikost hry je přibližně 50 MB. Jedná se o velikost pouze klienta. V případě že budete chtít mít na jednom PC současně i server je potřeba mít o 8MB větší kapacitu. V dnešní době, kdy kapacity harddisků dosahují velikosti několika terabajtů je tato velikost zcela zanedbatelná.

Hra oplývá základními grafickými prvky a volně dostupnými 3D modely. V případě načítání těchto modelů, může dojít ke zpomalení hry. Abychom se vyvarovali zbytečným problémům, jsou veškeré modely načteny na začátku spuštění aplikace.

5.2 Spuštění hry

5.2.1 Server

Jak jsem již výše zmínil, abychom zdárně spustili hru, je potřeba mít nainstalované Java rozhraní. Pro úspěšné vytvoření hry a následné připojení klienta je jako první potřeba spustit server. Serverová část se skládá ze souborů uvedených na Obrázku 9.



Obrázek 9: Struktura souborů pro server

Ve složce **lib** se nachází potřebné knihovny ke správnému chodu aplikace. Složka **maps** obsahuje soubor s názvy dostupných map a následně jednotlivé mapy. Dále se zde nachází soubor **runServer.bat**, který slouží k úspěšnému spuštění serveru. Hlavní soubor celého serveru se nazývá **MyGame.jar**. Pro spuštění tohoto souboru docílíte následovně :

1. Klikněte dvakrát na soubor **runServer.bat**. Tento soubor má v sobě již potřebné příkazy, které slouží k úspěšnému spuštění serveru.

2. Otevře se Vám konzole a pokud nedošlo k žádné chybě, tak by konzole při správném spuštění měla vypadat jako na Obrázku 10.

```
C:\Users\Patrik_NTB\Desktop\VšB 6.semestr\BP-Server>java -jar MyGame.jar
Dub 09, 2015 1:31:08 ODP. com.jme3.system.JmeDesktopSystem initialize
INFO: Running on jMonkeyEngine 3.0.10
Dub 09, 2015 1:31:08 ODP. com.jme3.system.Natives extractNativeLibs
INFO: Extraction Directory: C:\Users\Patrik_NTB\Desktop\VőB 6.semestr\BP-Server
Dub 09, 2015 1:31:08 ODP. com.jme3.asset.AssetConfig loadText
WARNING: Cannot find loader com.jme3.scene.plugins.blender.BlenderModelLoader
```

Obrázek 10: Úspěšné spuštění serveru

3. V tuto chvíli je server úspěšně spuštěn a čeká a připojení hráčů.

5.2.2 Klient

Spuštění klienta je o něco jednodušší. Pro úspěšné spuštění hry, je třeba mít funkční server. Hierarchie souborů klienta vypadá jako na Obrázku 11.

lib	25. 3. 2015 17:06	Složka souborů	
lwjgl64.dll	25. 3. 2015 17:05	Rozšíření aplikace	299 kB
MyGame.jar	25. 3. 2015 17:05	Executable Jar File	119 kB
OpenAL64.dll	25. 3. 2015 17:05	Rozšíření aplikace	374 kB
README.TXT	25. 3. 2015 17:05	Textový dokument	2 kB

Obrázek 11: Soubory pro klienta

Na obrázku je zde opět složka **lib** ve které se nachází knihovny. Dále jsou zde soubory pro vykreslení (lwjgl64.dll, OpenAL64.dll). Nás opět zajímá nejdůležitější soubor a tím je **MyGame.jar**. Neplést si se souborem pro server. Aplikace mají stejný název, avšak jsou v různých složkách a plní jinou funkci. Pro úspěšné spuštění klienta stačí pouze 2x kliknout na soubor MyGame.jar. Následně se nám spustí klient a přivítá nás úvodní obrazovka jME3 (Obrázek 12 vlevo).

Zde si nastavíme celou obrazovku, rozlišení obrazovky (doporučuji 800x600), barevnou hloubku a vyhlazování. Pokud jsme vše nastavili dle našich představ, klikneme na tlačítko **Continue** a spustí se nám hra s menu viz. Obrázek 12 vpravo.

5.3 Práce s klientem

V této sekci si popíšeme, jak ovládat klienta po úspěšném připojení na server. Po připojení závisí na klientovi, v jakém pořadí se připojil. Pokud se hráč připojí jako první na server, má o jednu úlohu navíc. V tomto případě musí hráč zvolit mapu. Na výběr má ze dvou možností :

Obrázek 12: Spuštění klienta

1. **Zvolit ze seznamu** - hráč si zvolí mapu z listu, který se nachází vpravo nahoře. Je zde seznam již vytvořených map. Hráč si vybere pomocí kliknutí na název. V případě, že hráč mapu nezvolí, je implicitně nastavena mapa č.1.

2. **Vytvoření nové mapy** - pokud si hráč chce vytvořit svou vlastní mapu, musí kliknout na tlačítko **Create Map (Vytvořit mapu)** a následně bude přesunut do editoru map. Podrobný návod na vytvoření mapy si popíšeme v další části.

Pokud jsme si mapu vybrali ze seznamu klikneme na tlačítko **Start** a v tu chvíli se zobrazí herní plocha, kde čekáme, až bude připraven druhý hráč, a poté se spustí celá hra.

5.3.1 Editace mapy

Při volbě editace mapy budeme přesunuti do editoru. Zobrazí se nám prázdná travnatá plocha do které budeme moci vkládat jednotlivé objekty, které si zvolíme z nabídky viz. Obrázek 13.

Zvolit si můžeme ze tří věcí. První z nich je klasický strom (Tree) a druhá věc je kámen (Stone). Tyto věci nejsou nijak podstatné a slouží pouze ke zkrášlení mapy. Nejdůležitější je zlatá truhla (Gold Chest). Ta slouží k tomu, abychom mohli těžit zlato. Bez této truhly **nebude** možné těžit, proto silně doporučuji při editaci mapy si tuto truhlu do mapy vložit. Po vytvoření mapy klikneme na tlačítko **start** a budeme přesunuti do hry.

Po vytvoření mapy se nám uloží a budeme ji moci při další hře použít znovu. Mapa se uloží s názvem **Map** a číslo následující po poslední vytvořené mapě.

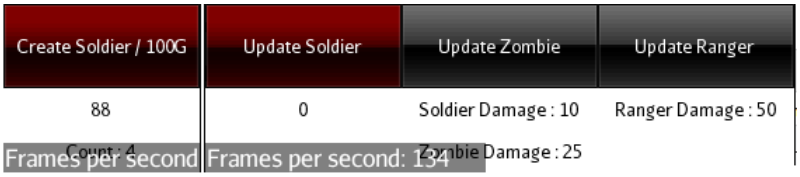
Obrázek 13: Editor map

5.3.2 Stavění a práce s menu

Po úspěšném připojení obou hráčů a potvrzení připravenosti se oběma hráčům načte mapa a spustí se hra. Zobrazí se nám menu, které lze vidět na Obrázku 3.

Jako první bychom měli postavit budovu s názvem Castle. Klikneme tedy na tlačítko **Build castle**, zobrazí se nám náhled budovy a můžeme s ní pohybovat po mapě. Jakmile se nám zalíbí místo k postavení, klikneme levým tlačítkem a budovu postavíme. Pro postavení ostatních budov opakujte stejný postup. V případě postavení budovy GoldTrend musíme na mapě najít zlatou truhlu a tuto budovu postavit pouze v její blízkosti. Jestliže je budova příliš daleko tak zčervená.

Zobrazením menu, které je pro každou budovu jiné, docílíme tak, že klikneme na danou budovu na mapě. Místo základního menu se nám dole zobrazí příslušné tlačítka. Každá budova má jiné možnosti a jiné vytváření jednotek. Pro vytvoření jednotky, či vylepšení nebo dalších jiných akcí je potřeba opět kliknout levým tlačítkem myši na příslušné tlačítko. Při vytváření jednotky vidíme čas, který slouží k zobrazení za jak dlouho bude daná jednotka vyrobena a v případě laboratoře vidíme současné poškození jednotek (Obrázek 14).

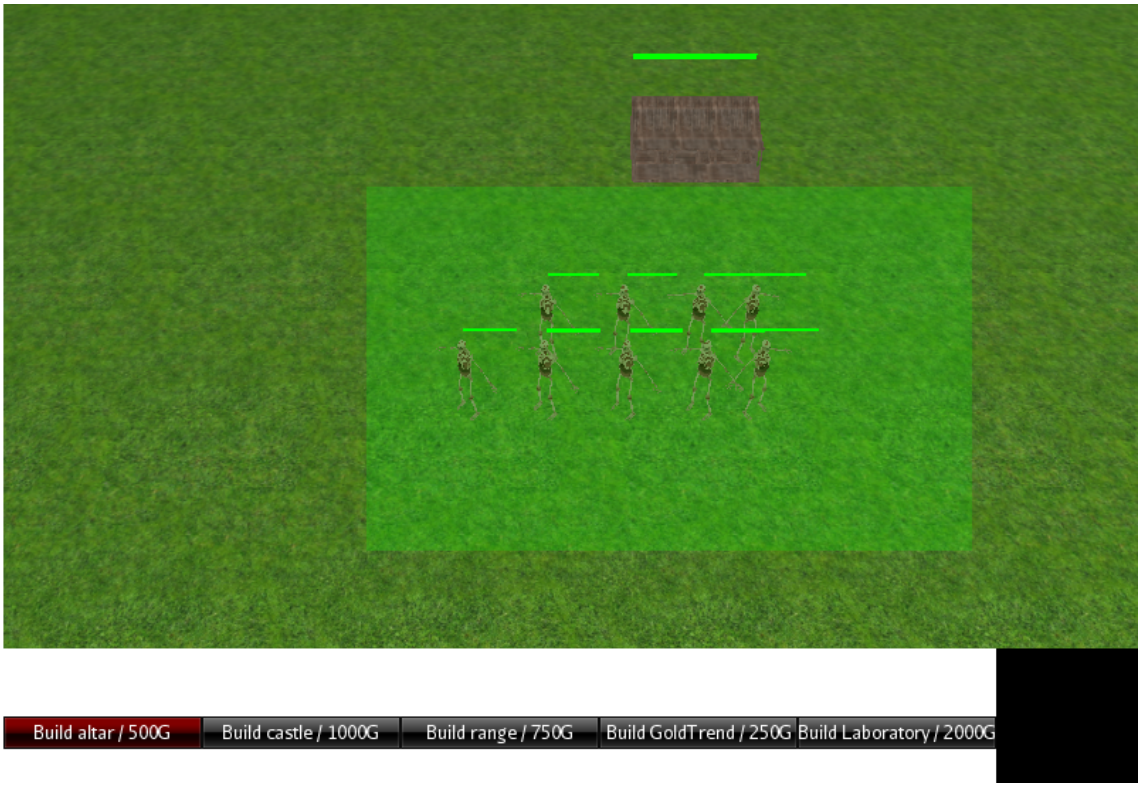


Obrázek 14: Menu budov

5.3.3 Práce s jednotkami

V této podkapitole si popíšeme práci s jednotkami. Označení jednotky docílíme tak, že :

1. **Klikneme na jednotku** - pouhým kliknutím docílíme označení jedné jednotky.
2. **Využití boxu pro označení (select box)** - v tomto případě držíme levé tlačítko myši a označíme počet jednotek viz. Obrázek 15.



Obrázek 15: Označení více jednotek

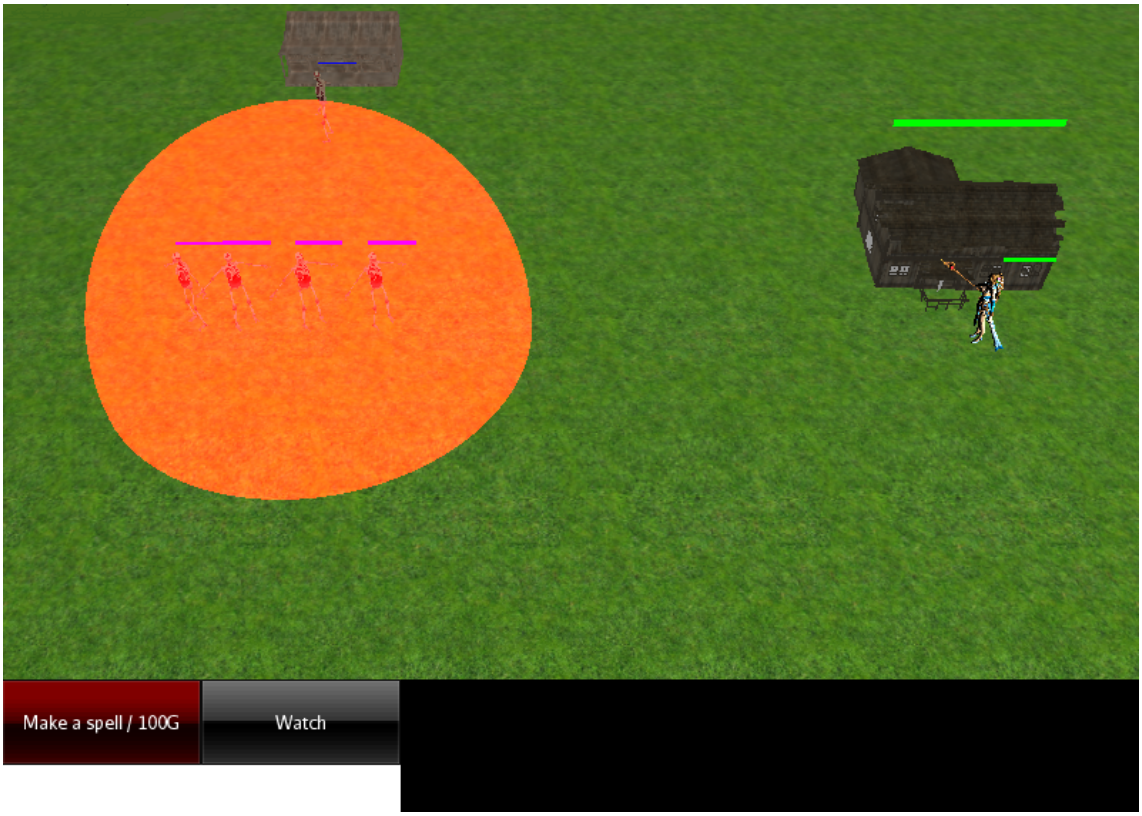
V případě úspěšného označení se nám zobrazí příslušné menu pro jednotky. Pokud označíme pouze jednu jednotku, která bude typu **Ranger**, tak se nám zobrazí menu viz. Obrázek 16. V opačném případě zde bude pouze možnost hlídkování. To samé nastane i v případě, pokud označíme více jednotek.

Obrázek 16: Menu jednotek

V menu vidíme dva typy akcí :

- **Make a spell** (Vytvořit kouzlo) - tato možnost se nám zobrazí pouze u označení jedné jednotky s názvem Ranger. Pokud chceme použít kouzlo, klikneme na tlačítko levým tlačítkem myši a zobrazí se nám plocha, na kterou bude kouzlo použito (Obrázek 17). Pohybem myši si vybereme místo a následně levým tlačítkem kouzlo použijeme. V rozsahu se provede poškození cizím jednotkám a dojde také

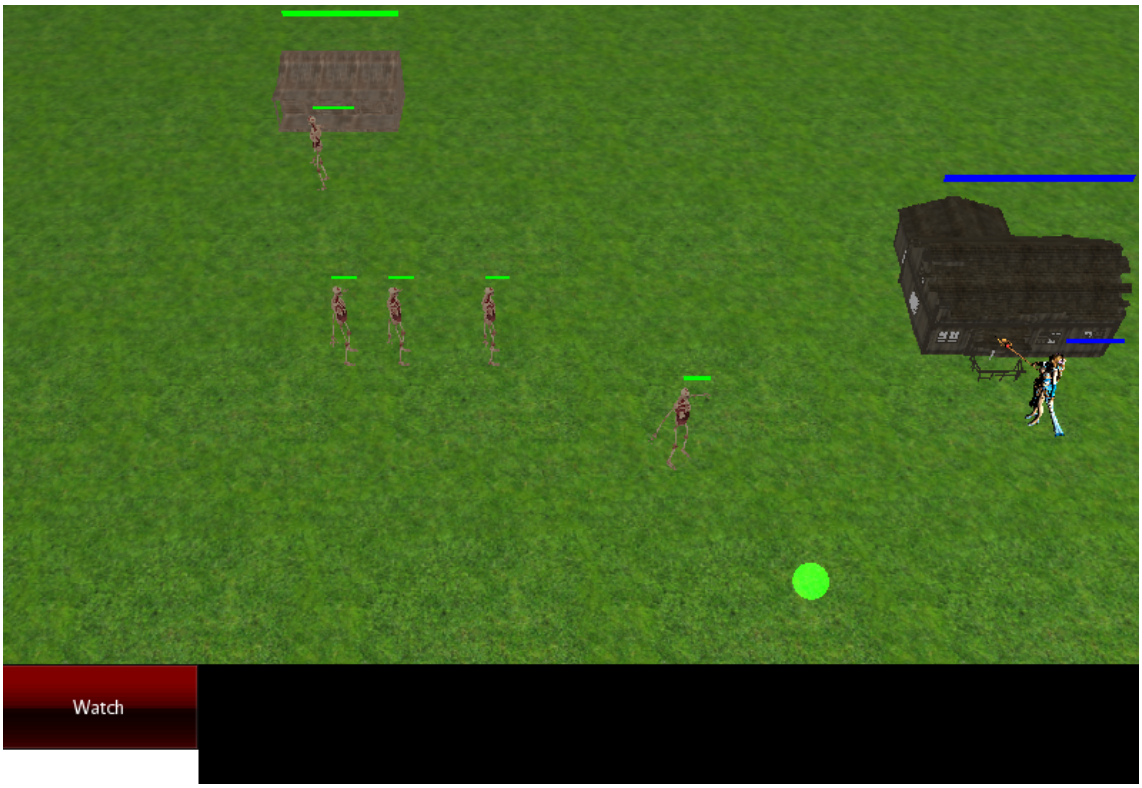
k tzv. **FEAR** efektu (Odstrašení jednotky - jednotka se pohne od středu vyvolání kouzla).



Obrázek 17: Použití kouzla

- **Watch** (Hlídkování) - pokud klikneme na tuto možnost, zobrazí se nám malý zelený kruh, který nám zobrazuje kam budou jednotky chodit. Klikneme levým tlačítkem na mapu a tím označíme bod kam budou jednotky chodit (Obrázek 18). Jakmile dosáhnou tohoto bodu, otočí se a jdou zpět na předchozí pozici. Tato činnost se stále opakuje.

V případě, že chceme s jednotkami pohybovat, tak je nejprve označíme a následně klikneme pravým tlačítkem myši kdekoliv na mapu. Na tuto pozici se nám naše jednotky rozejdou. Tuto stejnou akci opakujte i v případě, že chcete zaútočit na jednotky nebo budovu protihráče. Pravým kliknutím na budovu na ní zaútočíte. To samé platí i pro jednotky.



Obrázek 18: Použití hlídkování

6 Závěr

Cílem této práce bylo vytvořit strategickou hru zařazenou do historického světa. Nachází se zde několik typů budov a jednotek, které proti sobě bojují a cílem je zničit hlavní budovu. Práce se podobá jiným strategickým hrám, protože je zde stejný princip hry a pravidel. Hru jsem si zvolil vytvářet v jazyce Java. Díky této práci se mi rozšířily obzory co se týče programování. Na této práci jsem poznal, že to není nic jednoduchého, když máte za úkol vytvořit plně funkční hru, která by měla být připravena pro hraní kdekoliv.

Při vytváření jsem narážel na spoustu problému, které bylo potřeba vyřešit. Ač se jednalo o běžnou herní logiku, dokázala pěkně potrápit. Pochopit základní principy vykreslování a další co je pomocí jMonkeyEnginu přístupné bylo někdy zdlouhavé, avšak ne nemožné. Řešení problémů se cvičícím mi velmi pomohlo a díky skvělým radám byly některé problémy vyřešeny rychleji. Největší boj byl s těmi nejjednoduššími chybami. Řešení chyby, která nemá nikde žádné řešení ani na internetu a strávit nad tím 2 dny a přitom stačilo změnit jen jeden řádek bylo opravdu náročné. Problémy při práci nastaly také po dlouhém čase stráveném nad hrou, kdy už opravdu nezvládáte vnímat i tu nejmenší chybičku.

Myslím si, že jsem zadání splnil dle požadavků, kde jsem splnil všechny body, které mi byly zadány. Uměla inteligence je zde zavedena jednoduchá, ale je schopná bez problému odehrát celou hru. V případě změny stačí pouze nahradit tuto třídu, což je pro pozdější úpravy určitě jednodušší. Hraní hry je umožněno přes internet tak jak bylo zadáno, což je pro dnešní dobu určitě lepší než hrát jednu hru na jednom a tom samém počítači.

Na přiloženém CD se nachází implementovaná hra a soubory potřebné k jejímu spuštění. Jak jsem již dříve zmínil, ke spuštění je potřeba mít nainstalované Java rozhraní. Dále se zde nachází elektronická podoba písemné bakalářské práce a pro jistotu i odkaz na stažení softwaru pro jMonkeyEngine.

7 Reference

[1] TIOBE Software. *Tiobe Index* [online]. 2015 [cit. 2015-04-16]. Dostupné z: $http$: $//www.tiobe.com/index.php/content/paperinfo/tpci/index.html$

[2] Java (programming language). *Wikipedia.org* [online]. poslední aktualizace 2015-04-19 [cit. 2015-04-16].
Dostupné z: $http$: $//en.wikipedia.org/wiki/Java\_(programming\_language)$

[3] JMonkeyEngine - 3D game Development. *JmonkeyEngine 3.0* [online]. 2009 [cit. 2015-04-16]. Dostupné z: $http$: $//www.jmonkeyengine.org/$

[4] Jme3:advanced:nifty_gui. *JmonkeyEngine 3.0* [online]. poslední aktualizace 2013-08-02 [cit. 2015-04-16]. Dostupné z: $http$: $//wiki.jmonkeyengine.org/doku.php/jme3$: $advanced$: $nifty\_gui$

[5] Laying out the GUI in XML. *JMonkeyEngine 3.0* [online]. poslední aktualizace 2014-04-11 [cit. 2015-04-16]. Dostupné z: $http$: $//wiki.jmonkeyengine.org/doku.php/jme3$: $advanced$: $nifty\_gui\_xml\_layout$

[6] SpiderMonkey: Multi-Player Networking. *JMonkeyEngine 3.0* [online]. poslední aktualizace 2013-10-21 [cit. 2015-04-16]. Dostupné z: $http$: $//wiki.jmonkeyengine.org/doku.php/jme3$: $advanced$: $networking$

[7] A* PathFinding Tutorial. LAGUE, Sebastian. *Youtube.com* [online]. 2014-12-16 [cit. 2015-04-16]. Dostupné z: $https$: $//www.youtube.com/user/Cercopithecan/$

[8] A* search algorithm. *Wikipedia.org* [online]. poslední aktualizace 2015-04-07 [cit. 2015-04-16]. Dostupné z: $http$: $//en.wikipedia.org/wiki/A^*\_search\_algorithm$

A Obsah přiloženého CD

Struktura přiloženého CD se soubory :

- **/KOD** - v této složce se nachází celá programátorská část. Celý vytvořený projekt v enginu jMonkeyEngine.
- **/ODKAZ** - odkaz na stažení softwaru jMonkeyEngine 3.0.
- **/SPUSTITELNE** - zde se nachází spustitelné soubory pro spuštění hry a serveru.
- **/TEXT** - elektronická podoba textu bakalářské práce.