UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2017 Martin Lepeška

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Herní 2D engine v Javě

Martin Lepeška

Bakalářská práce

2017

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin Lepeška**
Osobní číslo:	**I14128**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Herní 2D engine v Javě**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je porovnat vybrané technologie a vytvořit v jazyce Java vlastní knihovnu pro tvorbu 2D síťových počítačových her.

V teoretické části budou porovnána dostupná aplikační rozhraní pro tvorbu grafických aplikací a knihoven, zejména herních. Dále budou představeny vybrané herní engine, souborové formáty používané při tvorbě 2D her a popsány základní algoritmy např. pro detekci kolizí, síťovou komunikaci atd.

V praktické části bude vytvořen vlastní 2D herní engine za použití technologie OpenGL, který bude umožňovat zobrazování rastrové grafiky, přehrávání hudby, ošetření myši a klávesnice, detekci kolizí a komunikaci po síti za pomoci protokolu UDP. Na závěr bude vytvořena ukázková RPG hra na vytvořeném engine.

Rozsah grafických prací:

Rozsah pracovní zprávy: **cca 35 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

KISZKA, Bogdan. 1001 tipů a triků pro jazyk Java. Brno: Computer Press, 2009. ISBN 978-80-251-2467-3.

TREGLIA, Dante. Game programming gems 3. Hingham Mass.: Charles River Media, c2002. ISBN 1584502339.

SHREINER, Dave., Graham. SELLERS, John M. KESSENICH a Bill. LICEA-KANE. OpenGL programming guide: the official guide to learning OpenGL, version 4.3. Eighth edition. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 0321773039.

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

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

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. 5. 2017

podpis autora

Martin Lepeška

PODĚKOVÁNÍ

Děkuji mému vedoucímu, Ing. Zdeňku Šilarovi, Ph.D., že mi umožnil pracovat na tomto projektu, který je mi rozptýlením i výzvou. Dále děkuji mé rodině za oporu a trpělivost, které při tvorbě této práce bylo velmi zapotřebí. Nakonec bych chtěl poděkovat svým přátelům, zejména Jakubu Jakoubkovi, kteří se mnou dokázali udržet krok při plánování a snění.

ANOTACE

Práce se zabývá tvorbou 2D herního enginu v Javě. V první části jsou představeny vybrané herní enginy a rozhraní pro tvorbu grafických aplikací a knihoven. Následně jsou představeny použité souborové formáty. Druhá část se zabývá základními algoritmy a návrhem praktické části.

KLÍČOVÁ SLOVA

engine, hra, Java, OpenGL, UDP

TITLE

2D Java game engine

ANNOTATION

This work is about developing 2D game engine in Java. First part introduces selected game engines and interfaces for creation of graphical applications and libraries. Additionaly, there are introduced used file formats. Second part deals with basic algorhitms and design of engine.

KEYWORDS

engine, game, Java, OpenGL, UDP

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

UDP	User datagram protocol
LWJGL	Lightweight Java Game Library
RPG	Roleplaying game
GPU	Graphics Processor Unit
PC	Personal Computer
API	Application Programming Interface
GUI	Graphical User Interface

1 ÚVOD

Cílem práce je vytvořit 2D herní engine v programovacím jazyce Java a následně demonstrovat jeho funkcionalitu na ukázkové RPG hře.

Důvodem, proč se zabývám touto problematikou, je můj sen z dětství naprogramovat vlastní hru, která by dosahovala kvalit herních titulů, jako jsou Baldur's Gate nebo Planescape Torment. Na projektu s touto problematikou jsem pracoval již na střední škole, nicméně jsem z technických důvodů musel začít znova. Naštěstí mi zůstaly zkušenosti, které v tomto projektu mohu zúročit.

Engine obstarává vykreslování rastrové grafiky, přehrávání hudby, ošetření vstupu z myši a klávesnice, detekci kolizí, komunikaci po síti za pomoci protokolu UDP a práci s dialogy.

V první kapitole jsou představeny vybrané herní engine. Cílem první kapitoly je poukázat na fakt, že pro různé herní žánry jsou vhodné odlišné knihovny se značnými rozdíly v možnostech a zaměření.

Druhá kapitola obsahuje přehled aplikačních rozhraní pro práci s grafickou kartou. Účelem této kapitoly je vysvětlit čtenáři základní rozdíly a odůvodnit, proč bylo jako aplikační rozhraní v praktické části práce zvoleno OpenGL.

Třetí kapitola popisuje souborové formáty, které jsou používány pro ukládání herních součástí. V této kapitole je také vysvětleno, jakým způsobem jsou jednotlivé formáty načteny a použity.

Čtvrtá kapitola je zaměřena na tvorbu herního engine, popis hlavních tříd a algoritmů, které jsou v praktické části použity. Podrobnější informace o implementaci jsou uvedeny v přílohách.

Pátá kapitola pojednává o síťové komunikaci pomocí protokolu UDP a tvorbě hry s podporou pro multiplayer ve vytvořeném herním engine.

2 HERNÍ ENGINE

Mezi herním engine a herní knihovnou leží tenká hranice. „*Herní stroj představuje seskupení programového kódu, který provádí úkoly spjaté s během hry"*[1]. Herní engine je možné považovat za knihovnu, ale na rozdíl od standartní knihovny nabízí ucelenou sadu nástrojů a funkcí potřebnou pro tvorbu hry. Příkladem takových nástrojů může být například editor herních úrovní, grafy výkonosti nebo, jako v této práci, editor dialogů.

2.1 Unreal Engine[1]

Jedná se o dílo firmy Epic Games. První verze vznikla v roce 1998 pro hru Unreal. Dnes Epic Games nabízí čtvrtou verzi engine za poplatek ostatním vývojářům her, simulátorů a filmů (Obr. 1).

Obr. 1 – Logo Unreal Engine [2]

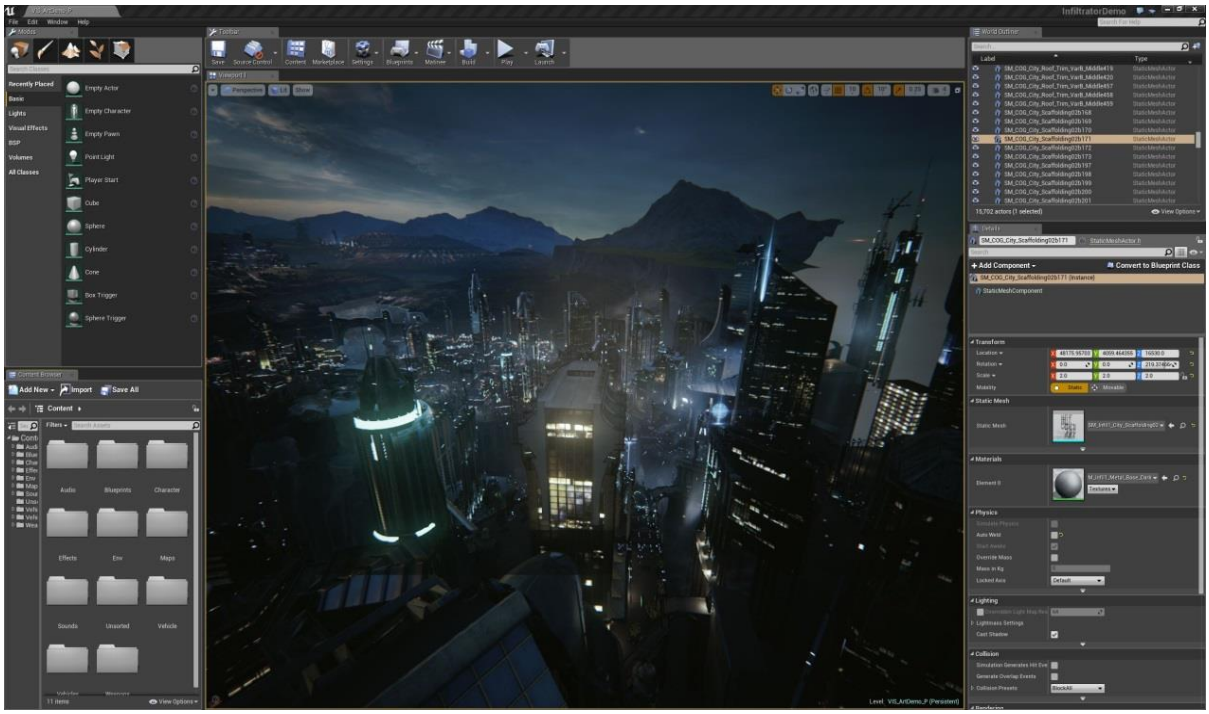
Jako aplikační rozhraní pro práci s GPU Engine využívá DirectX 12, OpenGL 3.3 nebo případně Vulkan, díky kterému podporuje většinu počítačových, konzolových a mobilních platforem. Uživateli je poskytnuta bohatá sada funkcí a nástrojů (viz. Obr. 2) od prohlížeče herních částí po podporu virtuální reality. Herní studia mohou navíc dokupovat a prodávat vytvořené herní části pomocí služby Asset Store [2].

Hlavní nevýhodou je chybějící podpora pro starší platformy. Unreal Engine nepodporuje starší verze OpenGL ani DirectX 9, na což doplatí například majitelé Windows XP.

Licence pro komerční využití je poskytnuta zájemci zdarma, ale každé čtvrtletí musí zaplatit 5% z výdělku nad $3000. K licenci je dodán I kompletní C++ kód. Programátor tedy může při vývoji přidávat a upravovat komponenty [2].

Mezi proslulé herní tituly, které využívají tento engine, patří například Bioshock Infinite, Borderlands nebo Dishonored.

[1] Více na: https://www.unrealengine.com



Obr. 2 – Editor levelů pro Unreal Engine [2]

2.2 Unity[2]

První verze Unity vyšla v roce 2005 pouze pro OS X. Dnes je Unity jeden z nejrozšířenějších a nejznámějších multiplatformních herních engine pro Indie hry (Obr. 3).

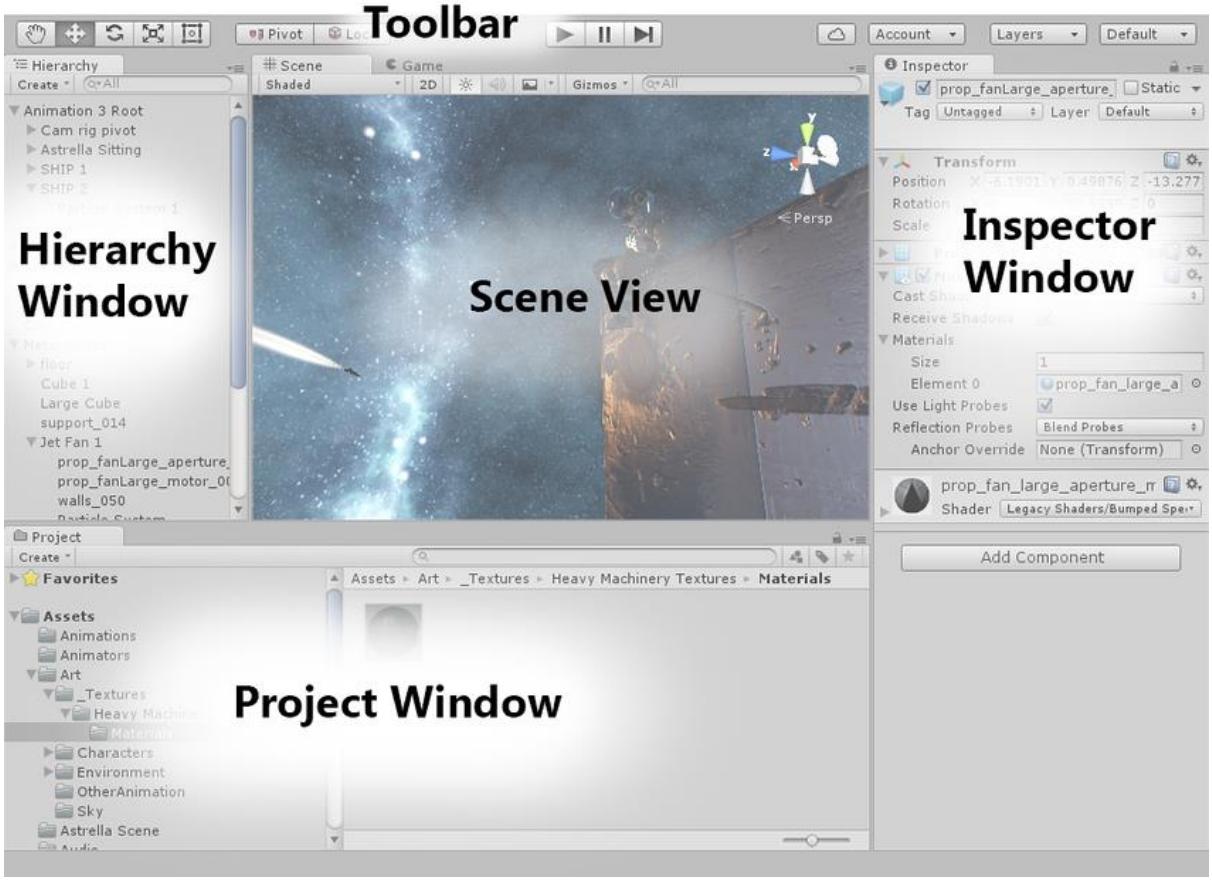
Obr. 3 – Logo Unity 5 [3]

Podobně jako Unreal Engine je Unity napsáno v C++. Při tvorbě hry ale vývojář využívá uživatelsky přívětivější C# nebo Javascript. Jako grafická rozhraní lze použít DirectX, Vulkan i OpenGL. Unity také poskytuje podobné nástroje (viz. Obr. 4) a služby jako Unreal Engine [3].

Případnému zájemci jsou nabídnuty různé typy licencí. Pro osobní využití je Unity volně dostupné, ale pro komerční využití je vývojář povinen platit měsíční poplatky, ke kterým se vztahuje i jedna z velkých nevýhod Unity – zdrojové kódy jsou zpřístupněny až v profesionální verzi [3].

[2] Více na: https://store.unity.com/

Mezi známé hry, které jsou vytvořeny za pomoci Unity, patří Hearthstone, Wasteland 2 a Pokémon GO.



Obr. 4 – Rozložení Unity editoru [3]

2.3 Infinity Engine

Tento engine vytvořila firma Bioware v roce 1998 pro hru Baldur's Gate. V průběhu dalších čtyř let v něm byly vytvořeny další hry, z nichž se nejvíce proslavilo Planescape: Torment. V roce 2012 engine vzkřísilo herní studio Beamdog, které v následujících letech přepracovalo většinu her využívajících tuto knihovnu. Zdrojové kódy engine nikdy nebyly zveřejněny, ale vznikla open-source implementace GemRB[3], která je volně dostupná pod licencí GPL (Obr. 5) [4].

Engine je stvořen pro simulaci pravidel deskové hry Dungeons and Dragons (2. edice). Proto je v něm implementován sofistikovaný systém dialogů, díky kterým se hry využívající tuto knihovnu proslavily.

[3] Více na: https://sourceforge.net/projects/gemrb

Obr. 5 – Logo Open source implementace Infinity Engine [4]

Engine podporuje tzv. 2,5D. Jedná se o vykreslování 2D isometrických obrázků, při němž je třeba hlídat vzájemnou pozici mezi objekty scény, aby nebyl obrázek, který je v popředí scény, vykreslen za obrázkem v pozadí.

Oproti původní verzi zmodernizované verze knihovny podporují kromě operačního systému Windows i Mac OS, Android a linuxové distribuce.

2.4 RPG Maker[4]

Tento program byl v roce 1997 vytvořen Japonskou firmou ASCII Corporation (dnes Enterbrain). Nejnovější verze tohoto nástroje vyšla v roce 2015. Záměrem tvůrců je poskytnout lidem bez zkušeností s programováním způsob, jak snadno vytvářet jednoduché adventury a RPG hry (Obr. 6) [5].

Obr. 6 – Logo programu RPG Maker [5]

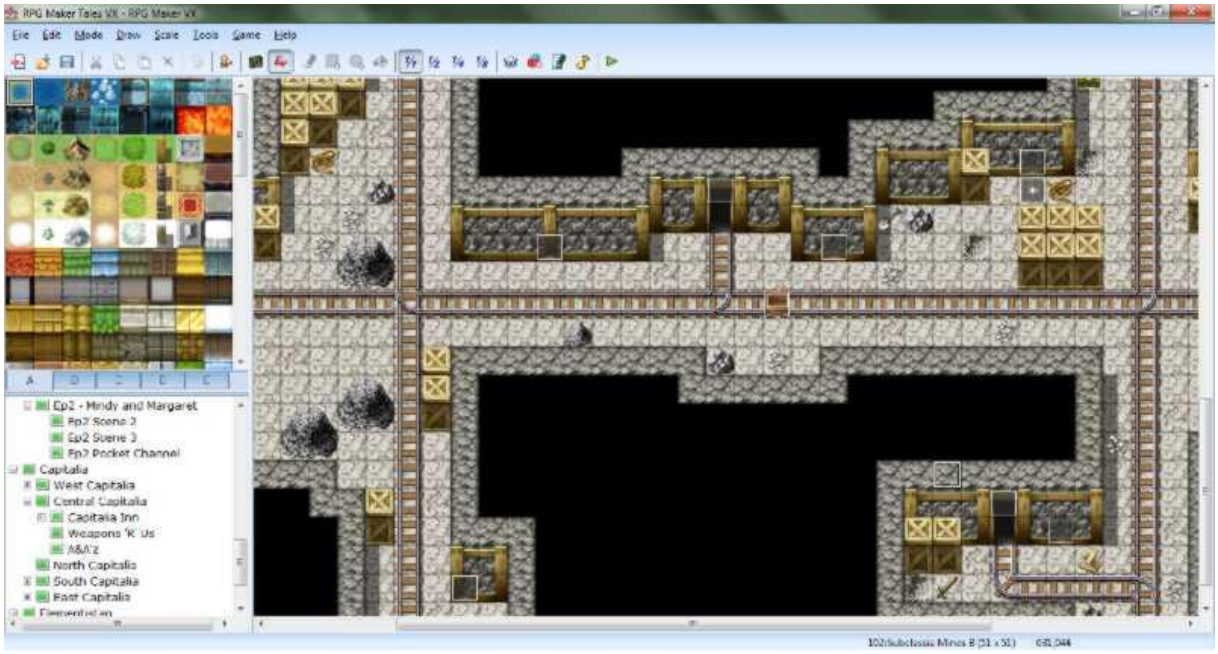
Nejnovější verze podporuje nejrozšířenější počítačové a mobilní platformy a umožňuje i export do HTML 5. Starší verze, které jsou stále dostupné, ale cílily zejména na operační systémy Microsoft Windows a herní konzole PlayStation [5].

Uživateli je poskytnut editor levelů, který využívá tzv. dlaždice. Herní prostor je poskládán ze stejně velkých obrázků, které společně utváří celek, například místnost (viz. Obr. 7). Součástí editoru je i vestavěný editor dialogů, animací a grafických prostředků, včetně základní sady.

Zájemce o program musí zaplatit $79.99 za nejnovější verzi. Pro každou nižší verzi, je cena o $10 nižší. Zdrojové kódy programu nejsou zveřejněny. Vznikla ale řada open source nástrojů, které jej napodobují, například projekt EasyRPG[5][5].

[4] Více na: https://www.rpgmakerweb.com

Většina her, které vznikly v RPG Maker, je známá pouze v Japonsku, kde vznikly. Jako příklad lze uvést herní sérii od tvůrců programu – RPG Tsuküru.



Obr. 7 – Screenshot programu RPGMaker VX [5]

2.5 Shrnutí

Tabulka 1 obsahuje shrnutí, které platformy výše představené engine podporují. Pokud je příslušné políčko v tabulce zelené, platforma je plně podporována. Žluté políčko značí pouze částečnou podporu. Červené značí nekompatibilitu. Podobně jsou v tabulce 2 orientačně shrnuty funkce těchto knihoven, kde zelená značí, že engine funkci poskytuje.

Tabulka 1 – Podpora operačních systému mezi engine

	Unreal Engine	Unity	Infinity Engine	RPG Maker
Windows	Novější, než Windows XP			
Mac OS				
Linuxové distribuce				Od verze MV
Mobilní OS				
HTML 5				
Konzole				Omezený výběr

[5] Více na: http://www.easyRPG.org

Tabulka 2 – Přehled základních funkcí engine

	Unreal Engine	Unity	Infinity Engine	RPG Maker	Vlastní engine
Podpora 2D					
Podpora 3D					
Kolize					
Fyzikální modely					
Rozšířená podpora pro AI					
Podpora pro hru více hráčů					
Editor úrovní					
Podpora dialogů					
Podpora multimédií (zvuky, videa)					
Online služby (např. Statistiky)					

3 APLIKAČNÍ ROZHRANÍ PRO PRÁCI S GPU

Grafické API (Application Programming Interface) je standardizovaná sada funkcí, tříd a struktur, které může programátor použít. Implementace částí API závisí na operačním systému nebo výrobci HW, pro který je rozhraní určené. V této kapitole jsou představeny API, které jsou běžně používány při tvorbě her a herních engine.

3.1 OpenGL

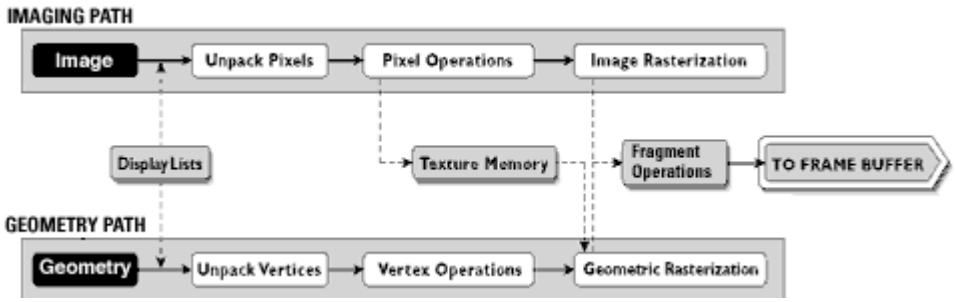
Open Graphics Library bylo vytvořeno firmou Silicon Graphics v roce 1992. Jednalo se o zdokonalenou verzi API IRIS GL (Integrated Raster Imaging System Graphics Library), která je volně dostupná a nezávislá na zvoleném operačním systému a programovacím jazyce [6].

V současnosti je poslední verzí OpenGL 4.5, jehož funkce jsou srovnatelné s DirectX 11. Vzniklo také zjednodušené API OpenGL ES pro vestavěná zařízení a později WebGL pro webové aplikace.

Následuje ukázka (Zdroj. kód 1), jak by byl vykreslen obrázek na souřadnicích [0;0] pomocí knihovny ve verzi 1.0.

```
// Pripojeni textury (data obrazku ulozena v pameti GPU)
glBindTexture(GL_TEXTURE_2D, tex1.getID());

glBegin(GL_QUADS);
      glTexCoord2f(0, 0);
      glVertex3f( 0, 0);
      glTexCoord2f(0, 1);
      glVertex3f(0, tex1.getHeight(), 0);
      glTexCoord2f(1, 1);
      glVertex3f(tex1.getWidth(),tex1.getHeight(), 0);
      glTexCoord2f(1, 0);
      glVertex3f( tex1.getWidth(), 0, 0);
glEnd();
```

Zdroj. kód 1 – Vykreslení textury v OpenGL 1.0



Obr. 8 – tzv. OpenGL „pipeline“ [7]

Podobně jako DirectX se OpenGL s postupem let vyvíjelo a umožňovalo programátorům více zasahovat do tzv. „pipeline.“ Jedná se o postup (Obr. 8), kterým OpenGL zpracuje zadané souřadnice, ze kterých vygeneruje tzv. „fragmenty,“ prostor, který je následně obarven přidělenou barvou z textury nebo jiného zdroje [7].

Ve verzi 2.0 bylo umožněno vývojářům programovat vlastní shadery pro manipulaci se souřadnicemi a fragmenty. Ve verzi 3.1 byla abstrakce posunuta na nižší úroveň. Uložení souřadnic do paměti GPU a napsání vlastních shaderů se stalo nezbytným krokem pro vykreslení jakéhokoliv objektu. Ačkoliv tyto kroky přidělávají programátorovi práci, umožňují účinnější využití grafické karty než univerzální pipeline z původní verze. Zdroj. kód 2 obsahuje zjednodušený příklad pro vykreslení obrázku ve verzi OpenGL 3.1.

```
// INICIALIZACE
float width = texture.getWidth();
float height = texture.getWidth();

float[] vertexPositions = new float[]{
                width,  -height,    0f, 1f,
               -width,   height,    0f, 1f,
                width,  -height,    0f, 1f,
                width,   height,    0f, 1f
};

float[] textureUVCoordinates = {
            0f, 0f,
            0f, 1f,
            1f, 0f,
            1f, 1f
};

// Ulozeni poli do pameti GPU
...
// VYKRESLENI
// Pripojeni struktury, ktera symbolizuje skupinu bufferu
glBindVertexArray(vertexArrays.get(0));

// Pripojeni bufferu, ktery obsahuje vertexPositions do pameti GPU
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffers.get(0));
// Predani informaci o bufferu se souradnicemi
glVertexAttribPointer(0, 4, GL_FLOAT, false, 0, 0);

// Pripojeni bufferu, ktery obsahuje textureCoords do pameti GPU
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffers.get(1));
glVertexAttribPointer(1, 2, GL_FLOAT, false, 0, 0);

// Pripojeni textury (data obrazku ulozena v pameti GPU)
texture.bind();

glDrawArrays(GL_TRIANGLE_STRIP, 0, coordCount);
```

Zdroj. kód 2 - Vykreslení textury v OpenGL 3.1

3.2 DirectX

"Microsoft DirectX poskytuje sadu API pro tvorbu her a dalších vysoce výkonných multimediálních aplikací [8]*."*

DirectX vznikl v roce 1994 pro operační systém Microsoft Windows 95 v době, kdy vývojáři nepoužívali žádné standardizované rozhraní pro práci s grafickou kartou. Se vzrůstajícím množstvím možných komponent pro PC rostl i čas, který programátor musel strávit implementací podpory pro hardware. Cílem projektu bylo přenést břemeno tvorby podpory z vývojáře na tvůrce HW a motivovat tak vývojáře, aby tvořili hry pro Windows [9].

Během let byl DirectX postupně vyvíjen a obohacován o nové funkce, jako například komprese textur, programovatelné shadery ale i podpora pro hru více hráčů po síti nebo 3D prostorový zvuk.

V současnosti (2017) je DirectX ve verzi 12. Tato verze přináší nízko úrovňovou abstrakci GPU, což umožňuje paralelní využití jednotlivých částí a celkové zvýšení účinnosti grafické karty [10].

Následuje zjednodušený Zdroj. kód 3, který pomocí DirectX 12 vykreslí trojúhelník na obrazovce. Pro zkrácení je v příkladu uveden pouze zlomek zdrojového kódu pro inicializaci a vykreslení.

```
// INICIALIZACE
struct Vertex
{
XMFLOAT3 position;
};

// Definice souradnic pro trojuhelnik
Vertex triangleVertices[] = {
{ 0.0f, 0.25f, 0.0f },
{ 0.25f, -0.25f, 0.0f },
{ -0.25f, -0.25f, 0.0f }}
};
// Ulozeni souradnic do tzv. Vertex Bufferu, jehoz adresa je nasledne
// ulozena do objektu VertexBufferView
…
// Nastaveni struktury s prikazy pro GPU
m_commandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// VYKRESLOVANI
// Predani prikazu do fronty GPU
ID3D12CommandList* ppCommandLists[] = { m_commandList.Get()};
m_commandQueue->ExecuteCommandLists(_countof(ppCommandLists),
ppCommandLists);
```

Zdroj. kód 3 – Vykreslení trojúhelníku pomocí DirectX 12 [8]

3.3 Vulkan

V roce 2013 oznámila firma AMD, že vytváří nové grafické API Mantle umožňující účinnější využití grafických karet, které společnost prodává. Hlavní myšlenkou Mantle bylo vytvořit rozhraní abstrahující na nižší úrovni GPU a umožňující tak programátorovi řídit vykreslovací proces [11].

V roce 2015 se firma rozhodla předat projekt konsorciu Khronos Group, které se v současnosti stará i o OpenGL. Mantle se stal základem nového API Vulkan, které je na rozdíl od DirectX12 navržené tak, aby jej šlo implementovat na PC, konzolích i mobilních platformách [12].

V současnosti již byla v některých rozšířených engine, například Unreal Engine a Unity, implementována podpora pro Vulkan. Cenou za výhody API je komplikovaný kód, který je třeba napsat i pro jednoduché úlohy, jako je zobrazení trojúhelníku.

3.4 Zvolené API

Jedním z cílů engine, který vznikl v rámci této práce, je podporovat vysoké množství grafických karet, včetně těch zastaralých. Proto je v projektu implementována podpora pro dnes již přežilé OpenGL 1.4 a moderní OpenGL 3.3. Mezi zvolenými verzemi lze nalézt nejvíce rozdílů. Implementace podpory dalších verzí, například OpenGL ES 2.0, by tedy nebyla náročná. Navíc se není třeba obávat, že by API nebylo podporováno operačním systémem.

4 SOUBOROVÉ FORMÁTY

V této kapitole jsou stručně popsány souborové formáty, se kterými engine vytvořený v rámci tohoto projektu pracuje. Ostatní engine mohou využívat některé z těchto formátů, ale i mnoho dalších, jako například skripty popisující chování herních prvků, formáty pro uložení textury 3D modelů, nebo binární soubory s uloženým stavem hry.

4.1 Obrázky

2D grafika je ukládána dvěma základními způsoby. Vektorová grafika ukládá informace o obrazu prostřednictvím základních objektů, například úseček, křivek nebo trojúhelníků. Rastrová grafika se skládá z tzv. pixelů, které reprezentují jeden bod obrazu. Vývojáři 2D her většinou pracují s rastrovou grafikou, protože je jednodušší vykreslit předpřipravený obrázek, než jej rekonstruovat.

Engine vznikající v rámci této práce podporuje standartní formáty rastrové grafiky, JPEG, BMP, PNG a GIF. Pro další formáty byla implementována experimentální podpora. Načtený obrázek ve výsledném projektu reprezentuje třída Sprite. Ve Zdroj. kódu 4 je předvedeno, jak uživatel načte a zobrazí jednoduchý obrázek.

```
Sprite obrazek = new Sprite("relativni/cesta/obrazek.png");
// Vykresli obrazek na souradnicich [0;0]
Core.addDrawable(obrazek);
```

Zdroj. kód 4 – Příklad použití třídy Sprite

Načtení obrázků do paměti ve formátu použitelném pro OpenGL obstarává knihovna *STB*[6], součást knihovny *LWJGL*[7], které budou věnovány příští kapitoly. Uživatel také může předat data prostřednictvím instance třídy BufferedImage z balíčku java.awt.image.

4.2 Fonty

Jedná se o sadu znaků, pomocí kterých je vykreslen text. Podobně jako běžné obrázky lze font uložit ve vektorové a rastrové podobě. Engine podporuje načítání fontů ve formátu TrueType. Tento formát ukládá informace o znacích pomocí křivek.

Engine pro práci s fonty OpenGL opět využívá knihovnu *STB*, díky níž ze souboru načte informace o velikosti a vzájemném odsazení jednotlivých znaků. Následně vytvoří texturu obsahující předkreslené znaky ve velikosti, kterou si zvolí uživatel. Ukázka, jak font načte uživatel, je uvedena ve Zdroj. kódu 5.

[6] Více na: *http://github.com/nothings/stb*
[7] Více na: *http://www.lwjgl.org*

```
// Nacte pro pouziti font.ttf o velikosti 12px a podporou pro ceske znaky,
// jsou-li dostupne.
FontManager.loadFont("/relativni/cesta/font.ttf", "fontID", 12,
IFont.CZECH_CHARACTERS)
// Nasledne lze k instanci fontu pristupovat pomoci FontManager.getFont()
IFont font = FontManager.getFont("fontID");
```

Zdroj. kód 5 – Příklad načtení fontu

4.3 Audio

Bity reprezentující zvuk obvykle bývají uchovány v zakódovaném formátu, který popisuje rozložení a účel dat v souboru. Formát také může specifikovat algoritmus, kterým jsou původní data převedena do komprimované podoby. V případě komprimace rozlišujeme, zda jsou při ukládání ztracena méně podstatná data nebo ne. Software nebo hardware, který má na starosti zakódování dat, se nazývá „kodek".

Pro načítání zvuků je použita knihovna *Paulscode Soundsystem*[8] společně s kodeky, které umožňují použití formátů OGG, WAW, MOD, XM a S3M. Pro přehrávání audia knihovna přistupuje k 3D zvukovému rozhraní OpenAL, které jí poskytuje *LWJGL*. V případě, že OpenAL není k dispozici, jsou využity funkce poskytované standartní knihovnou Javy. Zdroj. kód 6 předvádí, jakým způsobem uživatel může načíst a přehrát audio soubor.

```
// Instance predpripravena pro bezne pouziti
SoundSystem paulscodeSound = SoundManager.getSystem();
// Argumenty: nazev zdroje, cesta ke zdroji, nekonecna smycka
pauslcodeSound.backgroundMusic(„menu", „menu.ogg", true);
```

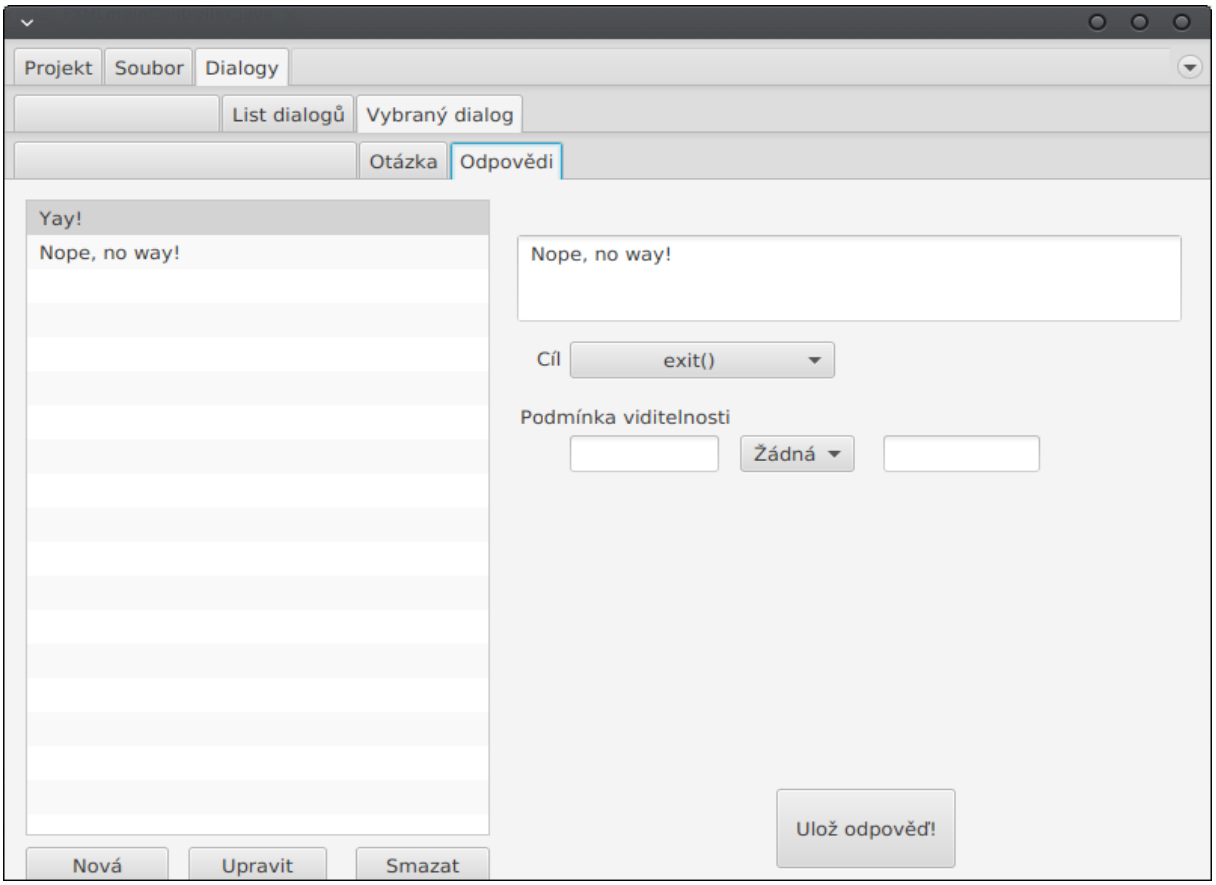
Zdroj. kód 6 – Příklad přehrání audio souboru

4.4 Dialogy

Engine podporuje načítání dialogů prostřednictvím knihovny *DialogEditor*[9], jenž vznikla jako vedlejší produkt při práci na projektu. Spoluautorem knihovny je spolužák Jakub Jakoubek, který vytvořil jednoduché GUI pro vytváření dialogů (Obr. 9).

Dialogy jsou uloženy prostřednictvím formátů XML a JSON. Pro práci s těmito formáty jsou použity knihovny JDOM[10] a GSON [11]. V XML souborech je uložena neměnná struktura a text dialogů, v JSON souborech jsou uchovávány hodnoty proměnných. Jeden dialog v XML reprezentuje text, který je hráči zobrazen, a soubor možných odpovědí, kterými může hráč zareagovat. Zvolená odpověď může spustit další dialog, nebo ukončit rozhovor. Ve Zdroj. kódu 7 je předvedena jednoduchá ukázka uloženého dialogu.

[8] Více na: *http://www.paulscode.com*
[9] Více na: *https://github.com/Mytrin/DialogEditor*
[10] Více na: *http://jdom.org/*
[11] Více na: *https://github.com/google/gson*



Obr. 9 – GUI knihovny DialogEditor

```
<dialog id=" hostinsky-start">
      <event source="hostinsky">
            <text>Vítej cizinče, co by sis přál?</text>
      </event>

      <responses>
            <response target="hostinsky-pivo">
                  <text>Dej mi nějaké to pivo!</text>
            </response>

            <response condition="GREATER_THAN" value1="$Player.Money"
      value2="5" target=" hostinsky-medovina">
                  <text>Chci medovinu!</text>
            </response>

            <response target="exit()">
                  <text>Neměl byste sklenici mléka?</text>
            </response>
      </responses>
  </dialog>
```

Zdroj. kód 7 – XML kód jednoduchého dialogu

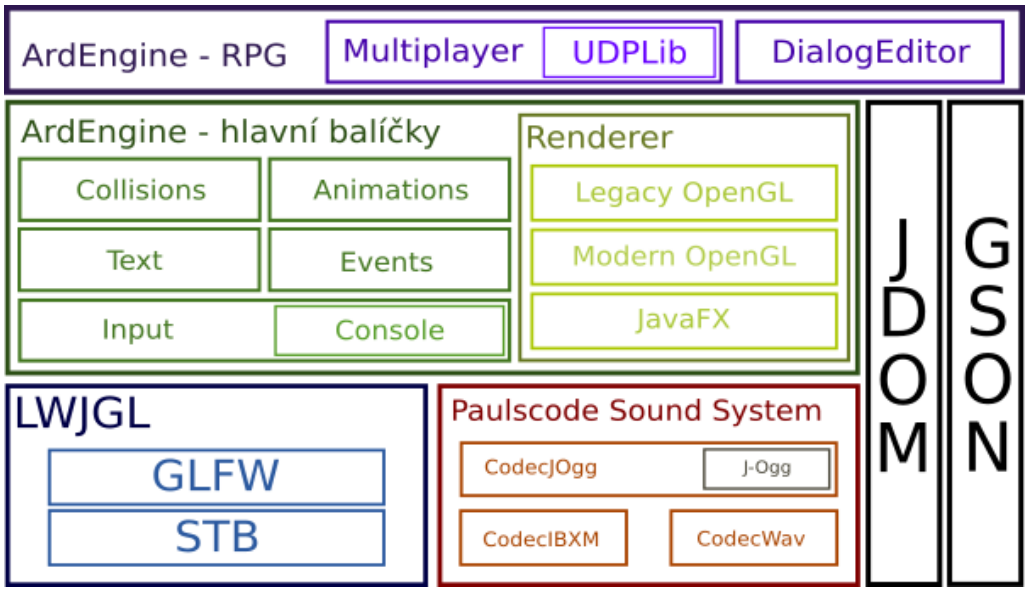
Tvůrce dialogu může určit, že některé odpovědi budou dostupné pouze tehdy, pokud jsou splněny určité podmínky. Knihovna poskytuje pouze základní podmínky pro porovnávání čísel a shodu řetězců, nicméně umožňuje tvůrci hry načíst jeho vlastní naprogramované podmínky. Obdobně knihovna umožňuje také volání vlastních funkcí pro práci s proměnnými. V následujícím Zdroj. kódu 8 je předvedeno, jak se pracuje s připraveným dialogem.

```
Dialog nactenyDialog = dialogs.loadDialog("hostinsky-start");
Response[] responses = loadedDialog.getAvailableResponsesArray();

System.out.println(loadedDialog.getEvent())

for(Response response : responses){
   System.out.println(">>>"+response.getText());
}
dialogs.selectResponse(responses[2]);

// VYPIS S KONZOLE
// Vítej cizinče, co by sis přál?
// >>>Dej mi nějaké to pivo!
// >>>Chci medovinu! //Pouze pokud je hodnota promenne Player.Money vyssi
nez 5
// >>>Neměl byste sklenici mléka?
```

Zdroj. kód 8 – Práce s dialogy

5 IMPLEMENTACE 2D HERNÍHO ENGINE

Jak již bylo zmíněno v předchozích kapitolách, engine využívá služby mnoha knihoven, jejichž přehled je zobrazen na Obr. 10, který také popisuje hierarchicky hlavní skupiny kódu. Skupiny s názvem „ArdEngine“ byly vytvořeny v rámci této práce a bude jim věnován zbytek této kapitoly. Přehled použitých knihoven je také uveden v Příloze C.



Obr. 10 – Rozvržení projektu

Skupina „ArdEngine – hlavní balíčky“ obsahuje veškerý kód potřebný pro samostatný 2D engine. Druhá skupina je nadstavba, která přidává funkce potřebné pro RPG hry s podporou pro hru více hráčů a nezasahuje do hlavních balíčků. Pokud by tedy v budoucnosti měla být knihovna použita pro odlišný žánr her, nebude mu vyšší vrstva překážet.

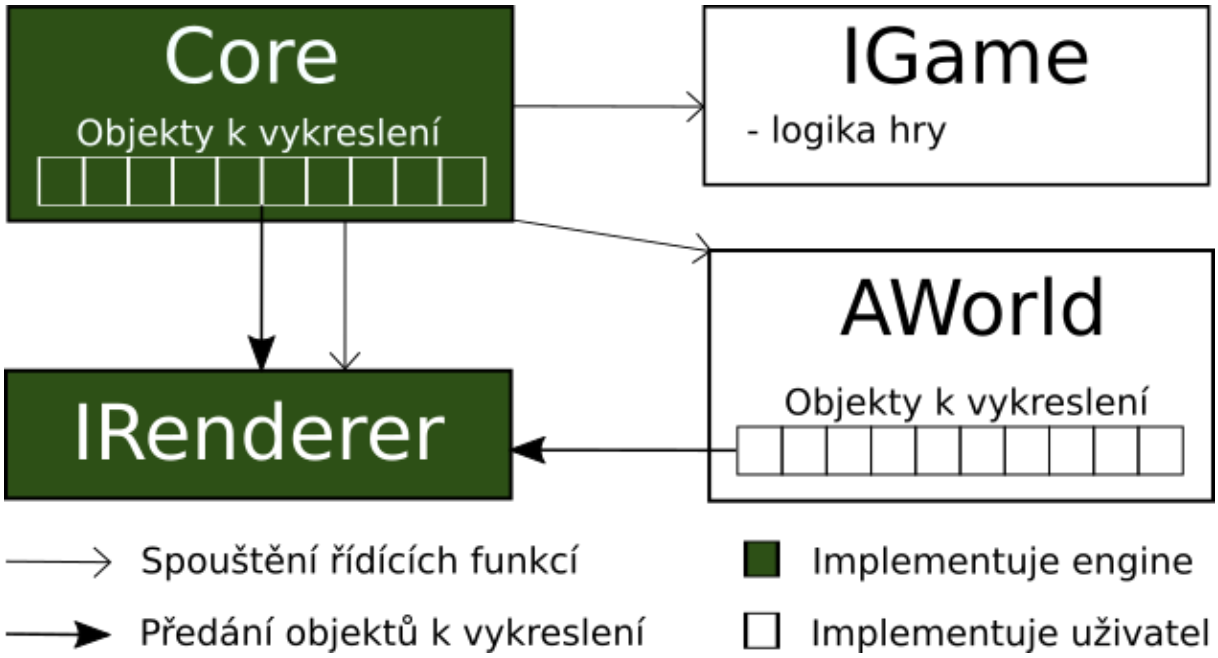
Knihovna *LWJGL 3 (LightWeight Java Game Library)* je vrstvou nad multiplatformní knihovnou *GLFW*[12], která je napsaná v jazyce C. *GLFW* poskytuje vývojáři funkce pro práci s oknem a ošetření vstupů z klávesnice, myši a dalších herních ovladačů.

LWJGL bylo vhodnější volbou, než alternativa v podobě knihovny *JOGL (Java Open Graphics Library).* Ta sice nabízí podobné funkce, ale pro poskytnutí okna využívá Java knihovnu *Swing*, která je v současné době nahrazována knihovnou *JavaFX*.

[12] Více na: *http://www.glfw.org*

5.1 Základní třídy a rozhraní

Kromě manažerů herních zdrojů představených v předchozí kapitole a herních prvků je vhodné, aby byl uživatel obeznámen s hlavními třídami a životním cyklem knihovny. Zjednodušený cyklus engine je uveden v Příloze A. Přehled tříd, kterými se budeme v této části věnovat, je zobrazen na Obr. 11.



Obr. 11 – Schéma interakcí mezi základními třídami engine

5.1.1 Třída *Core*

Core je hlavní třídou engine. Obsahuje statické funkce pro spuštění, běh a ukončení aplikace. Dále se v ní nachází aktuální souřadnice kamery, speciální herní prvky, informace o verzi a především reference na ostatní základní třídy.

5.1.2 Rozhraní *IRenderer*

IRenderer je rozhraní, jehož implementace mají na starosti vytvoření okna, načítání grafických zdrojů a vykreslování herních prvků. V současnosti engine podporuje tři implementace tohoto rozhraní: *ModernOpenGLRenderer*, *LegacyOpenGLRenderer* a *JavaFXRenderer*. Důvod, proč a jaké verze OpenGL jsou podporovány, byl již vysvětlen.

Vzhledem k tomu, že *ModernOpenGLRenderer* a *LegacyOpenGLRenderer* jsou závislé na stejné knihovně, byla vytvořena třída *JavaFXRenderer*, kterou se knihovna pokusí použít v případě, že inicializace *LWJGL* selže.

5.1.3 Rozhraní *IGame*

Implementace tohoto rozhraní by měla sloužit jako hlavní třída hry. Jsou zde definovány tři metody, které musí uživatel implementovat: *gameInit(), gameRun() a gameCleanUp()*. Metody jsou automaticky spouštěny ve třídě *Core*, aby uživatel mohl přidat vlastní úkoly při inicializaci, běhu hry a uvolnění zdrojů. Dále jsou v rozhraní vytvořeny defaultní metody pro snadnější práci s *Core*. Zdroj. kód 9 obsahuje příklad implementace *IGame*.

```
public class MyGame implements IGame{

	private Sprite pozadi;

	@Override
	public void gameInit() {
		// Upraveni vzhledu okna
		Core.renderer.setWindowTitle("Moje okno");
		Core.renderer.setWindowSize(800, 600);

		// Pridani obrazku na pozadi
		pozadi = new Sprite("cesta/k/obrazku.png")
		addDrawable(pozadi);
	}

	@Override
	public void gameRun() {}

	@Override
	public void gameCleanUp() {};

	public static void start(String[] args) {
		// Spusteni této hry s preferenci OpenGL implementace IRenderer
		Core.start(new MyGame(), Renderers.GL);
	}
}
```

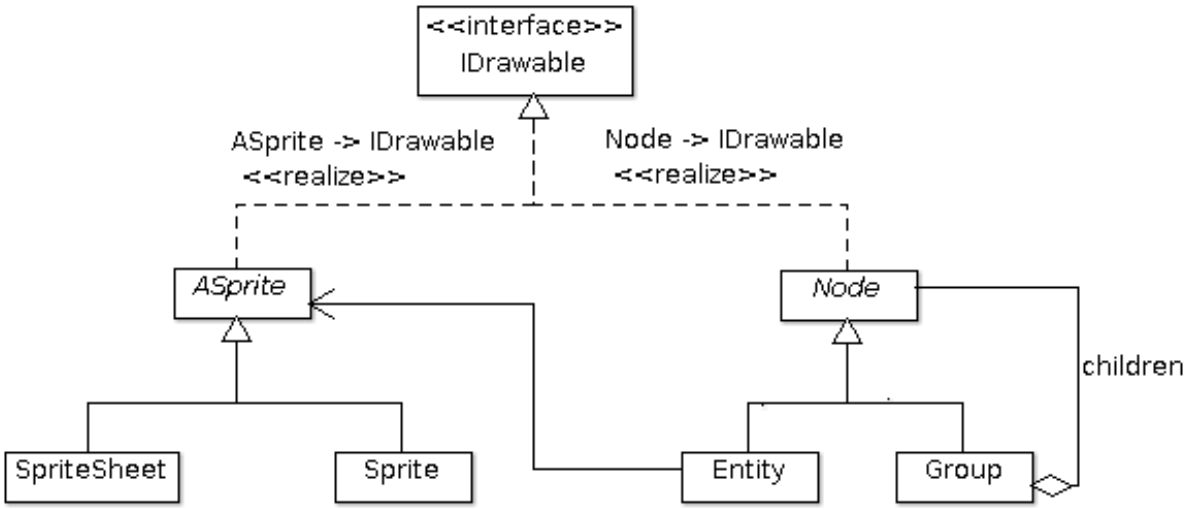
Zdroj. kód 9 – Vytvoření okna

5.1.4 Třída *AWorld*

Tato třída je v podstatě soubor herních prvků, které dohromady vytvářejí například herní úroveň, menu nebo načítací obrazovku. Podobně jako pro *IGame* musí uživatel implementovat metodu *run()* a volitelně *cleanUp()*, která je volána při změně na jiný svět. Změnu světa lze vyvolat pomocí metody *IGame.setWorld()*.

5.2 Základní herní prvky

Všechny herní prvky implementují rozhraní *IDrawable*, kterému je věnována příští část kapitoly. Toto rozhraní obsahuje hlavičky metod pro práci se souřadnicemi, měřítkem, barvou a úhlem natočení. Na Obr. 12 je zobrazen diagram tříd implementujících toto rozhraní.



Obr. 12 – Diagram tříd implementujících interface *IDrawable*

5.2.1 Třída *ASprite*

Potomci třídy *ASprite* reprezentují pouze načtený obrázek. Uživatel může obrázek obarvovat, zvětšovat i natáčet. *SpriteSheet* narozdíl od *Sprite* reprezentuje sadu obrázků, ze které je zobrazen vždy jen jeden na zvoleném indexu. Příklad vytvoření a použití instance třídy *SpriteSheet* je zobrazen ve Zdroj. kód 10.

```
//Rozseka obrazku.png na casti o velikosti 100x100px
SpriteSheet sada = new SpriteSheet("cesta/k/obrazku.png", 100, 100);
//Nastavi, ze bude vykreslovana ctvrta část obrazku
sada.setIndex(3);
```

Zdroj. kód 10 – Vytvoření a použití instance třídy SpriteSheet

5.2.2 Třída *Node*

Třída *Node* obohacuje herní prvky o další funkce, jako jsou kolize, události nebo interakce s myší. Třída *Entity* používá pro vykreslování instanci *Sprite* a umožňuje tak využívat zmíněné dodatečné funkce při práci s obrázky.

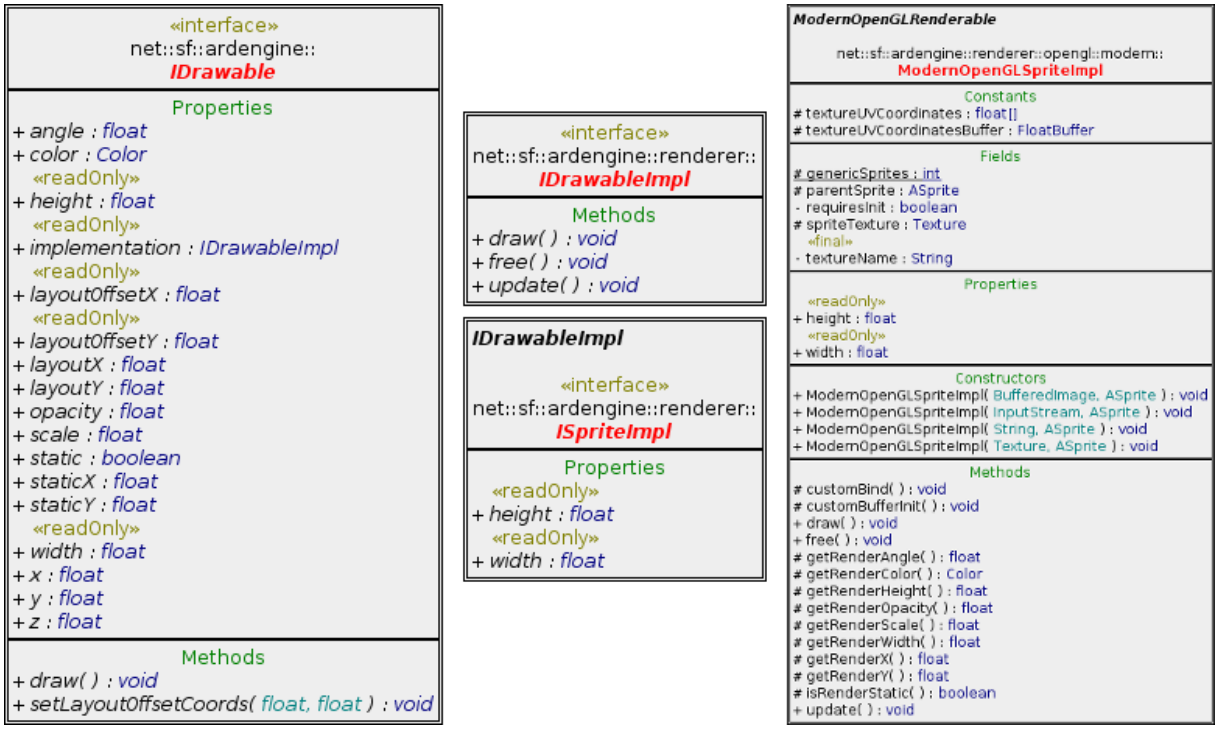
Účelem třídy *Group* je umožnit logické seskupení herních prvků. Například potomci *Entity Kolo*, *Motor* a *Karoserie* by byly uloženy ve třídě *Auto* dědící ze třídy *Group*. Pro posun auta by pak stačilo uživateli zavolat pouze metodu *Auto.setX()* a třída již dopočítá zbytek. Ve Zdroj. kód 11 je uveden další příklad použití.

```
// Cesta k obrazku, X, Y
Entity strecha = new Entity(new Sprite("strecha.png", 0, 0));
Entity zdi = new Entity(new Sprite("zdi.png"));
// Odsazeni, pokud je prvek soucasti skupiny
      zdi.setLayoutY(strecha.getWidth());
Group dum = new Group();
// Naplneni skupiny, k dispozici je Group.addChild()
dum.addChildren(new Entity[]{strecha, zdi});
```

Zdroj. kód 11 – Vytvoření skupiny

5.3 Vykreslení objektu pomocí *IDrawable*

Aby mohl být objekt vykreslen na obrazovce, musí implementovat rozhraní *IDrawable*, které obsahuje definici základních metod potřebných pro určení souřadnic a velikost výsledného obrázku (viz. Obr. 13). Souřadnice s prefixem „layout“ jsou aplikovány, pokud je objekt uložen v instanci již popsané třídy *Group*. Souřadnice s prefixem „static“ jsou určeny pro případ, kdy uživatel nastaví, aby se objekt pohyboval společně s kamerou.



Obr. 13 – UML diagram interface *IDrawable* a příklad jeho implementace

Nejdůležitější metodou v rozhraní *IDrawable* je *getImplementation()*. Při vytváření herního prvku je zvolený *IRenderer* požádán o objekt obsahující kód, který je použit pro vykreslení. Prostřednictvím tohoto objektu také prvek získává informace například o velikosti textury nebo naopak upozorňuje na změnu souřadnic, barvy nebo obsahu. Na Obr. 13 jsou zobrazeny diagramy tříd a rozhraní, která používá pro vykreslení *Sprite ModernOpenGLRenderer*.

5.4 Třída *Text*

Třída *Text* je dalším z potomků třídy *Node*. V přechozí kapitole bylo uvedeno, že pro OpenGL implementace *IRenderer* je z načteného fontu vytvořena textura s mapou znaků. Pro vykreslení textu by tedy stačilo pouze vedle sebe vykreslit znaky obsažené v zadaném textu. Tento přístup je ale neefektivní, protože procesor by musel při každém vykreslení GPU poskytovat stejné souřadnice pro každý znak, což by při vyšším množství textu zpomalilo celý proces. Z toho důvodu jsou nejprve předpočítány odhadované rozměry

textu. Poté je vytvořena textura této velikosti, do které je text vykreslen. Textura je následně použita při každém dalším vykreslení textu, jako by se jednalo o Sprite, dokud nedojde ke změně řetězce. Ve Zdroj. kódu 12 je předvedeno, jak s třídou *Text* pracuje uživatel.

```
// Text, pouzity font, zalamovani (px)
Text text = new Text(("Lorem ipsum", FontManager.getFont("mujFont"), 400);
text.setText("změna textu");
text.setColor(Color.DARKGREEN);
```

Zdroj. kód 12 – Vytvoření a práce s instancí třídy Text

5.5 Ošetření kolizí

Engine podporuje detekci kolizí mezi prvky, jsou-li potomky třídy *Node*. Takovému prvku lze přiřadit libovolné množství potomků třídy *ACollisionShape*. Knihovna již obsahuje předpřipravené třídy *CollisionPolygon* a *CollisionCircle*. Vzhledem k tomu, že *ACollisionShape* je veřejně dostupná, uživatel může snadno přidělat další tvary. Zdroj. kód 12 je příklad, jak nadefinovat nad herním prvkem kolize.

```
float width = entity.getWidth();
float height = entity.getHeight ();
// Oznaci prvek jako kolidovatelny
entity.setCollideable(true);
// Prida do kolizních tvaru prvku obdelnik jeho velikosti
entity.getCollisions().add(new CollisionRectangle(entity,
      [0, 0, width, 0, width, height, 0, height]));
```

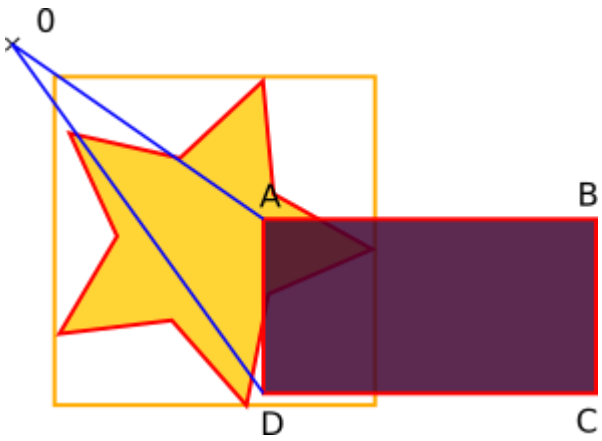
Zdroj. kód 13 – Přidání kolize

Při označení herního prvku kolidovatelným je pro prvek vytvořena instance třídy *PrivateAreaRect* (žlutá na Obr. 14). Tento čtverec obaluje herní prvek a je jako první testován, zda nekoliduje s instancemi *PrivateAreaRect* ostatních prvků. Pokud ano, pak jsou teprve otestovány tvary zadané uživatelem (červená na Obr. 14). Díky tomu je uspořen výkon, protože porovnání, zda nekolidují dva obdélníky, je výpočetně snazší, než výpočet kolizí mezi nekonvexními polygony.

Obr. 14 – Zobrazení kolizí v debugovacím režimu engine

Pro porovnávání kolizí mezi polygony je použita tzv. paprsková metoda. Základní princip spočívá v tom, že je zvolen bod ležící mimo oba polygony. Mezi tímto bodem a vrcholy jednoho z polygonů jsou vytvořeny úsečky (paprsky). Pro každou úsečku je spočítán počet průsečíků s druhým polygonem. Má-li některá z úseček lichý počet, znamená to, že koncový

bod této úsečky a tedy i vrchol polygonu leží v prostoru druhého. Příklad této metody je zobrazen na Obr. 15.



Obr. 15 - Ukázka paprskové metody

5.6 Ošetření myši a klávesnice

Vzhledem k tomu, že každá implementace *IRenderer* má vlastní způsob zpracování uživatelského vstupu, vznikla třída *InputManager* jako vrstva pro uživatele, která mu umožňuje snadno zachytávat a ověřovat stav klávesnice a myši. Jako univerzální textové pole byla vytvořena třída *TextBox*, kterou uživatel může využít pro zadávání i výpis textu.

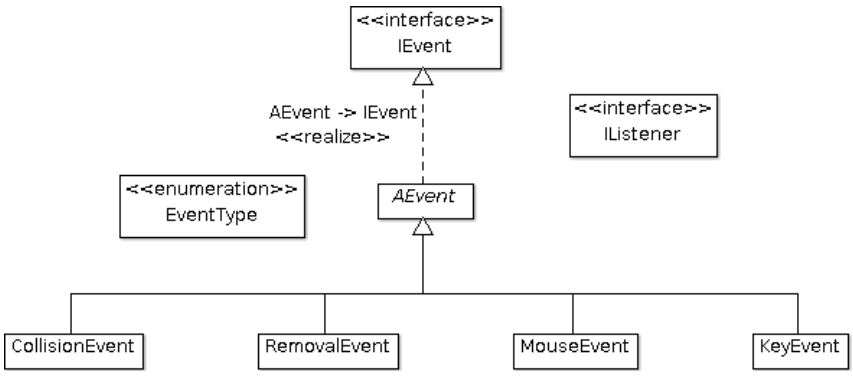
Implementace *IRenderer* pomocí metod *insertMouseInput(), insertKeyBoardInput()* a *insertCharInput()* posílá manažeru hlášení o aktuálním stavu. Manažer při každé aktualizaci tyto hlášení zpracuje a v případě potřeby vyšle události, které jsou popsány v příští části. Za zmínku stojí třída *MouseTracker*, pomocí které *InputManager* zachytává události spojené s myší, jako je tahání nebo kliknutí. Tato třída je potomkem třídy *Node* a využívá tak kolizí zmíněných v předchozí kapitole.

5.7 Události

Vzhledem k velkému množství stavů a situací, které by uživatel vždy nemusel chtít ošetřit, vznikl systém událostí využívající návrhový vzor Pozorovatel. Třídy *InputManager* a *Node* obsahují seznam objektů implementujících rozhraní *IListener*. Toto rozhraní definuje hlavičky dvou metod, *geType()* vrací typ události, na kterou posluchač čeká, a metoda process() je zavolána, pokud taková událost skutečně nastala.

Samotnou událost definuje rozhraní IEvent (Obr. 16). Opět se jedná o jednoduché rozhraní, které obsahuje metodu vracející typ události a metody pro označení události za zpracovanou, pro případ, že by chtěl uživatel naznačit dalším posluchačům v pořadí, že se nemají událostí

zabývat. Pro případ, že by chtěl programátor pracovat s vlastními typy událostí, které nejsou uvedeny ve výčtu *EventType*, může využít typ *EventType.CUSTOM*. Ve Zdroj. kódu 14 je předveden jednoduchý kód pro vytvoření tlačítka za pomoci událostí.



Obr. 16 – Předpřipravené události

```
Entity tlacitko = new Entity(new Sprite("tlacitko.png"), 0, 0);
float width = tlacitko.getWidth();
float height = tlacitko.getHeight ();

// Udalosti mysi jsou automaticky vyhodnocovany pro kolidovatelne prvky
tlacitko.setCollideable(true);
tlacitko.getCollisions().add(new CollisionRectangle(tlacitko,
      [0, 0, width, 0, width, height, 0, height]));
tlacitko.registerListener(new IListener() {

      @Override
      public EventType getType() {
            return EventType.MOUSE_CLICKED;
      }

      @Override
      public void process(IEvent event) {
            Systém.out.println("Třída pro tlačítka bude pravděpodobně brzy
přidána do knihovny, aby ušetřila uživatelům práci.");
      }

});
```

Zdroj. kód 14 – Příklad vytvoření tlačítka

5.8 Konzole

Konzole je speciální potomek třídy *TextBox*, kterou lze zobrazit stisknutím klávesy *F3*. Pomocí příkazů v konzoli může programátor snadno zjišťovat informace o běžícím programu, aniž by musel vytvářet vlastní mechanismy. Například příkaz *stats.renderer()* vypíše název třídy použité implementace *IRenderer* a příkaz *debug.enable()* spustí vykreslování kolizí u herních prvků. Konzole také obsahuje integrovanou nápovědu a možnost přidávat vlastní moduly příkazů. V Příloze B je zobrazeno testovací okno knihovny se zobrazenou konzolí.

6 TVORBA HRY S PODPOROU PRO MULTIPLAYER

Nad funkcemi engine, které byly představeny v minulé kapitole, byl vytvořen balíček *rpg*. Hlavními funkcemi této vrstvy jsou podpora pro dialogy, která již byla představena, a umožnění hry více hráčů. Účelem této kapitoly je představit tuto vrstvu při popisu tvorby ukázkové hry.

6.1 Knihovna UDPLib[13]

UDPLib je knihovna pro práci se síťovým protokolem UDP/IP. Tento protokol je alternativou k TCP/IP, na rozdíl od kterého nechává implementaci ochrany proti chybám na uživateli a urychluje tak celou komunikaci. Tento protokol je vhodný zejména pro aplikace, kterým nevadí krátkodobá ztráta dat, zejména pokud mají omezenou trvanlivost. UDP je tedy vhodné pro video hovory nebo akční hry. Další výhodou UDP je možnost skupinové komunikace. Jedná se o rozeslání jednoho datagramu více cílům, což ušetří programátorovi práci s obsluhováním tzv. soketů. *„Soket lze považovat jednoduše za koncový bod propojení mezi dvěma procesy“* [13].

Knihovna byla vytvářena s vědomím, že bude součástí tohoto projektu. V rámci této práce bylo provedeno několik nezbytných úprav, jako je čištění kódu a dodatečné testování. V současnosti knihovna nabízí funkce pro zjednodušení skupinové UDP komunikace, přenos menších binárních i textových souborů a šifrování pomocí algoritmu AES.

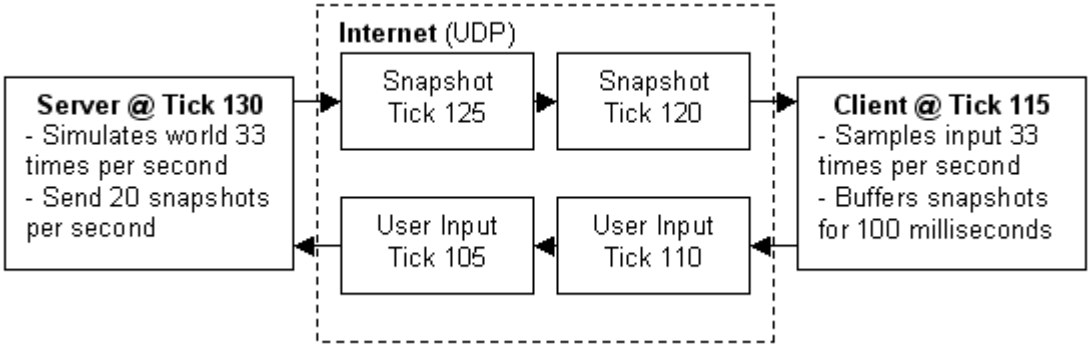
Knihovna zjednodušuje UDP komunikaci díky třídám, které reprezentují tzv. *multicast* skupinu. Jeden uživatel se ujme role hostitele a periodicky rozesílá do skupiny datagramy s informacemi o stavu členů skupiny. Členové tedy o sobě navzájem vědí a mohou si vyměňovat zprávy. Uživatel pouze vytvoří instanci třídy *ServerGroupNetwork* nebo *ClientGroupnNetwork* a zavolá funkci *start()*. Třída spustí vlastní vlákno, pomocí kterého udržuje připojení do skupiny a zpracovává zprávy knihovny. Zprávy pro uživatele jsou ukládány do datové struktury, ze které si je může kdykoliv vyzvednout.

6.2 Komunikace při hře více hráčů

Kromě knihovny UDPLib poskytuje engine uživateli třídy *AServerCore*, *AClientCore* a rozhraní *INetwork* zjednodušující komunikaci při hře více hráčů. Díky rozhraním si uživatel může zvolit, zda pro komunikaci využije nabídnutou knihovnu nebo doimplementuje

[13] Více na: https://github.com/Mytrin/UDPLib

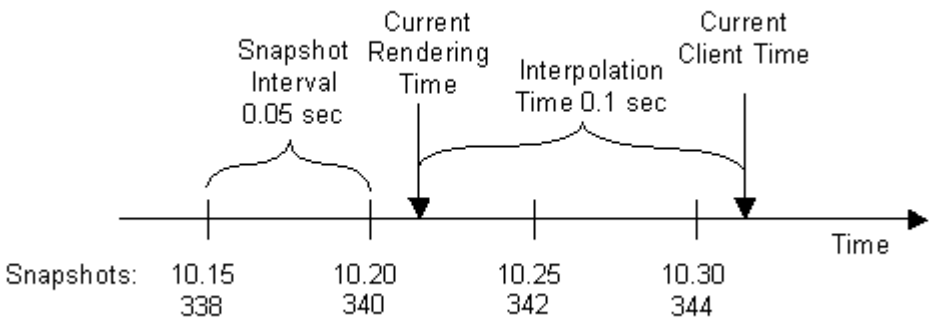
potřebné funkce za použití vlastní. Zmíněné třídy abstrahují komunikaci herního serveru a klienta, která je znázorněna na Obr. 17.



Obr. 17 – Přenos snapshotů mezi klientem a serverem [14]

Herní server simuluje průběh hry s předstihem přibližně 100ms. Při každé aktualizaci herního stavu odešle tzv. *snapshot*. Jedná se o současný stav herních prvků, které jsou sdíleny po síti. Například za stav letícího šípu lze považovat jeho souřadnice, rychlost a směr. Pro zmenšení objemu dat jsou většinou odesílány pouze informace o změně stavu oproti předchozímu *snapshotu*. Kompletní informace jsou rozesílány pouze v případě, že se do hry připojil nový hráč, který nezná výchozí stav [14].

Klient si přijaté stavy ukládá pro využití v budoucnu a zobrazuje stav 100ms starých. Pokud některý *snapshot* neobdržel, zkouší jej přeskočit a lineárně interpolovat k dalšímu ve frontě. Chybí-li ve frontě příliš mnoho *snapshotů* v řadě, nedokáže pro dotyčný prvek dále věrohodně simulovat. Pokud se jedná o již zmíněný letící šíp, pravděpodobně by byl zastaven ve vzduchu, nebo, počítal-li s touto situací programátor, by letěl stále stejným směrem a ignoroval fakt, že trefil cíl. Na Obr. 18 je zobrazena časová osa klienta a jeho chování v případě, že ztratil jeden *snapshot* [14].



Obr. 18 – Zpracování snapshotů na straně klienta [14]

Popsané chování je již ve třídách *AServerCore* a *AClientCore* implementováno. Stavy jsou odesílány ve formátu JSON pomocí třídy *JsonMessage.* Aby mohl být herní prvek sdílen po síti, musí jeho třída implementovat rozhraní *INetworkedNode*, ve kterém jsou metody pro vytvoření a interpolace mezi *snapshoty*. Tyto metody volá engine automaticky.

Třída *JsonMessage* obsahuje getter *getType*(), která specifikuje, o jaký druh zprávy se jedná. Uživatel si může vytvářet vlastní typy zpráv. V *AServerCore* a *AClientCore* lze zaregistrovat v mapě s typy zpráv objekt, který bude mít na starosti zpracování zvoleného typu *JsonMessage*.

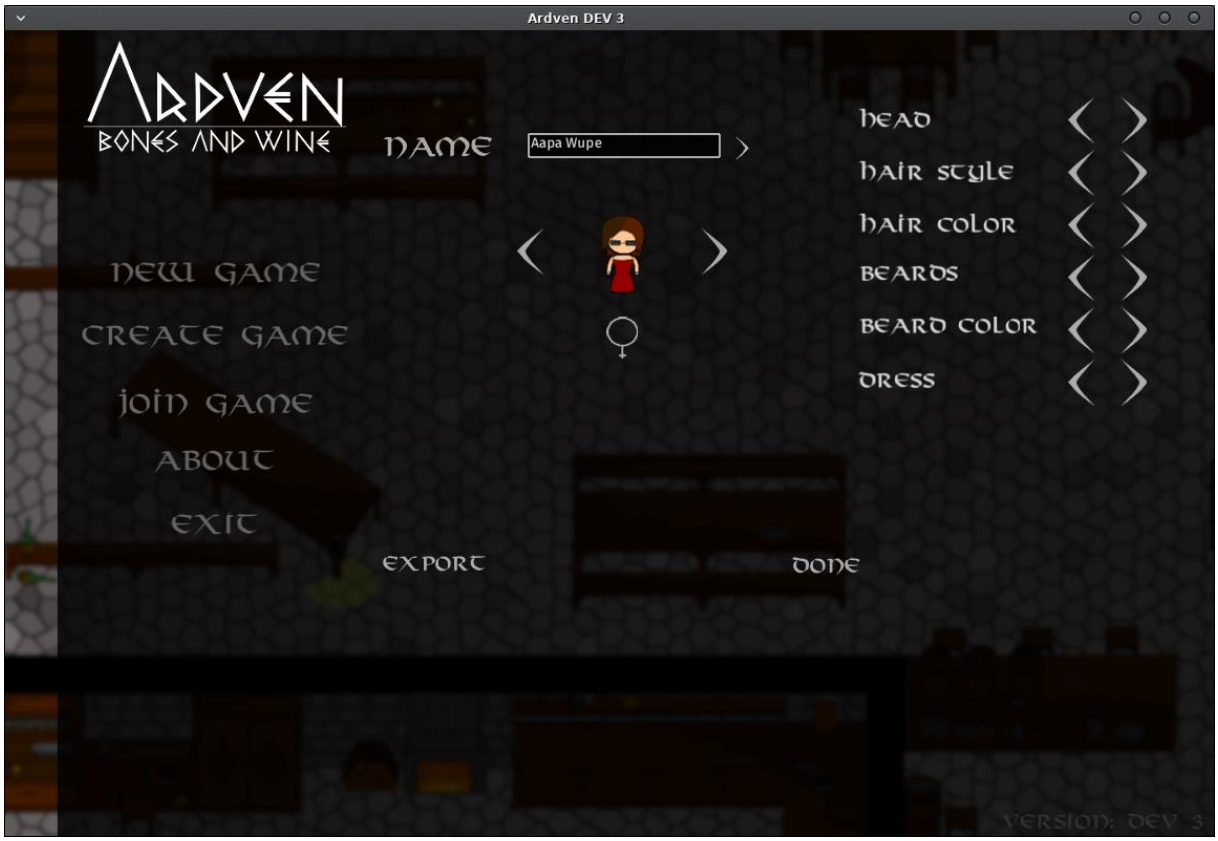
Ukázková hra využívá popsané třídy. Režim více hráčů je podporován, pokud hráč hraje na lokální síti, do které je připojen kabelem, nebo po internetu skrze VLAN, který poskytuje například program Hamachi[14]. Při připojení do LAN pomocí technologie Wifi hrozí v zahlušeném prostředí příliš vysoké riziko ztráty důležitých herních datagramů. Do budoucna je plánováno do knihovny UDPLib implementovat funkci, která by zaručovala doručení kritických zpráv.

6.3 Herní menu

Po spuštění hry se uživateli zobrazí nahrávací obrazovka engine. Po načtení se objeví herní menu (Obr. 19). Menu představuje třída *MenuWorld*, která je potomkem již zmíněné třídy *AWorld*. Po kliknutí na volbu v menu je animací zobrazen panel s příslušnou funkcí. Třídy panelů jsou potomky dříve popsané *Group* a komunikují společně pomocí událostí.

V menu jsou bohatě použity animace. Položka v menu se po najetí myší postupně rozsvěcí. Pokud myš prostor tlačítka opustí, položka se podobnou animací vrátí do původního stavu. Panel, který má být zobrazen, postupně mění průhlednost, aby byla změna obsahu okna pro uživatele příjemnější. V pozadí se náhodně pohybuje rozmazaný obrázek první herní úrovně. Všechny tyto efekty zajišťuje balíček engine *animations* nabízející základní animace pro přechod průhlednosti, barvy nebo měřítka i časované spouštěče.

[14] Více na: https://www.vpn.net



Obr. 19 – Herní menu

Funkce položek v menu:

- *New Game* spustí vytváření postavy a následně novou hru pro jednoho hráče.
- *Create Game* zobrazí formulář pro spuštění hry více hráčů v roli hostitele.
- *Join Game* zobrazí formulář pro připojení do běžící hry.
- *About* zobrazí autorovo poděkování, seznam použitých zdrojů a použité knihovny.
- *Exit* ukončí hru.

Na Obr. 20 jsou zobrazeny panely s formuláři, které hráč musí vyplnit pro připojení, či založení hry více hráčů. Pokud nejsou vyplněny nepovinná políčka, knihovna UDPLib použije implicitní nastavení, jenž lze změnit pomocí konfiguračního souboru. V případě, že hráč vytváří hru, přijímá roli serveru. Pokud nespecifikoval IP adresu serveru, budou použita všechna dostupná síťová rozhraní s vyjímkou lokální smyčky.

Obr. 20 – Panely menu pro připojení(vlevo) a založení hry

6.4 Hráčova postava

Hráčova postava se skládá z více částí, které jsou spojeny pomocí třídy *Group*. Tyto části jsou tělo, hlava, vousy a vlasy. Jsou uloženy v různých grafických souborech, které hra načte pomocí třídy *SpriteSheet*. Obr. 21 ilustruje, jak je z popsaných částí složena postava. Cesty k obrázkům, ze kterých je hráčova potava složena, jsou uloženy do souboru ve formátu JSON. Všechny obrázky ve hře byly vytvořeny autorem pomocí programů Inkscape a GIMP.

Obr. 21 – Tvorba postavy

6.5 Herní svět

Při startu hry se hráči zobrazí načítací obrazovka (Obr. 21). Pokud je hráč klientem, přijímá od serveru archiv obsahující data načítané hry. V případě, že hráč obstarává roli serveru, nebo hraje v režimu jednoho hráče, jsou všechna potřebná data k dispozici. Po přijetí archivu hráč obdrží od serveru signál a začne z archivu načítat hru.

Obr. 22 – Pozadí načítací obrazovky

Složka s daty hry obsahuje tři podsložky:

- Složka *characters* obsahuje uložené soubory s aktuálním vzhledem a vlastnostmi postav hráčů.
- Složka *dialogs* je projekt knihovny DialogEditor, který obsahuje veškeré dialogy hry a uložené proměnné. Nehrozí tedy, že by nově připojený hráč aktivoval dialog, kteří již ostatní splnili.
- Složka *levels* obsahuje soubory ve formátu JSON, ve kterých je nakonfigurován vzhled a chování herní úrovně (Zdroj. kód 15).

Pokud se do hry připojí nový hráč, server musí tyto složky a soubory v nich obsažené aktualizovat, zkomprimovat a odeslat novému klientovi. Díky tomu klient zná přesný stav všech objektů, které by se mohly měnit. Neměnné prvky, jako jsou například obrázky, nejsou v těchto složkách obsaženy. Na podobném principu by šlo založit i ukládání hry.

```
{
	"background": "res/stories/bones_and_wine/inn/level.png",
	"spawn": [570, 225],
	"collisions": [
		{
		"type": "polygon",
		"coords": [0, 0, 70, 0, 70, 1235, 0, 1235]
		},
	...
	"objects" : [
		...
		{
		"type": "block",
		"sprite": "res/sprites/furniture/chair_front.png",
		"x": 815,
		"y": 105,
		"title": "Klasická židle, která vrže jako houslista v 5
ráno..."
		},
		...
		{
		"type": "block",
		"sprite":
"res/stories/bones_and_wine/inn/person_sprites/alex.png",
		"x": 966,
		"y": 840,
		"dialog": "inn:innkeeper-start"
		},
	...
```

Zdroj. kód 15 – Ukázka se souboru herní úrovně

Vlastnost *background* nastavuje cestu k obrázku, který bude použit jako pozadí úrovně (Obr. 23). *Spawn* udává souřadnice prostoru, kde se objeví hráčova postava. *Collisions* obsahuje kolizní tvary, které jsou použity jako zdi.

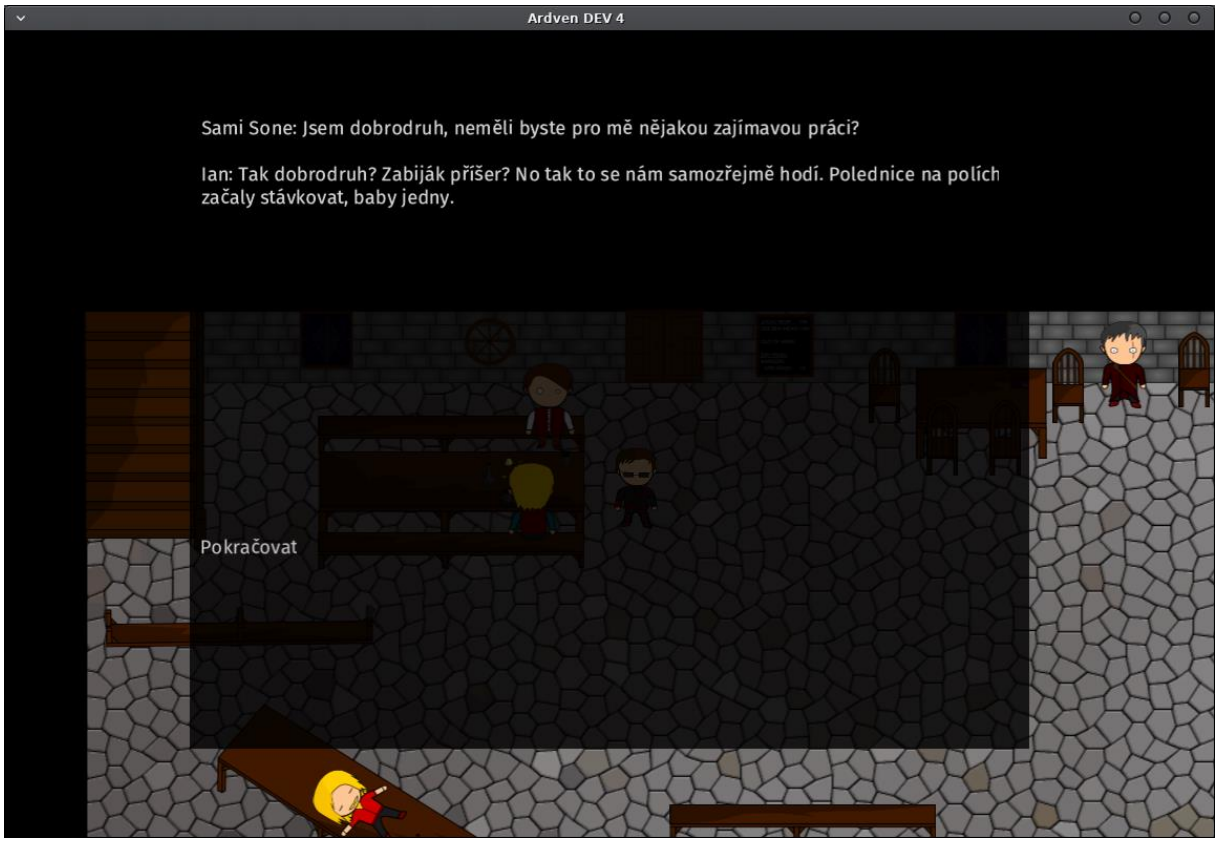
Obr. 23 – Naplnění herní úrovně

Na pozadí nastavené v souboru úrovně jsou tedy naskládány další herní prvky, které jsou nadefinovány ve vlastnosti *objects*. Tyto prvky se dělí na více se dělí na více typů.

Typ *decoration* je obrázek, který se nechová jako překážka a ani nemusí být testován na kolize, pokud není nastaven atribut *title*. Po kliknutí na herní prvek s atributem *title* se ve hře objeví text s obsahem atributu. Účel tohoto prvku je pouze sloužit jako dekorace, například talíř na stole, či obraz na zdi. Cestu ke zdrojovému obrázku udává atribut *sprite*.

Typ *block* při kolizi s postavou hráče se chová jako překážka. Také umožňuje použít místo *title* atribut *dialog*, který obsahuje cestu k dialogu (Obr. 25), který bude po kliknutí myší na dotyčný prvek spuštěn. Příklad obrázku použitého jako tento typ je představen na Obr. 24. Hotové úrovně jsou uvedeny v Přílohách D a E.

Obr. 24 – Příklad obrázku použitého jako herní prvek typu block



Obr. 25 – Panel s dialogy

7 ZÁVĚR

V rámci práce byl vytvořen funkční 2D herní engine. Pro projekt vznikly a jsou dále udržovány knihovny pro tvorbu herních dialogů a komunikaci v UDP skupině. Tyto knihovny jsou použitelné i v rámci jiného projektu. Nakonec byla vytvořena ukázková hra, která příkladně využívá zmíněné knihovny.

V současnosti se engine skládá ze 172 tříd, což je přibližně 15 000 řádků, nepočítaje ukázkovou hru a knihovny. UDPLib a DialogEditor jsou stvořeny z dalších 72 tříd obsahujících 7 000 řádků. Samotná hra zabírá přibližně 65 tříd s 4 600 řádky. Velikost projektu samozřejmě nevypovídá o jeho kvalitě, ale poukazuje na rozsáhlost problematiky, kterou se zaobírá.

Porovnám-li projekt s mou maturitní prací, je téměř dvakrát větší. Z tehdejšího projektu jsem přitom převzal přibližně jen 15 tříd, které byly zásadně upraveny a pročištěny. Engine je rozsáhlejší, protože původní koncept byl obohacen o nové funkce, jako jsou například události a vrstva pro hru více hráčů. Také řeší načítání obrázků a fontů tam, kde dřívější projekt využíval další knihovnu třetí strany.

V další části vývoje bych se zaměřil především na čištění kódu. Ve většině tříd engine, které využívá přímo uživatel, jsem s čistotou poměrně spokojen. Chtěl bych ale projít a optimalizovat, nejlépe s knihou Čistý kód v ruce, balíčky pro práci s OpenGL, kde jsem si vyhlédl několik rizikových tříd. Po úklidu bych lehce mohl do balíčků přidělat další funkce, například efekt rozmazání, záře nebo vyhlazování.

Na ukázkové hře se mi podařilo ověřit, že je engine použitelný a univerzální. RPG hry totiž patří do kategorie složitěji programovatelných her, jelikož bývají bohaté na funkce. Samotnou hru lze spíše považovat za funkční koncept. V současnosti by s minimálními úpravami šla okamžitě použít jako adventura. Kód je ale podobně jako u knihovny potřeba pročistit. Jednalo by se hlavně o přemístění některých částí kódu a úpravu jeho dostupnosti, která se při bouřlivém vývoji často měnila.

Při realizaci této práce jsem si vyzkoušel role programátora, návrháře herních úrovní, grafika, testera i spisovatele. S výsledky jsem spokojen.

8 POUŽITÁ LITERATURA

[1] MORRISON, Michael. *Naučte se programovat počítačovou hru za 24 hodin*. Brno: Computer Press, 2004. ISBN 80-251-0371-4.

[2] Get Started with UE4. *UnrealEngine.com* [online]. Cary: Epic Games, 2004 [cit. 2017-03-29]. Dostupné z: https://docs.unrealengine.com/latest/INT/GettingStarted/index.html

[3] Unity - Manual: Learning the interface. *Unity3d.com* [online]. San Francisco: Unity Technologies, 2017 [cit. 2017-04-05]. Dostupné z: https://docs.unity3d.com/Manual/LearningtheInterface.html

[4] *GemRB wiki* [online]. GemRB, 2017 [cit. 2017-03-27]. Dostupné z: http://www.gemrb.org

[5] *RPG Maker* [online]. Tokyo: Enterbrain, 2017 [cit. 2017-04-05]. Dostupné z: http://www.rpgmakerweb.com

[6] SEDDON, Chris. *OpenGL game development*. Plano, Tex.: Wordware Pub., 2004. ISBN 978-155-6229-893.

[7] *OpenGL: The Industry's Foundation for High Performance Graphics* [online]. Beaverton (Oregon): Khronos Group, 2017 [cit. 2017-04-05]. Dostupné z: https://www.opengl.org

[8] *Microsoft MSDN* [online]. Redmond: Microsoft, 2017 [cit. 2017-03-28]. Dostupné z: https://msdn.microsoft.com

[9] HAWKINS, Kevin a Dave ASTLE. *OpenGL game programming*. Roseville, CA: Prima Tech, 2001. ISBN 9780761533306.

[10] DirectX12. *DirectX Developer Blog* [online]. Redmond: Microsoft, 2014 [cit. 2017-04-05]. Dostupné z: https://blogs.msdn.microsoft.com/directx/2014/03/20/directx-12

[11] HAJDARBERGOVIC, Nermin. Introduction to Vulkan API. In: *Toptal* [online]. Sillicon Valley: Toptal, 2015 [cit. 2017-04-05]. Dostupné z: https://www.toptal.com/api-developers/a-brief-overview-of-vulkan-api

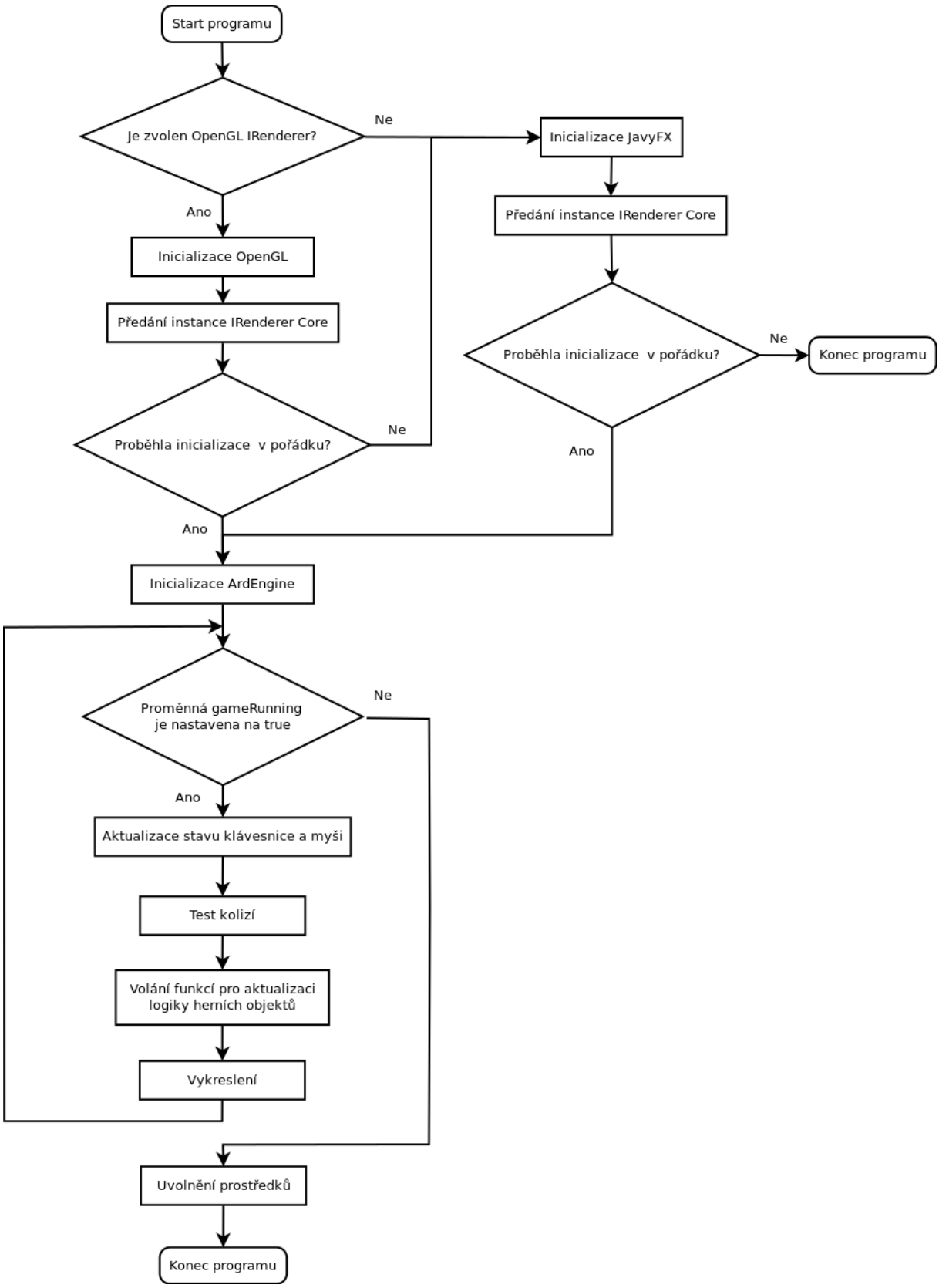
[12] *Vulkan Tutorial: Introduction* [online]. 2017 [cit. 2017-04-05]. Dostupné z: https://vulkan-tutorial.com

[13] KIZSKA, Bogdan. *1001 tipů a triků pro jazyk Java.* Brno: Computer Press, 2009. ISBN 978-80-251-2467-3.

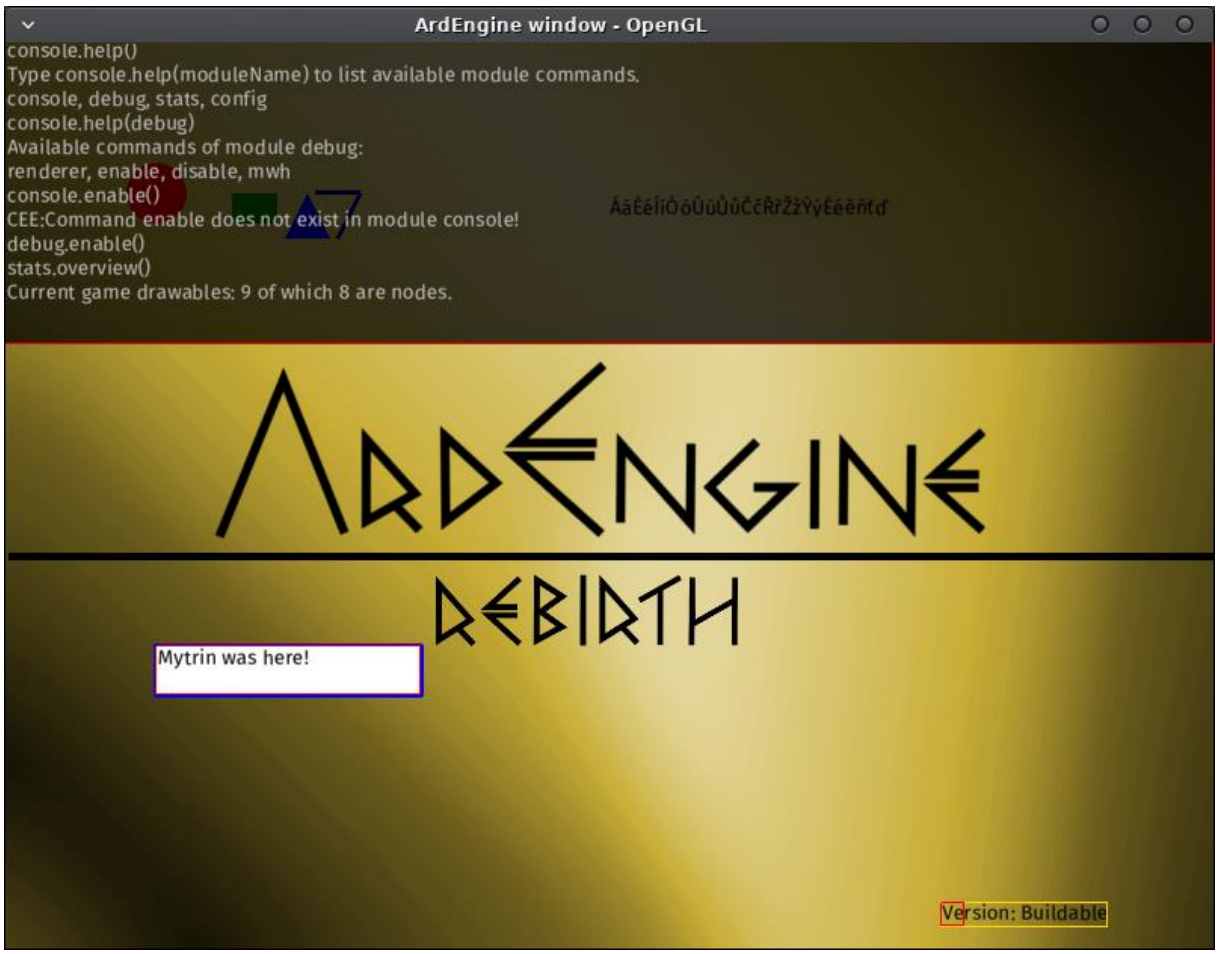
[14] Source Multiplayer Networking. *Valve Developer Community* [online]. Bellevue: Valve Corporation, 2017 [cit. 2017-04-23]. Dostupné z: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

9 PŘÍLOHY

Příloha A – Zjednodušený diagram běhu ArdEngine



Příloha B – Testovací okno ArdEngine



Příloha C – Použité knihovny třetích stran

Hlavní knihovna	Odkaz na webové stránky	Použité knihovny třetích stran
LWJGL	http://www.lwjgl.org	GLFW
		STB
Paulscode Sound System	http://www.paulscode.com	CodecJOgg
		J-Ogg
		CodecIBXM
		CodecWav
JDOM	http://www.jdom.org	-
GSON	http://github.com/google/gson	-

Příloha D – Vzhled první úrovně ve hře

Příloha E – Vzhled druhé úrovně ve hře