VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Souboj hradů ve 3D

Castle Battle in 3D

2015

Vojtěch Zdzieblo

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vojtěch Zdzieblo**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Souboj hradů ve 3D
Castle Battle in 3D

Zásady pro vypracování:

Cílem práce je vytvořit strategickou hru Souboj hradů v reálném čase v prostředí Internetu. Serverová část bude implementována v jazyce Java, klient bude podporovat platformu Android, která umožní výstavbu budov, vytváření více druhů bojových jednotek a umožní hru více hráčů. Pohled na herní plochu bude v 3D.

Hra bude umožňovat:
1. Mapa bojiště bude odkrývána postupně na základě pozice jednotek.
2. Výstavbu budov, vytváření více druhů bojových jednotek, zlepšování zbraňových systémů.
3. Hru více hráčů v prostředí Internetu.
4. Hru proti jednoduché umělé inteligenci.
5. Program umožní připojení do současně vyvíjeného portálu her v jazyce Java (vývoj portálu je součástí zadání jiné práce).

Práce bude obsahovat:
1. Přehled a porovnání 3D herních rámců pro platformu Android.
2. Implementaci výše popsané strategické hry.
3. Popis řešení s využitím diagramů jazyka UML.
4. 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

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 4. května 2015 .....

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

V Ostravě 4. května 2015 .....

Rád bych poděkoval Ing. Davidu Ježkovi, Ph.D. za cenné rady a připomínky při tvoření této bakalářské práce.

Abstrakt

Tématem mé bakalářské práce je vytvoření 3D strategické hry s možností více hráčů přes internet, ve které dvě proti sobě bojující strany posílají jednotky za účelem zničení nepřátelského hradu. Hra je implementována v jazyce Java s použitím rámce LibGDX, který zaobaluje OpenGL. V první části se věnuji porovnání 3D herních rámců pro platformu Android, dále implementaci hry, popisu řešení s diagramy UML a uživatelské dokumentaci.

Klíčová slova: bakalářská práce, hra více hráčů, souboj hradů, bitva, java, 3d, opengl, android, mlha války, libgdx

Abstract

The theme of my bachelor thesis is creating 3D strategic game with multiplayer over internet, in which two sides fightning against each other are sending units for a purpose of destroying enemy castle. The game is implemented in Java with use of framework LibGDX, which wraps OpenGL. In the first part I am describing comparison of 3D gaming frameworks for Android platform, further implementation of the game, description of the solution with UML diagrams and user documentation.

Keywords: bachelor thesis, multiplayer game, castle battle, fight, java, 3d, openhl, android, fog of war, libgdx

Seznam použitých zkratek a symbolů

GUI	–	Graphical User Interface
OpenGL	–	Open Graphics Library
OpenGL ES	–	OpenGL for Embedded Systems
TCP	–	Transmission Control Protocol
UDP	–	User Datagram Protocol
AI	–	Artificial Intelligence
OBJ	–	Formát 3D modelu
FBX	–	Formát 3D modelu
G3DB	–	Formát obsahující 3D model
JSON	–	JavaScript Object Notation
AR	–	Augmented Reality
GLSL	–	OpenGL Shading Language
HUD	–	Heads-Up Display
FPS	–	Frames Per Second
MVC	–	Model View Controller

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Castle Fight je 3D strategická hra v reálném čase. Hlavním cílem hry je zničit protější nepřátelský hrad a k tomu má každý z hráčů k dispozici tři typy budov. Kasárny, které cvičí rytíře, jeskyně, které cvičí zuřivce, budovu lučištníků, která cvičí lučištníky a obrannou věž, která střílí dělové koule. Budovy je možné dvakrát vylepšit a ty potom cvičí lepší jednotky. Hráči získávají zlaťáky každých patnáct vteřin a nebo za poražení nepřátelské jednotky.

Na výběr je ze dvou typů her, *Single player*, kde se hráč utká s jednoduchou umělou inteligencí a *Multi player*, kde se můžou utkat dva hráči proti sobě.

Hra také obsahuje postupné odkrývání mapy jednotkami a to tak, že nepřátelskou stranu mapy zakrývá mlha. Jakmile jednotka přejde na nepřátelskou stranu, začne svým dosahem zraku odkrývat mlhu a pokud narazí na nepřátelskou jednotku, pustí se s ní do boje. Po odhalení nepřátelské budovy zůstane budova odhalena i pod mlhou. Součástí je také minimapa, která určuje výhled kamery a pozici jednotek a budov na mapě.

Rozhodl jsem se implementovat tuto hru, abych se zdokonalil ve vývoji 3D her, jelikož je to moje první vytvořená 3D hra.

V práci jsou porovnány další Android rámce společně s použitým libGDX, dále popisuji implementaci hry včetně vytvoření efektu odhalování mlhy a komunikaci mezi klienty při hře po síti.

2 3D herní rámce pro Android

Níže popisuji tři mnou vybrané 3D herní rámce pro platformu Android s porovnáním.

2.1 libGDX

LibGDX[3] je rámec založen na jazyce Java s částmi kódu psanými v C a C++ pro lepší výkon. V současné době podporuje systémy Windows, Linux, Mac OS X, iOS a HTML5. Libgdx umožňuje, aby jeden kód aplikace mohl být spuštěn na všech uvedených platformách. Takže stačí aplikaci testovat na PC a v případě potřeby vyexportovat na danou platformu, odpadá tak zdlouhavé nahrávání aplikace do mobilního zařízení a testovaní na něm.

Obstarává přehrávání hudby a zvuků, detekce gest, práci se soubory, vlastní serializaci pomocí JSON, pomocné třídy pro matematické výpočty. Díky podpoře shaderů umožňuje vykreslovat objekty s okolním světlem, odlesky, normal mapping, mlhou a stíny. Obsahuje správce zdrojů, který se stará o správu paměti a načítání souborů potřebných pro chod aplikace.

Rámec má vlastní nástroj pro vytváření projektů. Stačí zadat jméno projektu, název hlavní třídy a nástroj vše vygeneruje za nás. Uvnitř projektu se dále pracuje s rozhraním *Screen*, které má metody *show*, *render*, *resize*, *dispose*, *pause*, *resume* a *hide*, viz sekce Životní cyklus 3.1.1. Nejdůležitější metodou je *render*, která se volá několikrát za sekundu, nejlépe však šedesátkrát, a obsahuje vše co se má v daný snímek vykreslit na obrazovku.

Pro GUI libGDX používá graf scény Scene2D 3.1.3. Ten obsahuje několik komponent, jako jsou popisek, tlačítko, ukazatel průběhu, zaškrtávací políčko, panel a další. Je proto vhodný pro sestavení grafického rozhraní, herního menu nebo HUD. Poradí si i s různými typy akcí (animací), které slouží k zpestření scény.

LibGDX má také podporu knihoven třetích stran jako jsou:

- **Bullet** - podpora fyziky ve 3D
- **Ashley** - entitní systém
- **Box2D** - podpora fyziky ve 2D
- **Kryonet** - TCP a UDP komunikace
- **Spine** - 2D animace pomocí kostí
- **FreeType** - generování fontů

2.2 Rajawali

Rajawali [4] je vyvinut přímo pro plafromu Android a podporuje technologii Gradle. Podporuje vytváření primitivních 3D objektů, normal mapping, detekci kolizí, mlhu, vlastní shadery a další.

Inicializace je jednoduchá a může fungovat v jakékoliv aktivitě. Stačí vytvořit instanci třídy *RajawaliSurfaceView* z kontextu aplikace a nastavit mu implementaci abstraktní třídy *RajawaliRenderer*. Třída *RajawaliRenderer* obsahuje metody *initScene*, *onRender*, *onTouchEvent*, *onOffsetsChanged*. Nejdůležitější je metoda *onRender*, která vykresluje na obrazovku. Metoda *initScene* slouží k naplnění dané scény. *RajawaliSurfaceView* má možnost nastavení FPS.

Na druhou stranu načítání ze souboru už tak snadné není. Při načítání souboru *OBJ* a *FBX* jsem se potkal se spoustou problémů v implementaci rámce, takže se mi model nepodařilo vůbec načíst a zobrazit. Dokumentace k načítání ze souboru je také chabá a zmatená, protože odkazuje na třídy, které již neexistují.

Tento rámec stejně jako libGDX podporuje Gradle, což je nástroj na automatizaci všeho a vychází z nástrojů Ant a Maven. Nejčastější využití je pro sestavení projektů. Výhoda spočívá v tom, že se vývojáři nemusejí starat o knihovny projektů a Gradle jakožto automatizační nástroj, zjistí dostupné aktualizace, stáhne je a připojí k projektu. Knihovny jsou umístěny mimo projekt, tudíž je snadno přenositelný, samozřejmě pouze na místo s přístupem na internet.

Rajawali podporuje tyto knihovny třetích stran:

- **Vuforia** - podpora AR
- **VR** - knihovna pro podporu virtuální reality jako je třeba Google Cardboard.

2.3 jPCT-AE

jPCT-AE [5] je rámec zaměřený pouze na platformu Android. Jeho možnosti nejsou příliš obsáhlé a je poměrně nízko úrovňový. Je velmi jednoduchý, lehký a není tedy těžké si jej osvojit. Umožňuje vytvářet primitivní 3D objekty, podporuje vyhlazování hran neboli *anti aliasing*, okolní světlo objektů, stíny, normal mapping, mlhu a GLSL shadery.

jPCT byl původně rámec vytvořen pro použití na osobních počítačích v jazyce Java, ale od roku 2010 podporuje platformu Android pod názvem jPCT-AE.

Dokáže fungovat v jakékoliv aktivitě a inicializace je velmi jednoduchá, stačí z kontextu aplikace vytvořit *GLSurfaceView* a tomu nastavit instanci rozhraní *Renderer*. Implementace rozhraní Renderer obsahuje metody *onSurfaceCreated*, *onSurfaceChanged* a *onDrawFrame*. Vytváření scény probíhá v metodě *onSurfaceChanged*, tato metoda dále pracuje s objekty jako jsou kamera, textury a 3D objekty. Vykreslování probíhá v *onDrawFrame*, což je metoda, která se vykoná několikrát za sekundu, v nejlepším případě šedesátkrát. Vše se vykresluje do vyrovnávací paměti obrazu, která se následně zobrazí.

Rámec není příliš aktualizovaný, ale jako dobrý základ pro menší projekt určitě poslouží. K rámci je dostupná dokumentace, několik tutorialů a ukázkové aplikace. Komunita není moc veliká, na fóru je registrováno kolem tisíc pět set členů. Na stránkách rámce je registrováno téměř padesát her, které jej využívají a jsou publikovány na Google Play.

Podpora knihoven třetích stran:

- **Bullet** - podpora fyziky ve 3D
- **Bones** - podpora animací pomocí kostí
- **Vuforia** - knihovna společnosti Qualcomm pro podporu AR

2.4 Porovnání

Jako porovnání výkonu jsem udělal dva testy. Vše jsem testoval na zařízení LG Nexus 5 s verzí Android 5.1.

Popis testů:

1. scéna je naplněna čtyři sta modely sfér, vytvořených pomocí metod rámců, s počtem ploch 576 a texturou o velikost 1024 na 512 pixelů.

2. scéna je naplněna čtyři sta modely kasáren, viz obrázek č.16, načtených ze souboru s počtem ploch 286 a texturou o velikost 256 na 256 pixelů.

Objekty v obou testech rotovaly kolem své osy Y. U rámce Rajawali se mi bohužel nepodařilo načíst model ze souboru *OBJ* ani *FBX*. Hodnoty výsledků testů jsou snímky za sekundu (*FPS*).

	libGDX	**Rajawali**	**jPCT**
Test 1	31	13	15
Test 2	28	-	16

Tabulka 1: Porovnání výkonu rámců

Dále jsem vytvořil tabulku s porovnáním některých funkcí a vlastností rámců.

Funkce	**libGDX**	**Rajawali**	**jPCT**
OpenGL ES 3.0	✓	✓	✗
Shadery	✓	✓	✓
Import modelu	✓	✓	✓
Bitmap font	✓	✗	✗
Podpora živé tapety	✓	✓	✗
Animovaný model	✓	✓	✓
Stíny	✓	✗	✓
Bullet	✓	✓	✓

Tabulka 2: Porovnání funkcí rámců

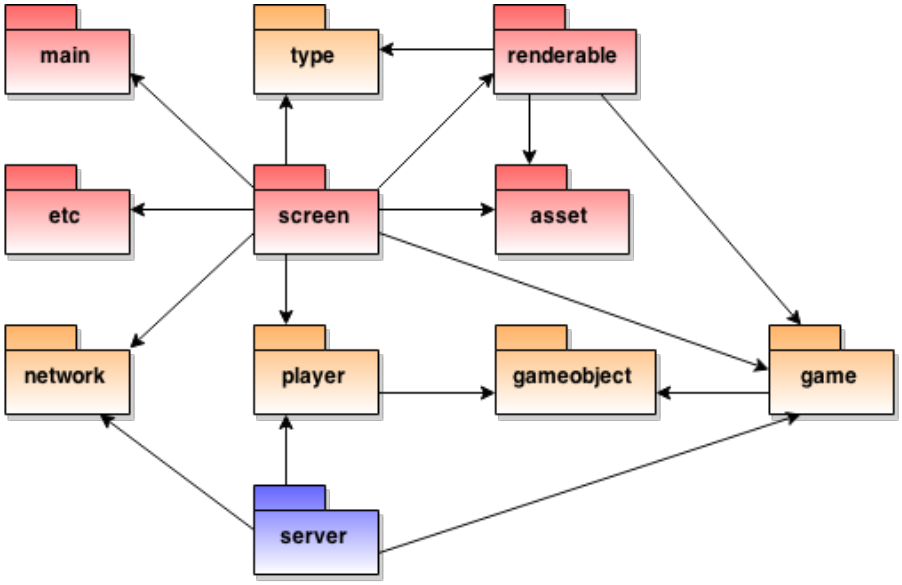
2.5 Zhodnocení

Já osobně jsem zvolil pro vývoj libGDX a zvolil bych jej znovu. Mám s ním už delší dobu zkušenost a díky velké komunitě se rámec stále zdokonaluje a rozrůstá. Po libGDX bych volil jPCT, protože se mi líbí jeho jednoduchost a také to, že neobsahuje moc tříd, ale přesto toho zvládne mnoho. Rajawali se mi neosvědčil při načítání modelu ze souboru, proto jsem ho musel při druhém testu vynechat. LibGDX má oproti Rajawali a jPCT velkou výhodu v podpoře spustitelnosti aplikace na všech populárních platformách.

3 Implementace

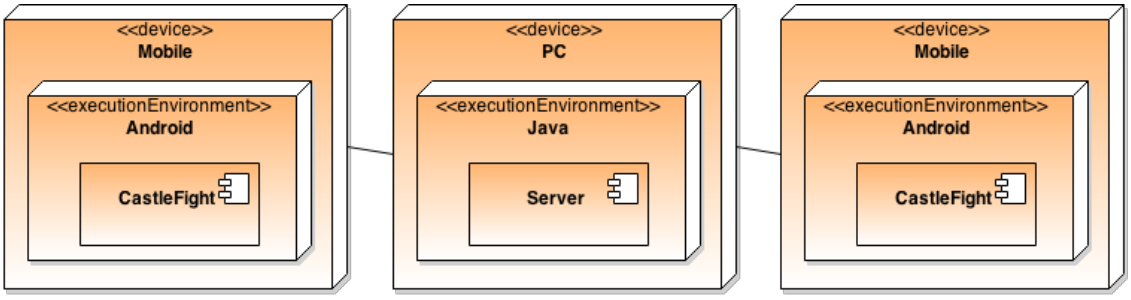
V této části se budu zabývat postupy, které jsem zvolil při tvorbě aplikace. Dále je tato část rozdělena do dvou hlavních částí. Implementace hry a implementace serveru. Vše jsem implementoval ve vývojovém prostředí Eclipse.

Níže je ukázán diagram balíčků, který je rozdělen podle barev a ty označují, ve kterém projektu se nachází. Modrá barva je projekt *CastleFight-server*, červená *CastleFight* a oranžová je *CastleFight-common*. *CastleFight-common* je společná část pro klient a server.

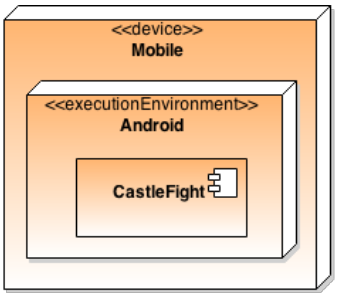


Obrázek 1: Diagram balíčků [7]

První diagram nasazení ukazuje propojení dvou hráčů pomocí serveru, k jehož spuštění je potřeba přístup k síti a prostředí Java. Druhý diagram nasazení ukazuje hru proti umělé inteligenci, u tohoto typu hry není potřeba přístup k síti.



Obrázek 2: Diagram nasazení pro hru více hráčů [7]



Obrázek 3: Diagram nasazení pro hru jednoho hráče [7]

3.1 Implementace hry

3.1.1 Životní cyklus

```
public class Game implements ApplicationListener {
    // ...
}
```

Výpis 1: Implementace aplikačního rozhraní

Aplikace má definovaný životní cyklus [3], upravující její stav. Přístup k událostem životního cyklu je zajištěno implementováním rozhraní *ApplicationListener* a předáním jeho instance startovacím třídám. Odtud už aplikace bude volat určité metody, jakmile nastane nějaká událost. Tabulka níže popisuje metody, které je potřeba naimplementovat.

Metoda	Popis
create()	Metoda je volaná pouze jednou při spuštění aplikace
resize(int width, int height)	Tato metoda se volá při každé změně obrazovky, pokud není hra pozastavená. Volá se také hned po metodě *create()*. Parametry jsou nové rozměry obrazovky v pixelech.
render()	Metoda se volá pokaždé, když by mělo proběhnout vykreslení. Většinou obsahuje i herní logiku.
pause()	Na Androidu je volána, když uživatel zmáčkne tlačítko Domů nebo jakkoliv opustí aplikaci. Na počítači se volá při ukončení aplikace, těsně před *dispose()*.
resume()	Metoda se volá pouze na Androidu a to, když uživatel najede zpět do aplikace.
dispose()	Metoda se volá při ukončení aplikace. Před ní se nejprve volá *pause()*

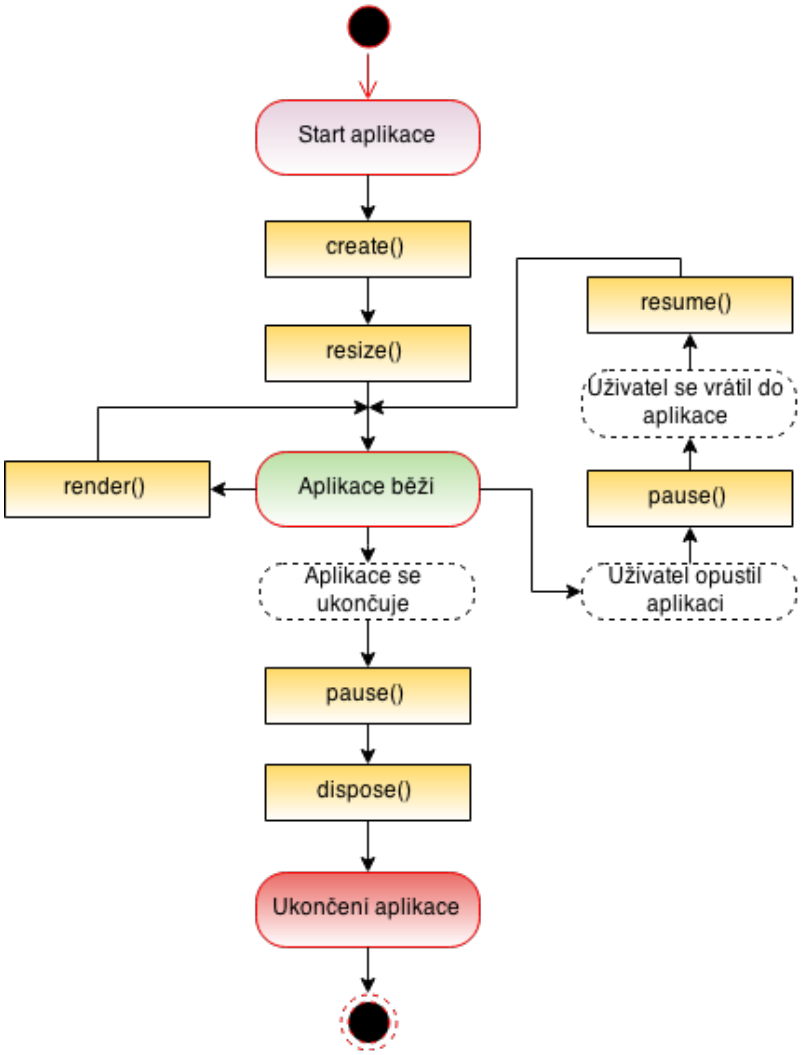
Tabulka 3: Vlastnosti jednotek

3.1.2 Obrazovky

LibGDX pro změnu obrazovek používá rozhraní *Screen*. To obsahuje metody podobně jako životní cyklus 3.1.1. Ve hře používám třídu *Game*, která je součástí libGDX a nabízí podporu změny obrazovek metodou *setScreen*. Pro implementaci jsem si připravil třídu *StageScreen*, ta má naimplementovány základní prvky, které bych musel psát v každé obrazovce zvlášť.

Obrazovka	Popis
MainMenu	Obrazovka, která se objeví po zapnutí hry. Obsahuje tlačítka pro hru jednoho hráče a pro hru více hráčů. Viz obrázek 12
MultiSettings	Po kliknutí na tlačítko pro více hráčů, se objeví tzv. předsíň, ve které uživatel vyplní adresu serveru a svou přezdívku, poté čeká na spojení s druhým hráčem. Viz obrázek 13
MainGame	Obrazovka obsahující veškerou herní logiku 3.1.5 a vykreslování 3.1.4. Viz obrázek 14

Tabulka 4: Popis obrazovek



Obrázek 4: Životní cyklus aplikace [7]

3.1.3 Scene2D

Scene2D je graf pro tvoření aplikací a GUI pomocí hierarchie herců (*actor*). Je velmi dobře vybavený pro určení, vykreslování a zajišťování uživatelského vstupu pro herní menu, HUD a další. Balíček *scene2d.ui* obsahuje spoustu herců a dalších nástrojů pro sestavení GUI. *Scene2D* má však nedostatek v tom, že spojuje model a pohled ze vzoru MVC [3].

Důležitými třídami jsou *Actor*, *Group* a *Stage*.

- **Actor** - třída, která je v grafu jako uzel a obsahuje pozici, velikost, počátek, měřítko, rotaci a barvu.
- **Group** - může obsahovat podřízené herce.
- **Stage** - třída vlastní kameru, kořenovou skupinu a zajišťuje vykreslování herců a rozdělování vstupních událostí.

Balíček *scene2d.ui* obsahuje spoustu připravených komponent, které jsou rozšířením třídy *Actor*. Příklad komponent použitých ve hře:

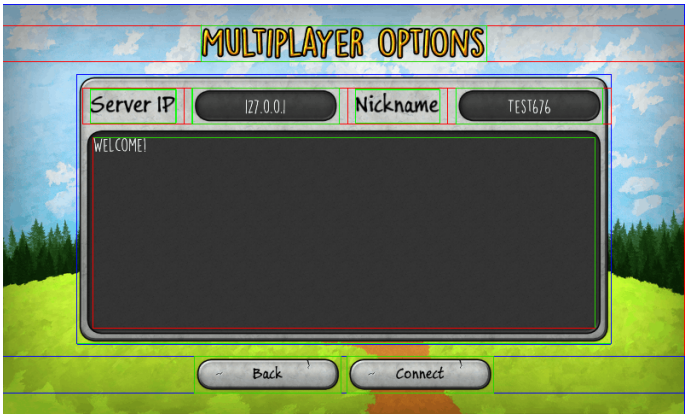
- **Label** - slouží jako popisek
- **Button** - klasické tlačítko, má i rozšíření pro zobrazení textu
- **TextField** - pole, do kterého lze psát
- **ProgressBar** - komponenta, která se procentuálně naplňuje
- **CheckBox** - zaškrtávací tlačítko se stavy zapnuto a vypnuto

Grafické rozhraní hry je tvořeno tabulkou typu *Table*, která je rozšíření třídy *Group*. Do tabulky se postupně přidávají buňky zleva doprava a seshora dolů. Pro přidávání buňky slouží metoda *add()* a pro odřádkování metoda *row()* Takže v případě GUI ve hře, se nejprve přidá horní lišta s penězi a tlačítkem pro odchod do hlavního menu. Poté se přidají popisky pro ladění a následují zaškrtávací tlačítka pro určité funkce. Dále se přidá mapa a tlačítka pro interakci s budovami, jako zničit, vylepšit a stavět, viz obrázek č.14. Celá tabulka se vloží jako herec do instance třídy *Stage*.

Pro správně zobrazení GUI obsahuje libGDX třídu *Viewport*. Ta má několik rozšíření a ve hře používám *ExtendViewport*.

- **ExtendViewport** - tato třída zajistí to, že zobrazovaný GUI bude vypadat stejně na jakékoliv obrazovce, díky tomu že zachovává poměr stran bez černých pruhů, tak, že rozšíří obrazovku do jednoho směru
- **ScreenViewport** - výřez bude vždy totožný s rozlišením obrazovky, ale problém je v tom, že uživatel s větším rozlišením uvidí více hry, než uživatel s menším rozlišením.
- **StretchViewport** - pracuje s virtuální velikostí obrazovky, to znamená, že GUI bude vždy v rozlišení daném virtuální velikostí a výřez se tedy vždy roztáhne na celou obrazovku, bez ohledu na poměr stran, takže nevznikají černé pruhy.

Pro usnadnění jsem si vytvořil třídu *StageScreen*, ze které dědí všechny další obrazovky. Třída se stará o vytvoření instance třídy *Stage* a *Table*, poté nastaví *ExtendViewport* s rozlišením 1280 na 720 pixelů.



Obrázek 5: Obrazovka v módu pro ladění

3.1.4 Vykreslování

Vykreslování provádí metoda *render()*, viz sekce Životní cyklus 3.1.1. Jelikož jsou ve hře použity stíny, probíhá vykreslování v pěti fázích. Při vykreslování se navíc používá hodnota zvaná *delta*, která obsahuje čas, který uběhl od posledně vykresleného snímku. Tato hodnota je důležitá hlavně proto, že pokud hra nepoběží šedesát FPS, tak pomocí delty dopočteme hodnoty, které jsou závislé na čase.

- Z pohledu světla se zapíše hloubka všech jednotek a budov do textury, pomocí upraveného shaderu a zároveň se provede veškerá herní logika jako pohyb jednotek, kolize a aktualizace stavů.
- Provede se vykreslení jednotek a budov normálním způsobem a při tom se využívá textury, do které se v předešlé fázi zapsala hloubka a vykreslí se tak i stín.
- Když už jsou jednotky budovy vykresleny, vykreslí se kolem nich kruh, který určuje dosah viditelnosti, více v sekci Odkrývání mapy 3.1.12.
- Vykreslí se životy jednotek a budov, pokud je však tato volba povolena. Životy jsou reprezentovány jako barevný obdélník s černým ohraničením a zmenšuje se zprava na základě toho, kolik má objekt životů.
- Aktualizuje se GUI a poté se vykreslí, podrobnější informace v sekci Scene2D 3.1.3. Přes metody *act()* a *draw()* volané na třídu *Stage*.

3.1.5 Herní logika

Herní logika probíhá ve třídě *Game*, kvůli hry více hráčů, protože tato část běží na serveru a rozesílá informace o průběhu klientům. Aby se nemusela logika implementovat podruhé pro hru jednoho hráče, použije se třída *Game* akorát místo vzdálených klientů, jsou klienti místní. V klientu pak v metodě *render()*. Třída *Game* provádí pohyb, kolizi s nepřáteli, produkce budov, smrt jednotek, útok, atd. O některých událostech informuje klienty.

Všechno začíná spuštěním vlákna, které obsahuje nekonečnou smyčku a v každé iteraci provádí jisté operace. Nejprve čeká se načte hra obou hráčů, ti hru informují zprávou *LOADED*, viz tabulka č.5. Dále každých patnáct sekund odešle hráčům pravidelný příjem (*income*). Dále cyklem projede všechny budovy, ty aktualizuje a v případě, že budova dosáhla svého produkčního času, vytvoří jednotku podle typu budovy. O tom informuje oba hráče a ti si stav následně aktualizují a jednotku zobrazí. Pokud se jedná o věž, tak se místo jednotky vytvoří projektil, konkrétně dělová koule. Hned poté hra jednotce určí souřadnice jak se dostat k nepřátelskému hradu a ty také pošle hráčům. Následně hra aktualizuje všechny jednotky a to konkrétně jejich pohyb. Pokud však jednotka spatří nepřátelskou jednotku, či hrad, dostane povel zaútočit a informuje o tom hráče. Nakonec už jen zkontroluje, jestli nebyl dobyt některý z hradů. Více informací v sekci Síťová komunikace 3.2.1.

3.1.6 Herní plocha

Herní plocha je obdélníkovitého tvaru a je vytvořena v Blenderu jako model, který se skládá z trávy, cesty a lávy. Uživatel může stavět budovy pouze na trávu. Svět má velikost X $(-400, 400)$ a Y $(-200, 200)$, ale stavět budovy lze pouze na liché souřadnice. Celá polovina herní plochy nepřítele je pokryta tvz. mlhou války, která slouží k tomu, aby hráči neviděli nepřátelské budovy, ihned jak jsou postaveny.

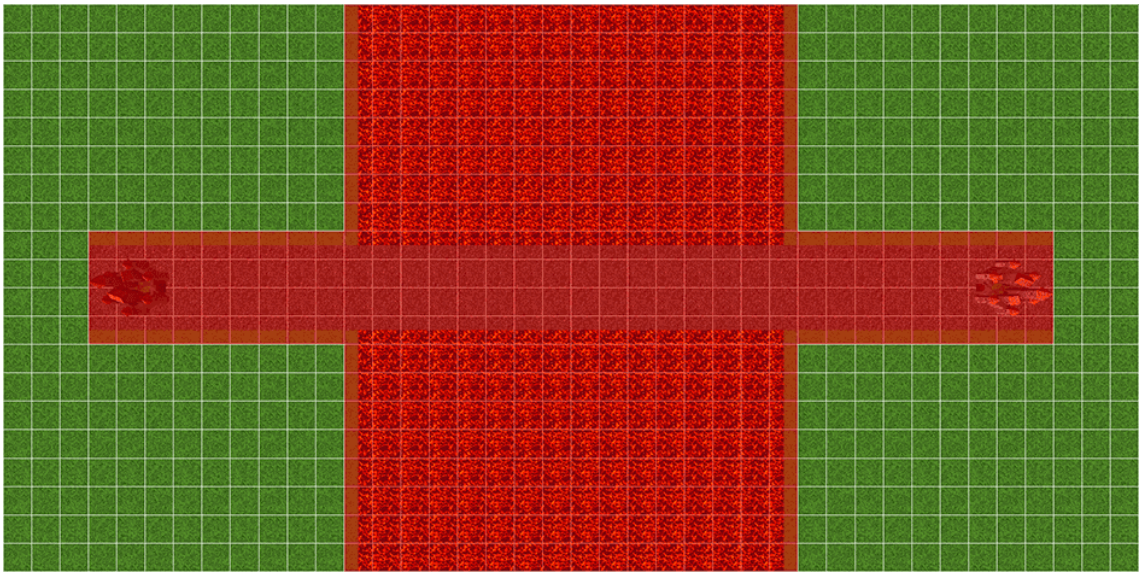
Výpis kódu níže, kontroluje zda není budova na lávě nebo na cestě, pokud je vrací *false*. Dále je třeba zkontrolovat, zda na daném místě již stojí budova.

```
if (((column > 2 && column < 37) && (row > 7 && row < 12)) || (column > 11 && column < 28))
    return false;
```

Výpis 2: Kontrola při stavění budovy

3.1.7 Umělá inteligence

Umělá inteligence se staví na úroveň hráče tak, že rozšiřuje třídu *Player*, která obsahuje přezdívku a instanci implementace rozhraní *Sender*, viz výpis č.7. Umělá inteligence běží na svém vlastním vlákně, kde každé dvě sekundy rozhodne, co postaví. Nejprve se náhodně vybere jedna ze tří budov, které generují jednotky a na tu AI šetří zlaťáky. Jakmile našetří na vybranou budovu, postaví ji a náhodně se vygeneruje další budova, tak to pokračuje pořád dokola.



Obrázek 6: Pohled z vrchu na herní plochu se zvýrazněným zákazem stavění

3.1.8 Budovy

Budovy jsou statické modely a jsou vymodelované mnou v Blenderu [6]. Všechny budovy, kromě hradu, zaujímají jednu buňku v mřížce, hrad zaujímá čtyři. Hlavním úkolem budov je generovat jednotky. Jakmile je budova postavena začne se odpočítávat čas, za který se vygeneruje jedna jednotka a takhle se to opakuje pořád dokola. Výjimku tvoří obranná věž, ta vystřelí dělovou kouli proti jednotce, která vkročí do rádiusu věže. Každá budova lze dvakrát vylepšit a generuje potom silnější jednotky, v případě věže, silnější střely. Budovy vylepšené na první úroveň jsou zbarvené modře a po vylepšení na druhou úroveň jsou zbarvené fialově, vygenerované jednotky jsou zbarvené taktéž. Dále se dají budovy prodat a hráč získá nějakou část zlaťáků, které do výstavby vložil.

3.1.9 Jednotky

Generují se za určitou dobu, která závisí na typu budovy. Jakmile uběhne čas potřebný na výrobu jednotky, je vytvořena a ze všeho nejdříve dostane instrukce, jak se dostat na kamennou cestu a jak se dostat k nepřátelskému hradu. Povely jsou ve formátu souřadnic X a Y a rozesílá je třída *Game*. Jednotky chodí výhradně po hranách buněk mřížky, aby se vyhly kolizi s budovami. Dále jednotky pokračují cestou k nepřátelskému hradu, pokud po cestě potkají nepřátelskou jednotku, ihned na ni zaútočí a pokud ji porazí, pokračuje v cestě. Jakmile se dostane k nepřátelskému hradu začne na něj útočit.

Ve hře jsem řešil problém, kde jednotky na rozdíl od budov se mohou pohybovat libovolně ve světe a tím pádem jejich pozice může nabývat i desetinných hodnot. Budovy jsou na rozdíl od jednotek udržované v poli (mřížce), o rozměrech 20 řádků a 40 sloupců, proto jsem si vytvořil pomocné metody pro převod mezi těmito prostředími. Když po-

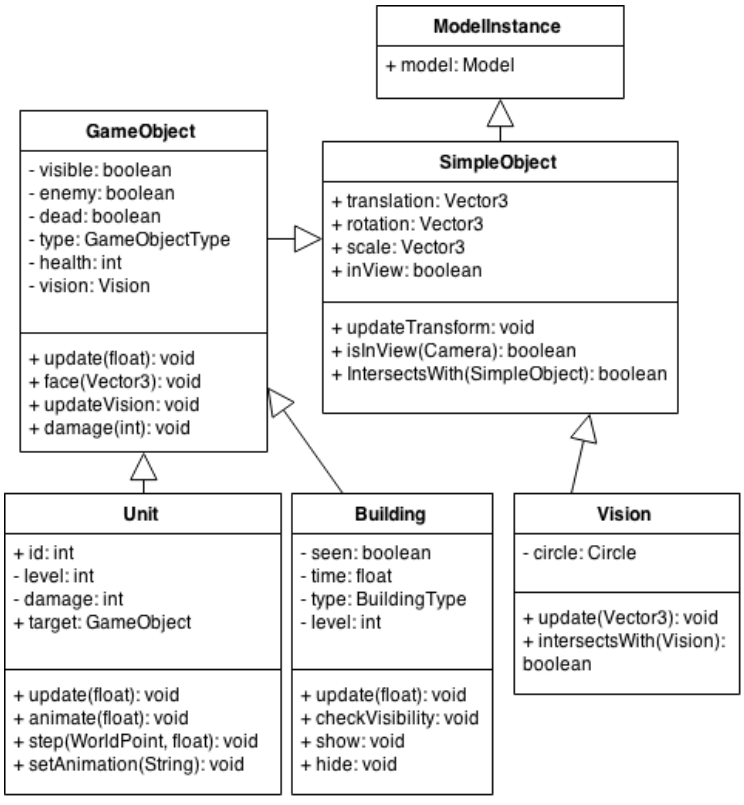
stavím budovu na řádku 7 a sloupci 11 tak ve světových souřadnicích to bude X -170 a Y -50.

Dále jsem řešil způsob otáčení jednotek směrem, kterým jdou. Použil jsem funkci $arctg2(y, x)$, která vrací úhel v radiánech v rozsahu $(-\pi, \pi\rangle$ jenž je sevřený mezi osou x a průvodičem daného bodu, ten se převede na stupně vynásobením s podílem 180 a π a přičtením 180 z něj uděláme číslo z rozsahu $(0, 360\rangle$

$$angle = arctg2(y, x) * \frac{180}{\pi} + 180$$

```
public static WorldPoint getWorldCoordinates (int row, int column) {
    return new WorldPoint(column * 20 - GRID_X + 10, row * 20 - GRID_Y + 10);
}
public static GridPoint getGridCoordinates (float x, float y) {
    return new GridPoint(((int) ((y - 10) / 2) + HALF_GRID_Y) / 10, ((int)((x - 10) / 2) +
        HALF_GRID_X) / 10);
}
```

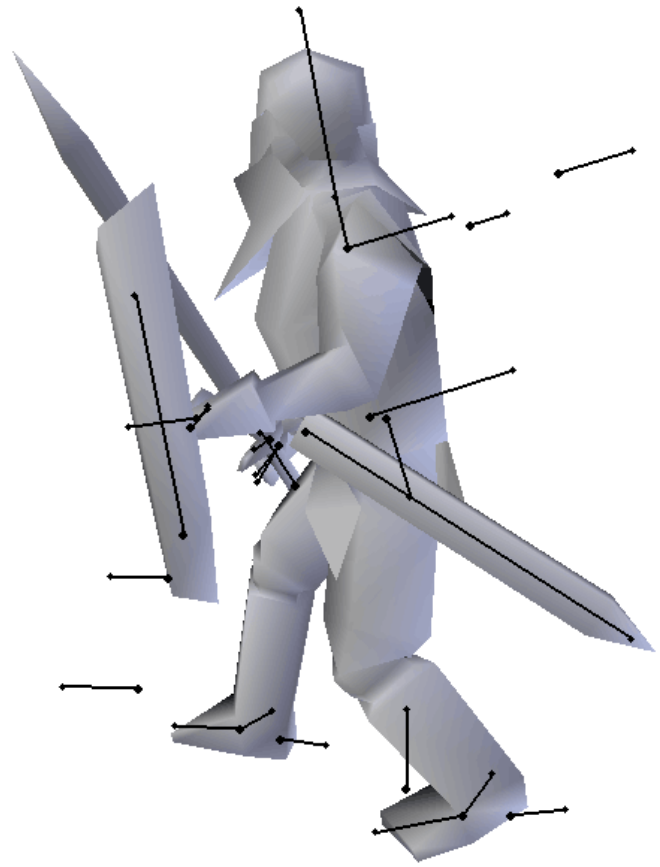
Výpis 3: Metody pro převod z mřížky do herního světa a naopak



Obrázek 7: Třídní diagram dědičnosti budov a jednotek [7]

3.1.10 Animace

Jednotky jsou animovány v Blenderu, kde se animování provádí ukládáním klíčových snímků. Klíčový snímek obsahuje pozici, rotaci a měřítko daných objektů, které se transformují. Mezi klíčovými snímky se potom provádí interpolace, což je matematické vkládání přibližné hodnoty funkce v daném intervalu.



Obrázek 8: Kosti rytíře v Blenderu

Každá jednotka má animace s názvy *Attack*, *Die*, *Idle*, *Hit*, *Pose* a *Walk* [8][9][10]. Ve hře používám pouze *Attack*, *Die* a *Walk*. V kódu se pak o animace stará *AnimationController*, tomu se jako parametr předá *ModelInstance* a voláním metody *update* interpoluje mezi klíčovými snímky a mění transformaci objektů. Třída *Unit* dědí z *ModelInstance*, proto ji můžeme předat pomocí *this*.

```
AnimationController controller = new AnimationController(this);
```

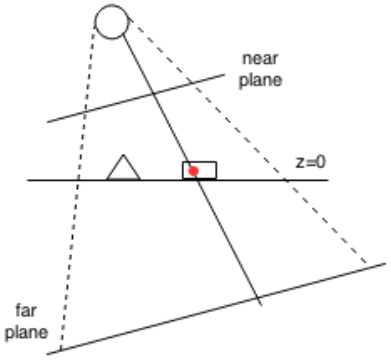
Výpis 4: Vytvoření animace

3.1.11 Výběr objektů

K výběru objektů se používá metoda zvaná *Ray casting* [2]. Pro každou souřadnici na obrazovce existuje nekonečně mnoho 3D souřadnic. Jako příklad uvedu kameru, která má souřadnice [0, 0, 0] jako střed obrazovky a dívá se směrem na -Z. Objekt na [0, 0, -10] bude vykreslen na stejné pozici jako objekt na [0, 0, -20]. Takže střed obrazovky reprezentuje pozice obou objektů. Můžeme si metodu představit jako rovnou čáru, která prochází všemi možnými 3D souřadnicemi pro daný bod na obrazovce. Čára je reprezentována počátečním bodem (*origin*), což je první viditelný bod, a směrem, kterým se kamera dívá (*direction*). Matematicky můžeme vyjádřit jako $f(t) = origin + t * direction$, kde t je vzdálenost mezi počátečním bodem a objektem.

Dále už jen stačí vyhledat objekty, které čára protíná. K tomuto účelu slouží metoda *getBuildingFromRay*, která bere v potaz všechny budovy a postupně je kontroluje.

Na obrázku níže jsou pojmy *near* a *far plane*, které určují rozmezí, ve kterém kamera vidí.



Obrázek 9: Ukázka metody ray casting

3.1.12 Odkrývání mapy

Každá jednotka má instanci třídy *Circle*, která udává, v jakém okruhu kolem sebe vidí. Pokud tento kruh překryje nepřátelskou jednotku nebo budovu, tak ji odhalí a bude viditelná i na mapě. Budovy po prvním odhalení zůstávají viditelné po celou dobu, jednotky však ne. Odkrývání v OpenGL je řešeno tak, že přes nepřátelskou stranu je potáhnuta černá napůl průhledná plocha, ze které se odečte kruh daného objektu, ten je reprezentován jako objekt s texturou masky umístěný těsně nad černou plochou. Svou roli tady také hraje hloubkový zásobník, který funguje tak, že při zápisu do něj se přepíší pouze hodnoty, které jsou dále než ty právě zapisované. Toho je docíleno pomocí barevného zásobníku, který se vypne před vykreslením kruhů, tím dojde k zápisu pouze do hloubkového zásobníku a poté se zase zapne a vykreslí se černá plocha, která se ale nevykreslí na některá místa, protože neprojde testem u hloubkového zásobníku. Ve výsledku je mlha s průhlednými dírami.[1]

```
Gdx.gl.glColorMask(false, false, false, true);
```

Výpis 5: Vypnutí zápisu do barevného zásobníku

3.2 Implementace serveru

Server používá třídu *SocketServer* z balíku *java.net* a běží na portu 23015 a protokolu TCP. Je tvořen nekonečnou smyčkou, která v každé iteraci čeká na připojení klienta. Při připojování klientů vždy počká, až se připojí dva a ty spolu propojí jako oponenty. Veškeré dění se odehrává na serveru a ten odesílá klientům jenom důležité informace o změnách. Pokud jednotka zemře nebo se zaměří na jinou jednotku, server pošle o této změně klientům zprávu a ti si následně hru aktualizují.

Při připojení klienta k serveru musí ze všeho nejdříve odeslat přezdívku hráče a pak čeká na zprávu typu *CONNECTED* společně s přezdívkou protihráče a stranou na které bude hrát.

```
ServerSocket server = new ServerSocket();
InetSocketAddress address = new InetSocketAddress(23015);
server.bind(address);
```

Výpis 6: Vytvoření serveru na portu 23015

3.2.1 Síťová komunikace

Celá komunikace probíhá posíláním třídy *Message*, ta má jako první parametr *CommandType*, což je výčet všech možných informací, které se dají poslat a seznam atributů, které jsou pro každý *CommandType* rozdílné. Třída se při posílání rozloží na text, kde jsou veškeré atributy odděleny středníkem a při přijetí se zase složí a vytvoří se z nich instance třídy *Message*.

Zprávy vypadají například takto, viz tabulka č.5:

NB;AR;14;39 - Vytvoření budovy pro lučištníky na řádku 14 a sloupci 39 v mřížce

NU;AC;0;380;90;true;14;39 - Vytvoření lučištníka na souřadnicích X 380 a Y 90

MO;0;380;90;380;40;280;40;-240;9 - Povely jako souřadnice pro jednotku s ID 0, jak se dostat k nepřátelskému hradu, vždy v pořadí X;Y

Zvolil jsem tento způsob zpráv, protože mi přišla serializace tříd zbytečně náročná na velikost zpráv posílaných přes síť. Více informací v sekci Velikost zpráv 3.2.4

```
public interface Sender extends Closeable{
    public Message get () throws IOException;
    public void send (Message msg) throws IOException;
}
```

Výpis 7: Rozhraní Sender pro síťovou komunikaci

3.2.2 Spojení hry jednoho hráče

Pro hru jednoho hráče jsem vytvořil novou implementaci třídy *Sender*, viz výpis č.7, aby mohl hráč komunikovat s umělou inteligencí a zároveň aby se nemusel vytvářet lokální server na který by se pomocí třídy *Socket* připojili. Od toho je třída *LocalSender*, která odeslanou zprávu přesměruje na příjem jiné instance třídy *LocalSender*. Níže je ukázka kódu vytvoření dvou instancí, které si potom na sebe předají reference. Proces potom funguje tak, že při odeslání zprávy instancí *sender1* se zpráva dostane na příjem instanci *sender2* a zase naopak.

```
LocalSender sender1 = new LocalSender();
LocalSender sender2 = new LocalSender();

sender1.setSender(sender2);
sender2.setSender(sender1);
```

Výpis 8: Přesměrování zprávy třídou LocalSender

3.2.3 Spojení hry více hráčů

Pro vzdálené spojení se serverem používám třídu *SocketSender*, ta je implementací rozhraní *Sender*, viz výpis č.7. Používá třídu *Socket* k příjmu a odesílání zpráv. Při odeslání se zpráva rozloží na text a jednotlivé části se oddělí středník a při příjmu zase složí zpět. Pro tento typ připojení je potřeba přístup k internetu v případě, že je server vzdálený nebo přístup k síti v případě, že je server někde na místní síti.

```
sb.setLength(0);
sb.append(msg.type.command + ";");

for (String string : msg.attributes) {
    sb.append(string + ";");
}
sb.setLength(sb.length() - 1);

out.write(sb.toString());
out.newLine();
out.flush();
```

Výpis 9: Rozložení zprávy a odeslání

Název	Zkratka	Popis
NEW_BUILDING	NB	Vytvoření budovy
NEW_UNIT	NU	Vytvoření jedntky
NEW_PROJECTILE	NP	Vytvoření dělové koule (věž)
DESTROY	DE	Zničení budovy
LOADED	LO	Oznámení o načtení hry
CONNECTED	CO	Oznámení o připojení hráče
DISCONNECTED	DI	Oznámení o odpojení hráče
PING	PI	Zpráva pro zkoušení spojení
USERNAME	US	Přeposlání přezdívky
INCOME	IN	Pravidelné oznámení o příjmu
OK	OK	Potvrzení předešlého dotazu
NO	NO	Odmítnutí předešlého dotazu
UPGRADE	UP	Vylepšení budovy
DEAD	EA	Oznámení o úmrtí jednotky
MOVE	MO	Instrukce o cestě pro jednotku
TARGET_UNIT	TU	Příkaz k zaútočení na jednotku
TARGET_BUILDING	TB	Příkaz k zaútočení na budovu
END	EN	Oznámení o vítězství nebo prohře

Tabulka 5: Výčet všech možných typů zpráv

3.2.4 Velikost zpráv

Nyní ukážu, proč jsem si zvolil vlastní formát zpráv oproti serializaci třídy Message. Pro serializaci je třeba poslat každý detail o třídě včetně proměnných, které vlastní. Musí se odeslat celá cesta balíčku, kde se daná třída nachází a pak až hodnoty.

```
E..H..@. ........
....Y... ..Q.<..\
P...[6.. NU;BE;0;
-160.0;- 50.0;tru
e;7;11..
```

Text oddělený středníky - Celý rámec má 72 bajtů a z toho 32 bajtů obsahují data, zpráva je kompletní.

```
E.....@. ........
......Y. .(......
P...lm.. sr.(com.
zdzieblo .castlef
ight.net work.Mes
sagej... .oL5...L
..attrib utest..L
java/uti l/List;L
..typet. .Lcom/zd
zieblo/c astlefig
ht/netwo rk/Comma
ndType;
```

Serializace - Část zprávy, která prošla přes síť. Tento rámec má 183 bajtů z toho 143 bajtů zabírají data, ale jak je vidět na obrázku, obsahuje jenom cesty k třídám a zpráva není zcela kompletní.

3.3 Optimalizace

3.3.1 Správa paměti

Pro řádné spravování využívaných zdrojů jsem použil *AssetManager*, který je součástí libGDX. Umožňuje asynchronní načítání, kterého lze využít při změně scény a můžeme tak zobrazit průběh načítání. Manažer si své zdroje ponechává, dokud mu neřekneme, že jich má zbavit. Problém nastává na platformě Android, kde může dojít ke ztrátě kontextu, třeba při odchodu z aplikace na plochu. Zařízení potřebuje více paměti a tak musí některé věci z paměti vymazat. Manažer si s tímto problémem chytře poradí a to tak, že při přechodu zpět do aplikace zkontroluje všechny momentálně načtené zdroje a pokud byly některé uvolněny, znovu je načte za nás. Pro lepší zacházení se zdroji jsem si dodatečně vytvořil třídu *Assets*, která obsahuje výčty několika typů zdrojů.

- **Skins** - výčet pro soubory obsahující vzhled pro GUI, viz obrázek č.10
- **Atlases** - výčet pro sloučené textury zvané atlas, viz obrázek č.10
- **Regions** - výčet pro jednotlivé výřezy atlasu
- **Fonts** - výčet pro soubory se znaky
- **Models** - výčet pro 3D modely s texturou

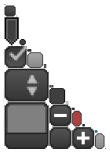
Jednotlivé výčty jsou implementací rozhraní *Asset*, které obsahuje základní informace o zdroji, jako cestu, název, třídu a taky metody pro získání zdroje nebo jeho načtení a smazání z paměti.

```
interface Asset<T> {
    public String getPath ();
    public Class<T> getType ();
    public T get ();
    public AssetLoaderParameters<T> getParameter ();
    public void load ();
    public void unload ();
}
```

Výpis 10: Rozhraní zdroje

3.3.2 Sloučení textur do atlasu

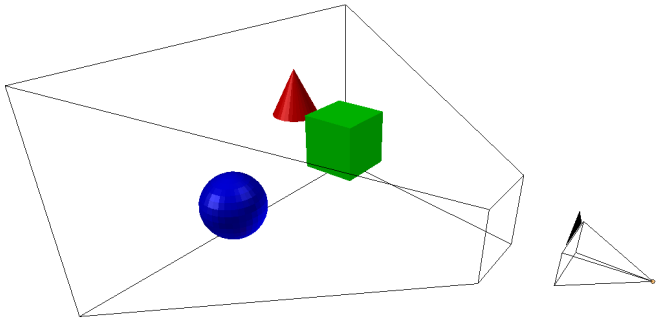
Základem grafických aplikací jsou textury. V OpenGL se textury používají tak, že se nejprve nahrají do paměti grafického čipu a potom při vykreslení objektu se vybere textura, která se připne (*bind*) na objekt. Tento proces stojí čas obzvlášť na grafických čipech chytrých telefonů. Navíc grafický čip má omezený počet tzv. texturovacích jednotek, které slouží k nahrávání textur, proto nemůže být v paměti hodně textur najednou a musejí se střídat. Tento problém částečně řeší sloučení jednotlivých textur do jedné velké textury. Stačí potom nahrát tuto velkou texturu do paměti grafického čipu a budeme moct používat jenom jednotlivé části, pokud vykreslíme jenom určitý výřez. V projektu jsem tento způsob použil na prvky GUI.



Obrázek 10: Příklad atlasu

3.3.3 Ořezávání pomocí komolého jehlanu

Při vykreslování 3D scény, je počet viditelných objektů často menší než celkový počet objektů na scéně. Vykreslování všech objektů, i těch, které nejsou viditelné, může zbytečně plýtvat čas při vykreslení grafickým čipem a tím snížit plynulost hry. Ideální je vykreslit pouze objekty, které jsou opravdu viditelné kamerou a ignorovat ostatní, které jsou mimo. Toho lze docílit pomocí komolého jehlanu (*frustum culling*) [2]. Ten je tvořen z šesti ploch, jmenovitě levá, pravá, horní, dolní, blízká a daleká plocha. Pokud je objekt mezi těmito šesti plochami, tak je viditelný kamerou. Kamera se nachází na pomyslné špičce komolého jehlanu. Další problém nastává, pokud pro ořezávání používáme pouze pozici objektů. Pro modely je pozice většinou ve středu, pokud není model posunut už v základu při modelování, takže se může stát, že objekt je ještě viditelný, ale zároveň se ořeže, jenom proto, že jeho pozice je už mimo kameru. Proto se musíme ujistit, že celý objekt je za kamerou. Mohli bychom kontrolovat každý vrchol objektu, jestli je mimo kameru, ale tento způsob by byl velmi náročný, proto použijeme tzv. ohraničující box (*bounding box*). Ten je zpravidla ve tvaru kvádru a stačí ho vypočítat pouze jednou z vrcholů modelu a používat několikrát později. Skládá se ze středu (*center*) a velikosti v každé dimenzi (*dimensions*). Teď, když známe hrany objektu, můžeme správně určit, zda je viditelný na kameře celý nebo vůbec.



Obrázek 11: Komolý jehlan a kamera

4 Uživatelská dokumentace

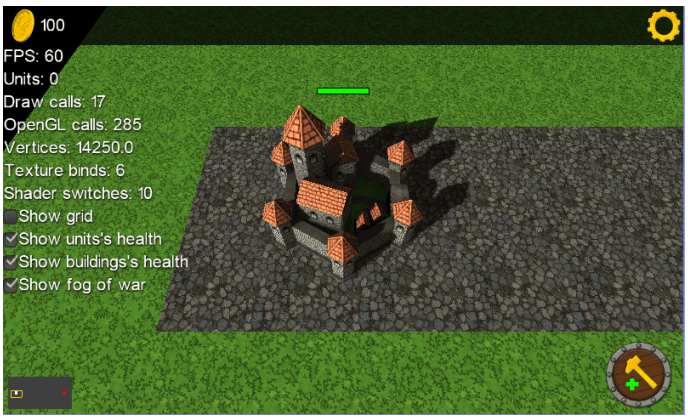
V hlavním menu má uživatel na výběr ze dvou typů her, *Single player*, kde se hráč utká s umělou inteligencí a *Multi player*, kde se přes síť spojí dva hráči a utkají se proti sobě.

Obrázek 12: Ukázka hlavního menu

Po kliknutí na Multi player se hráč přenese do místnosti, kde po zadání přezdívky, adresy a po kliknutí na *Connect* se hra spojí se serverem a čeká na protihráče.

Obrázek 13: Spojení s hráčem přes síť

Jakmile se hra načte, pohled je umístěn na hráčův hrad a souboj může začít.



Obrázek 14: Začátek hry

Oba hráči mají pro začátek 100 zlaťáků a dalších 50 zlaťáků dostávají opakovaně každých 15 sekund. Tento počet zlaťáků si může hráč zvýšit, pokud si vylepší svůj hrad. Ten, jako každá jiná budova jde vylepšit maximálně dvakrát.

Popis tlačítek a ikon:

Stavět - vyvolá dialog s třemi budovami, které může hráč postavit.

Vylepšit - umožňuje za poplatek vylepšit určenou budovu a ta poté bude produkovat silnější jednotky nebo v případě hradu bude hráč dostávat více zlaťáků.

Zničit - zničí zvolenou budovu a hráči vrátí čtvrtinu původní ceny.

Kasárny - budova produkující rytíře.

Jeskyně - budova produkující zuřivce.

Budova lučištníků - budova produkující lučištníky.

Věž - v daném rozsahu automaticky útočí na nepřátelské jednotky.

Zlaťáky - hlavní měna ve hře, kterou lze získat každých 15 sekund nebo poražením nepřátelských jednotek.

Menu - vrátí hráče do hlavního menu.

4.1 Jednotky

Ukázka jednotek z větší blízky a výpis jejich vlastností, hodnoty rychlosti a dosahu zraku jsou jednotky v herním světě. [8][9][10] Po tom co je jednotka vytvořena, ovládá se sama, takže nevnímá žádný uživatelský vstup.

Obrázek 15: Jednotky rytíř, zuřivec a lučištník

Vlastnost	**Rytíř**	**Zuřivec**	**Lučištník**
Životy	70	40	45
Rychlost [jednotky/s]	12	10	15
Síla útoku	10	19	15
Doba útoku [s]	1.1	0.75	1.5
Odměna [zlaťáky]	5	6	7
Dosah zraku	30	20	45

Tabulka 6: Vlastnosti jednotek

4.2 Budovy

Hlavním úkolem budov je cvičit jednotky. V případě obranné věže je to střílení dělových koulí. Níže je tabulka s důležitými atributy. Jednotky neútočí na budovy, pouze na hrad.

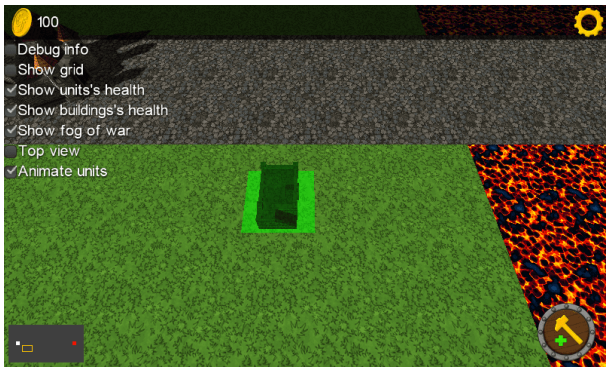
Obrázek 16: Hrad, věž, kasárny, jeskyně a cvičiště lučištníků

Vlastnost	**Hrad**	**Věž**	**Kasárny**	**Jeskyně**	**Cvičiště**
Životy	250	110	95	70	80
Produkuje	-	Dělová koule	Rytíř	Zuřivec	Lučištník
Trvání produkce [s]	-	2	8	10	12
Cena	-	91	52	68	76
Cena 1. vylepšení	200	108	73	89	96
Cena 2. vylepšení	300	133	102	109	121

Tabulka 7: Vlastnosti budov

Pro stavění budov stačí stisknout tlačítko Stavět a zobrazí se dialog s výběrem budovy a cenami. Po zvolení budovy, samozřejmě je potřeba mít dostatek zlaťáků, se ve hře zobrazí napůl průhledná zelená budova. S tou je možné pohybovat kliknutím na ni a tažením, poté co je klik nebo dotek povolen, budova se postaví na vybrané místo. Pokud je budova postavena na nevhodné místo, zbarví se červeně.

Obrázek 17: Dialog s výběrem budov



Obrázek 18: Stavění budovy

U věží je navíc ukázán rádius, který značí, jak daleko věž dostřelí kolem sebe. Je proto vhodné stavět věže k cestě aby na nepřátelské jednotky vůbec dosáhly.

Obrázek 19: Radius věže

5 Závěr

Cílem této práce bylo vytvořit 3D strategickou hru v reálném čase s podporou hry více hráčů na platformu Android. Zadání jsem splnil a hra je plně funkční. Na výběr je ze dvou her, první je proti jednoduché umělé inteligenci a druhá proti vzdálenému hráči, ať už v místní síti nebo v prostředí Internet. Hra je hlavně mířena na mobilní zařízení a pro její spuštění je potřeba Android ve verzi 2.2, ale díky rámci libGDX ji lze také spustit na jakémkoliv počítači, které podporuje prostředí Java.

V bakalářské práci byly popsány a porovnány tři herní rámce, libGDX, Rajawali a jPCT. Nejprve jsem popsal jejich možnosti a potom udělal menší test pro porovnání výkonu, v tomto bodě se ale vyskytly problémy při načítání modelu ze souboru u rámce Rajawali, takže jsem ho musel u testu vynechat. Dále jsem vytvořil tabulku s porovnáním funkcí jednotlivých rámců.

Výsledkem práce je i herní server, který umožňuje spojení dvou hráčů a lze jej spustit na jakémkoliv počítači v prostředí Java s připojením k síti.

Castle Fight je první hra, kterou jsem vytvořil ve 3D. Tvorba této hry mi umožnila zdokonalit se v programování v jazyce Java a blíže seznámit s rozhraním OpenGL. S výslednou prací jsem velice spokojen a plánuji ji v budoucnu více rozšířit. Například přidat nové budovy, jednotky nebo více map. Tento styl hry mě baví a myslím si, že má potenciál zaujmout větší publikum, třeba díky distribuci na Google Play.

6 Reference

[1] DESLAURIERS, Matt. LibGDX Masking. *Github Wiki* [online]. 2014 [cit. 2015-04-24]. Dostupné z: https://github.com/mattdesl/lwjgl-basics/wiki/LibGDX-Masking

[2] XOPPA, B.V. *XOPPA* [online]. 2013 [cit. 2015-04-24]. Dostupné z: http://blog.xoppa.com/basic-3d-using-libgdx-2/

[3] SWEET, Nathan. Libgdx wiki. *Github Wiki* [online]. 2012 [cit. 2015-04-25]. Dostupné z: https://github.com/libgdx/libgdx/wiki

[4] THOMAS, Ian a Jared WOOLSTON. Rajawali wiki. *Github Wiki* [online]. 2012 [cit. 2015-05-04]. Dostupné z: https://github.com/Rajawali/Rajawali/wiki

[5] OLSEN, Egon. *JPCT wiki* [online]. 2009 [cit. 2015-05-04]. Dostupné z: http://www.jpct.net/wiki/index.php/Main_Page

[6] FLEISCHAUER, Mike. 3D models and animation from Blender to LibGDX. *Game From Scratch* [online]. 2014 [cit. 2015-04-25]. Dostupné z: http://www.gamefromscratch.com/post/2014/01/19/3D-models-and-animation-from-Blender-to-LibGDX.aspx

[7] REJNKOVÁ, Petra. *Příklady použití diagramů UML 2.0* [online]. 2009 [cit. 2015-04-30]. Dostupné z: http://uml.czweb.org/index.html

[8] WU, Clement, NIKOLAUS a BOTANIC. Animated Knight. In: *OpenGameArt* [online]. 2011 [cit. 2015-04-30]. Dostupné z: http://opengameart.org/content/animated-knight

[9] WU, Clement, NIKOLAUS a BOTANIC. Animated Archer. In: *OpenGameArt* [online]. 2011 [cit. 2015-04-30]. Dostupné z: http://opengameart.org/content/animated-archer

[10] WU, Clement, NIKOLAUS a BOTANIC. Animated Bezerker. In: *OpenGameArt* [online]. 2011 [cit. 2015-04-30]. Dostupné z: http://opengameart.org/content/animated-bezerker

7 Seznam příloh

Obsah CD

Tabulka popisuje obsah jednotlivých složek na disku CD ROM.

Adresář	Popis
/bin/Android	Apk soubor pro instalaci na zařízení Android
/bin/Desktop	Jar soubor pro spuštění hry na PC
/bin/Server	Jar soubor pro spuštění serveru na PC
/src/**CastleFight**	Hlavní zdrojové kódy hry
/src/CastleFight-**android**	Zdrojové kódy pro spuštění na Android
/src/CastleFight-**desktop**	Zdrojové kódy pro spuštění na PC
/src/CastleFight-**common**	Zdrojové kódy společné části
/src/CastleFight-**server**	Zdrojové kódy serveru
/thesis	Bakalářská práce

Tabulka 8: Obsah CD