Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

3D Software pro návrh RPG her

Bakalářská práce

Autor: Viktor Horáček
Studijní obor: Aplikovaná informatika

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 24.4.2019 Viktor Horáček

Poděkování:

Děkuji vedoucímu práce prof. RNDr. PhDr. Antonínu Slabému, CSc. za cenné rady, věcné připomínky, vstřícnost při konzultacích a čas který mě věnoval v průběhu zpracování bakalářské práce. Děkuji také Bc. Janu Rajsovi za pomoc při gramatické kontrole práce. Dále chci poděkovat rodině a přátelům, kteří mi byli velkou oporou a bez jejichž podpory a trpělivosti by tato práce nevznikla.

Anotace

Bakalářská práce je zaměřena na analýzu a vývoj aplikace, sloužící k vytváření statické trojrozměrné scény a dějové linie s následnou možností hraní vytvořeného návrhu. Tvorba scény je možná pomocí vytvořených nástrojů pro editaci terénu, post procesorových efektů a podnebí v uživatelsky přívětivém rozhraní. Příběh je vytvořen a vyprávěn formou textových oken, které se budou zobrazovat při interakci s postavami, řízených počítačem. Hlavní podmínkou vyvíjené aplikace je zachování snadné rozšiřitelnosti o další editační nástroje pomocí stavů aplikace, které JMonkeyEngine nabízí.

V první části práce je seznámení s JMonkeyEnginem, ve kterém je aplikace vyvíjena a herním stylem RPG her, pro jejichž navrhování bude aplikace určena. Druhá část práce popisuje některé součásti JMonkeyEnginu s ukázkami jejich implementace, jež jsou využity při vývoji aplikace, protože popsání všech součástí JMonkeyEnginu je nad rozsah této práce. V příloze na CD se nachází okomentovaný zdrojový kód aplikace a sestavená aplikace s připravenou ukázkovou scénou a dějovou linií, která je vytvořena pomocí vyvinuté aplikace.

Klíčová slova: Java, JMonkeyEngine, OpenGL, LWJGL, Blender, RPG, Počítačové hry

Annotation

Title: Software for story designing of 3D RPG games

The bachelor thesis is focused on the analysis and development of an application, which is used to create a static three-dimensional scene and a storyline with the subsequent possibility of playing the created design. Scene creation is possible with the tools created for terrain editing, post processor effects and climate in a user-friendly interface. The story is created and narrated in the form of text windows that appear when interacting with computer-controlled characters. The main prerequisite for the application being developed is to maintain the ability to easily extend with other editing tools using „Application state" that JMonkeyEngine offers.

The first part of the thesis deals with the familiarization of JMonkeyEngine in which the application is developed and the game style of RPG games for which the application will be designed. The second part describes some components of JMonkeyEngine with examples of their implementation, which are used in the development of the application, because the description of all the components of JMonkeyEngine is beyond the scope of this work. The appendix on the CD contains a source code of the application with notes and a built-in application with a prepared sample scene and a storyline that is created using the developed application.

Key words: Java, JMonkeyEngine, OpenGL, LWJGL, Blender, RPG, Computer games

Obsah

1 Úvod ... 1

2 Cíl práce ... 2

3 Metodika zpracování ... 3

4 JMonkeyEngine 3 ... 4

4.1 3D prostor ... 5

4.2 SDK ... 9

4.3 Blender ... 10

4.3.1 UV Mapování Textur ... 10

4.4 Nifty GUI ... 11

4.5 Lightweight Java Game Library ... 13

4.5.1 OpenGL ... 14

4.5.2 OpenAL ... 14

4.6 Java ... 15

4.6.1 Výkon a výhody ... 15

5 Počítačové hry ... 17

5.1 Hra na hrdiny ... 18

5.1.1 Klasické RPG ... 18

5.1.2 Počítačové RPG ... 19

6 Vývoj aplikace v JME3 ... 20

6.1 Třída SimpleApplication ... 20

6.2 Stavy aplikace ... 24

6.3 Herní assety ... 27

6.4 Aktualizační smyčka ... 29

6.5 Kamera ... 29

6.6 Uživatelské vstupy ... 30

6.7 Materiály 32
6.8 Animace a kostra modelu 36
6.9 Ray Casting 40
6.10 Fyzika a kolize 41
6.11 Generování terénu 45
7 Shrnutí výsledků 48
8 Závěry a doporučení 49
9 Seznam použité literatury 50
10 Přílohy 52

Seznam obrázků

Obr. 1 Souřadnicový systém v JME3 5
Obr. 2 Rotace pomocí Quaternionu 7
Obr. 3 Graf scény 8
Obr. 4 Transformace modelu na bitmapu 11
Obr. 5 Inicializace Nifty v JME3 12
Obr. 6 Ovladač obrazovky 12
Obr. 7 Vytvoření vlastního textu na obrazovce 13
Obr. 8 Porovnání výkonu C# a Java 16
Obr. 9 Hexový a čtverečkovaný herní plán 19
Obr. 10 Inicializace třídy Main 20
Obr. 11 Změna velikosti okna 22
Obr. 12 Statistika vykreslované scény 22
Obr. 13 Inicializace stavů 23
Obr. 14 Nastavení spuštěné aplikace 23
Obr. 15 Vlastní výchozí nastavení 24
Obr. 16 Struktura AppState třídy 25
Obr. 17 Inicializace AppState 26
Obr. 18 Načítání assetů 28
Obr. 19 Struktura vlastní Savable třídy 28
Obr. 20 Využití aktualizační smyčky 29
Obr. 21 Informace o transformací kamery 30
Obr. 22 Inicializace třídy ChaseCamera 30
Obr. 23 Implementace ActionListener a AnalogListener 32
Obr. 24 Porovnání materiálů v JMonkeyEnginu 33
Obr. 25 Material s a bez normálové mapy 33
Obr. 26 Difúzní textura, normálová a světelná mapa 34
Obr. 27 SkyBox textura 34
Obr. 28 Alpha mapa terénu 35
Obr. 29 Programová úprava materiálu 35
Obr. 30 Vlastní tvorba materiálu pomocí SDK 36

Obr. 31 Animace klíčové snímky....37
Obr. 32 Animace ve Scene Exploreru....38
Obr. 33 Implementace animace....39
Obr. 34 SkeletonControl ukázka....39
Obr. 35 Ray Casting implementace....41
Obr. 36 Pipeline knihovny JBullet....42
Obr. 37 Ukázka inicializace fyziky....44
Obr. 38 Zobrazení kolizních modelů....44
Obr. 39 Dynamická úroveň detailů terénu....46
Obr. 40 Převod Float pole na mapu....46
Obr. 41 Generátor fraktálového vzoru....47
Obr. 42 Mapa vygenerovaná z obrázku....47

Seznam tabulek

Tab. 1 Výhody Netbeans oproti Eclipse....4
Tab. 2 Převod stupňů na radiány....7
Tab. 3 Popis uzlů a geometrií....8
Tab. 4 JIT Optimalizace kódu....17
Tab. 5 Složka assets....27
Tab. 6 Statické, Kinematické, Dynamické objekty ve fyzice....43

1 Úvod

Hra na hrdiny přeloženo z anglického role-playing game dále jen RPG, je velmi oblíbený žánr v herním průmyslu, avšak vývoj takovéto hry je velmi náročný, jelikož kombinuje perfektně zpracovaný příběh a atraktivní grafiku, která při hraní umožnuje hráči vžít se do role hlavního hrdiny.

Základem každé RPG hry je příběh, který není lineární a nabízí hráči více možných konců, jak dokončit dějovou linii. RPG hru tvoří primárně hlavní dějová linie, která je doplňována vedlejšími dějovými úkoly, které by měly být pro hráče stejně lákavé jako hlavní linie. Dobře zpracovaný scénář je důležitou úlohou pro vývoj kvalitního RPG. Mnoho vývojářských studií si najímá scénáristy, kteří jim pomáhají s vypracováním hlavních a vedlejších dějových linií, tak aby reflektovaly hráčova rozhodnutí, které mohou ovlivnit vzhled herního světa.

Součástí scénářů musí být popis prostředí, ve kterém se děj odehrává a také detailní popis vzhledu prostředí, architektury a osvětlení scény, která v hráči umocňuje prožitek podle toho, zda prostředí je veselé, barevné či pochmůrné. Zde přichází velký problém, každý člověk si na základě psaného textu může představovat odlišný vzhled prostředí herního světa, ve kterém se děj odehrává, proto vymodelované 3D herní prostředí může působit jiným dojmem, než si představoval autor scénáře.

Při plánování vývoje pokračování herního titulu, některá herní studia naslouchají své fanouškovské komunitě a inspirují se navrhovanými příběhy a prostředím, ve kterých by se pokračování mohlo odehrávat. Mezi schválenými návrhy se ve finále hlasuje a zde jsem si všiml problému, který by moje práce měla vyřešit. Fanoušci nejsou scénáristi, takže amatérské texty nemusí reflektovat to, co chtěli říct a mohou zde vznikat nejasnosti, které mohou ovlivnit hlasování ostatních členů komunity.

2 Cíl práce

Cílem mé práce je vyvinout aplikaci pomocí JMonkeyEnginu, který umožní uživateli vytvořit hratelný příběh a amatérský vzhled 3D světa bez znalosti programování. Při modelování herního světa bude uživatel schopný editovat všechny prvky pouze pomocí předpřipravených nástrojů pomocí uživatelského rozhraní. V textu práce budou představeny části JMonkeyEnginu, které budu využity pro tvorbu aplikace.

Uživatel bude mít k dispozici nástroje pomocí kterých může editovat výšku a rozměry terénu (snižování, zvyšování, zarovnávání, zvrásnění a vyhlazování). Možnost editovat texturu terénu složenou až z 12 vrstev společně s normálovými texturami, které bude aplikovat pomocí kreslení texturou na terén. Editor rozložení a velikosti jednotlivých částí mapy, aby nepoužívaný terén nebyl vykreslován z důvodu optimalizace. Editor rozmísťování geometrických modelů na jednotlivé části mapy, kolizní, nekolizní a počítačem řízené charaktery dále jen NPC z anglického NonPlayer Control. K nasvětlení a doladění vizuální stránky scény bude sloužit editor efektů, kde si uživatel nastaví vykreslování stínů, osvětlení scény, denní dobu, rychlost plynutí času, oblohu, hloubku ostrosti, mlhu a vykreslování paprsků slunce.

Hlavní částí softwaru bude editor na vytváření dějové linie, kde si uživatel v přehledném uživatelském prostředí vytvoří úkoly a jejich jednotlivé fáze, které bude možné editovat v nativním textovém editoru podporovaného operačního systému na kterém aplikace poběží. Součástí dějové linie budou i výchozí texty jednotlivých NPC, které budou hráči zobrazeny, pokud součástí právě aktivního úkolu nebude interakce s danou NPC. Ve vytvořeném světě nemusí být všechny NPC součástí dějové linie, ale informace, které mohou sdělit jsou zajímavé. Např informace o lokalitě, kde se hráč momentálně nachází.

Klíčové na aplikaci je, aby byla snadno rozšiřitelná o nové editační nástroje (např. rozmísťování zvuků) nebo aby bylo možné rozšířit úkoly o nové funkce jako je například řetězení úkolů. Hráč nemůže začít úkol bez splnění přechozích úkolů.

3 Metodika zpracování

Výsledkem bude koncept aplikace, která bude nadále mimo rozsah této práce rozšiřována a následně prezentována vývojářům her. Aplikace bude umožňovat návrh vlastního statického prostředí trojrozměrného světa s rozmístěnými počítačem řízenými postavami, se kterými je možné provést interakci. Při interakci se otevře textové okno, ve kterém bude napsaný text k aktuální příběhové linii, která byla vytvořena autorem dějové linie. Tyto cíle byly vybrány na základě mých vlastních zkušeností s hrami typu RPG, mým zájmem o tvorbu a hraní her a několika letým sledování vývojářských fór a herních trendů. Herní soubory (assety) jako jsou textury a modely nebudou přímou součástí projektu, ale budou stažitelné s následnou možností jejich nahrání do přidělené datové složky.

Cílem je vyvinou aplikaci tak, aby byla uživatelsky velmi přívětivá, lehká na ovládání a porozumění jakémukoliv uživateli počítače se základními znalostmi o žánru RPG her.

JMonkeyEngine a všechny externí knihovny, které využívá jsou open source a podporují velké množství operačních systémů (Mac, Linux, Android, iOS). Zvolil jsem tento engine taky díky jeho abstraktnosti, co se týče žánru vyvíjených her (není soustředěn na konkrétní typ her). Open-source koncept umožnuje snadné rozšiřování velkého množství třídy. V dokumentaci jsou některé třídy přímo doporučeny k rozšíření, dle potřeby, jelikož základní třída tvoří základní funkční koncept.

Tvořené scény se budou periodicky ukládat do datové složky aplikace. Pokud bude třeba někomu vytvořenou scenérii s příběhem poslat, bude stačit zabalit datovou složku a přeposlat bez nutnosti posílání celé aplikace. Vytvořené datové soubory budou uloženy pomocí obrázků, XML souborů a textových souborů, takže v případě, že se bude vývojáři výtvor líbit, může soubory z datové složky využít, protože výsledná aplikace této práce bude distribuována jako open-source.

4 JMonkeyEngine 3

JMonkeyEngine [9], je multiplatformní herní engine pro vytváření 3D her využívající OpenGL shader technologie pro renderování trojrozměrných scén s vysokou snímkovou frekvencí. Umožnuje vyvíjet aplikace jak pro podporované desktopové systémy, tak i pro systém Android 2.3 a vyšší. Podporované systémy musí mít nainstalovanou Javu verze 5 a vyšší a grafická karta musí podporovat OpenGL 2.0 a vyšší. Engine je distribuován pod licencí BSD a může být stažen jako knihovna, kterou je možné přidat do libovolného vývojářského prostřední nebo jako sada vývojářských nástrojů z anglického Software Development Kit zkráceně SDK založený na vývojovém prostředí NetBeans. Výhodou toho SDK je implementace několika pluginů, které velice usnadní vývoj aplikace. V tabulce č. 1 je uveden seznam bodů, kterýma odůvodňují vývojáři použití prostředí NetBeans, oproti populárnějšímu nástroji Eclipse.

1. Eclipse používá proprietární GUI systém (SWT), NetBeans používá implementaci Java AWT, pro kterou v LWJGL/JME3 existuje vysoce výkonný Canvas, česky plátno. Kompatibilita s AWT povoluje implementaci několika pluginů jako je například NeoTexture.
2. Projekty v Eclipse jsou proprietární a nelze je otevřít bez použití Eclipse. Platforma NetBeans používá standard ANT Builder, který funguje i mimo vývojové prostředí a může být rozšířen o další procesy sestavení, které ANT Builder používají. Projekty generované v prostřední JMonkeyEngine SDK je následně možné otevřít v prostředí Eclipse.
3. Neexistuje žádný způsob, jak projekty v Eclipse řádně rozšířit.
4. Eclipse nenabízí Nodes API, které umožnuje snadné zabalení grafu scén do vizuální reprezentace.
5. Eclipse má pouze komerční editory GUI, NetBeans má bezplatný AWT GUI editor pro snadný navrhování pluginů.
6. Eclipse a NetBeans jsou stejně funkční.
7. Hlavní vývojáři JMonkeyEnginu používají primárně NetBeans vývojové prostředí.

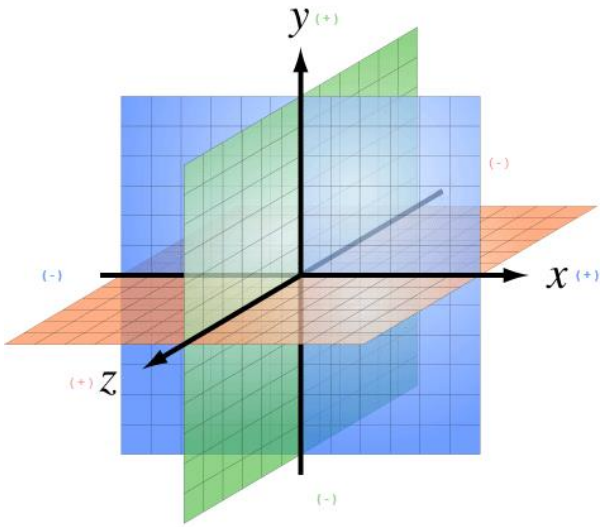
Tab. 1 Výhody Netbeans oproti Eclipse
Zdroj: [7]

Historická poznámka: Vývoj enginu začal *Mark Powell* v roce 2003. Později se k němu připojil *Joshua Slack* a začali vyvíjet engine aby dokázali, že programovací jazyk Java má dostatečný výkon na zpracování a renderování plnohodnotné 3D hry.

V průběhu beta testování byl engine přejmenován z JMonkeyPlatform na JMonkeyEngine. V roce 2008 byla vydána verze enginu 2.0 a téhož roku *Joshua Slack* opustil projekt. V roce 2010 byla oficiálně vydána verze enginu 3.0, kdy společně s enginem byla představena první alpha verze JMonkeyEngine SDK. V roce 2014 byla uvedena stabilní verze SDK, která se stala hlavním a doporučovaným nástrojem pro vývoj her v JMonkeyEnginu. Díky využívání knihovny Lightweight Java Game Library, která podporuje verzi OpenGL2 až po nejnovější v době psaní práce OpenGL 4.6 je zajištěno využití plného výkonu i u nejnovějších grafických karet.

4.1 3D prostor

JMonkeyEngine používá pravotočivou kartézskou soustavu souřadnic s počátkem v bodě x=0, y=0, z=0. Orientaci os znázorňuje obrázek č. 1. Každý bod ve 3D prostoru je definován souřadnicemi X, Y, Z, popisují polohu vzhledem k počátku. K definování pozice v JMonkeyEnginu se používá třída com.jme3.math.Vector3f. která v konstruktoru přijímá 3 parametry typu float. *„Jednotka vzdálenosti (jeden krok) v JME3 je nazvána World Unit zkráceně wu. Typicky se 1wu považuje za jeden metr.“* [7]



Obr. 1 Souřadnicový systém v JME3
Zdroj: [7]

Třída Vector3f a její dvourozměrný ekvivalent Vector2f obsahují základní metody k práci s vektory, jako je sčítání, odečítání, násobení a dělení vektorů. Tyto operace lze provádět dvěma způsoby. Například sčítání lze provádět pomocí metod

add nebo addLocal. Slovo local zde znamená, že se nebude vytvářet nová instance třídy jako návratový typ, ale upraví se existující instance, nad kterou je tato metoda zavolána. Tento přístup je jednou z optimalizačních možností aplikace převážně u starších systémů Android, u kterých je potom rychlost operací podstatně větší. *„Starší systémy Android, používají velmi pomalý Garbage Collector, který musí zpracovat každý objekt zvlášť, a to jak během jeho vzniku, tak i během jeho zániku, takže vytváření nových lokálních proměnných způsobuje velké množství práce pro Garbage Collector, pokud je funkce často volána.“* [7]

Pro vytváření rotací objektů ve 3D prostoru se v JMonkeyEnginu nejčastěji používá třída com.jme3.math.Quaternion. Quaterniony představují rozšíření množiny komplexních čísel ve 3D. Platí zde např vztah: $i^2 = j^2 = k^2 = ijk = -1$. Umožnují zobrazit jakoukoliv rotaci ve 3D prostoru použitím pouze čtyř proměnnými typu float, v situaci, kdy by rotační matice vyžadovala devět proměnných. Použití méně proměnných vede k efektivnějšímu řetězení rotací, což umožnuje snadnou interpolaci mezi dvěma rotacemi. Pochopení matematických operací za těmito čtyřmi proměnnými může být poměrně obtížné, proto tato třída obsahuje velký počet jednoduchých metod, které usnadní nastavení X, Y, Z, W pomocí prostředků reprezentujících rotace. Nejpoužívanější metody jsou fromAngelAxis(float, Vector3f), fromAngles(float[]) a fromAxes(Vector3f[]), jejich použití je vidět na obrázku č. 2. V tabulce č.2 pod obrázkem je znázorněn převod stupňů na radiány. Třída Quaternion nám umožnuje nastavování absolutní rotace k aplikování relativní rotace na Spatial, což je každý objekt ve 3D prostoru (detailní popis níže) slouží metoda Spatial.rotate(X, Y, Z), které jako parametr předáváme rotaci v radiánech. Snadný převod úhlu na radiány umožnuje konstanta z balíčku FastMath. Dvojitým zavoláním metody rotate(0f, FastMath.DEG_TO_RAD * 90f, 0f) dosáhneme rotace objektu o 180 stupňů vůči původní rotaci po ose Y.

```
//fromAngleAxis
Vector3f axis = Vector3f.UNIT_Y; // ekvivalent k vektoru (0, 1, 0)
float angle = FastMath.PI //rotace o 180 stupnů
Quaternion rot = new Quaternion().fromAngleAxis(angle, axis);

//fromAngel
float[] angles = {1, 3, 0}; //rotace o 1 radian na ose X a o 3 radiany na ose Y
Quaternion rot = new Quaternion().fromAngles(angles);

//fromAxes
//rotace nahoru o 45 stupnů
Vector3f[] axes = new Vector3f[3];
axes[0] = new Vector3f(-1, 0, 0); //vlevo
axes[1] = new Vector3f(0, 0.5f, 0.5f); //nahoru
axes[2] = new Vector3f(0, 0.5f, 0.5f); //směr
Quaternion rot = new Quaternion().fromAxes(axes);
```

Obr. 2 Rotace pomocí Quaternionu
Zdroj: Vlastní tvorba

Úhel ve stupních	Úhel v radiánech
45 stupňů	FastMath.PI / 4
90 stupňů	FastMath.PI / 2
180 stupňů	FastMath.PI
270 stupňů	FastMath.PI * 3 / 2
360 stupňů	FastMath.PI *2
x stupňů	FastMath.PI * x / 180

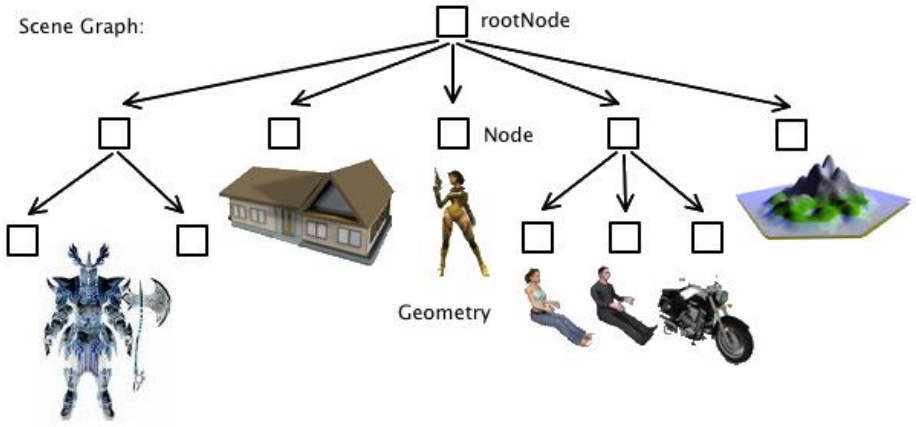
Tab. 2 Převod stupňů na radiány
Zdroj: [7]

Trojrozměrný prostor v JMonkeyEnginu se nazývá Scene Graf, česky graf scény. Graf scény je obecná datová struktura používaná pro vektorovou grafiku a scény v počítačových hrách, která je kolekcí uzlů ve struktuře grafu nebo stromu. Ve scéně je jeden hlavní uzel, v JMonkeyEnginu nazvaný rootNode, který je rodičem všech ostatních objektů přidaných do scény. Všechny tyto objekty v grafu scény jsou instancí abstraktní třídy Spatial. *„Spatial se nikdy neinstancuje pomocí Spatial s = new Spatial(). Protože je to abstraktní koncept. Je možné vytvořit instanci třídy com.jme3.scene.Node nebo com.jme3.scene.Geometry. Některé metody přesto vyžadují typ Spatial, protože jako argument přijímají Node i Geometry. V tomto případě se Node nebo Geometry přetypovává na Spatial) “* [7]

V tabulce č. 3 a obrázku č. 3 jsou popsány vlastnosti tříd Node a Geometry, které rozšiřují třídu Spatial. Tyto dvě třídy se v grafu scény používají k jiným účelům.

Každá aplikace v JMonkeyEnginu rozšiřuje třídu com.jme3.app. SimpleApplication. Tato třída inicializuje dvě instance třídy Node rootNode a

guiNode. Uzel rootNode je rodičem všech Spatials, které budou součástí scény ve 3D prostoru. Uzel guiNode je rodičem všech Spatials přidané do prostoru vykreslovaném vždy před kamerou.



Obr. 3 Graf scény
Zdroj: [7]

	com.jme3.scene.Spatial	
Popis:	Spatial je abstraktní datová struktura ukládající uživatelské data a transformace (posun, rotaci a měřítko) jednotlivých elementů grafu scény.	
	com.jme3.scene.Geometry	com.jme3.scene.Node
Viditelnost:	Geometrie reprezentuje viditelný 3D objekt v grafu scény	Uzel je neviditelný a drží skupinu Spatials v grafu scény
Popis:	Geometrie se používá k reprezentování vzhledu objektu. Každá geometrie obsahuje síť polygonů a materiálů, specifikující její tvar, barvu, texturu a průhlednost. Geometrie se připojují k uzlům.	Uzel je struktura nebo skupina geometrií nebo uzlů. Každý uzel je připojený k jednomu rodiči a každý uzel může mít nula a více potomků (Uzel nebo Geometrie) připojených k sobě. Když transformujete (posunujete, rotuje, atd) rodičovský uzel, všechny jeho potomci jsou také transformovány.
Obsahují:	Transformace, uživatelská data, tvar, materiál	Transformace, uživatelská data
Příklad:	Krabice, koule, hráč, budova, terén, vozidlo atd.	rootNode, guiNode, audioNode a vlastní skupiny uzlu

Tab. 3 Popis uzlů a geometrií
Zdroj: [7]

4.2 SDK

Vývojové prostředí bylo vytvořeno přímo pro požadavky enginu a je souběžně aktualizováno s vývojem enginu. Při vytváření projektu je inicializováno několik složek, mezi kterými jsou základní dvě složky assets a src. Do asset jsou ukládány všechny soubory, které jsou potřeba pro běh aplikace a při kompilování aplikace je složka zabalena jako jar archiv, pokud engine rozpozná podporované formáty souborů viz. tabulka č.5. Do src se ukládají všechny zdrojové kódy aplikace a struktura balíčku je zde plně v režii programátora.

Jde o velmi účinné prostředí „JMonkeyEngine SDK byste měli alespoň zkusit, protože obsahuje mnoho pluginů, které nenajdete v ostatních vývojových prostředích: správce assetů, převaděč modelů do binárních formátů optimalizovaných pro JMonkeyEngine, Scene Explorer, editor materiálu, editor a generátor textur, paleta kódů a mnoho dalšího.“ [10]

Správce assetů umožnuje importovat modely, textury, zvuky a další z veřejných uložišť do projektů. Při importování modelů vytvoří SDK soubory pro model, definici materiálu a textury ve formátu optimalizovaném pro engine. Modely mají koncovkou j3o, materiály koncovku j3m/j3md a shadery j3f, vert + frag.

Editor textur umožnuje základní operace s obrázky, nástroje pro generování světelných a normálových map na základě nastavených parametrů. Herní modely mají nízký počet polygonů, proto je použití normálových textur velice vhodné, protože i z málo detailního hranatého modelu, vytvoří hladce vypadající model při mnohem nižším výkonu, než při použití modelů s velkým množstvím polygonů.

Editor trojrozměrných scén zvaný Scene Composer umožnuje přidávat do scény objekty, upravovat transformace objektů, přidávat emitory částic, zdroje světla a generovat kolizní model objektů. Takto vytvořenou scénu je možné uložit a v kódu následně načíst.

Paleta kódů obsahuje funkce pro generování základních struktur kódu. Umožňuje např. vkládání filtrů scény nebo zdrojů světla. Generování funguje principem Drag and Drop, kdy stačí uchopit tlačítko s požadovanou funkcí a přesunou ho na místo v kódu, kde se následně vygeneruje kód.

4.3 Blender

Blender [2] je bezplatný multiplatformní open source software pro vytváření 3D obsahu s použitím OpenGL technologie a širokou škálou pluginů napsaných v jazyce Python. Nabízí širokou škálu nástrojů jako je modelování, tónování, animování, střih videa, texturování, vytváření koster modelů a mnoho typů simulací.

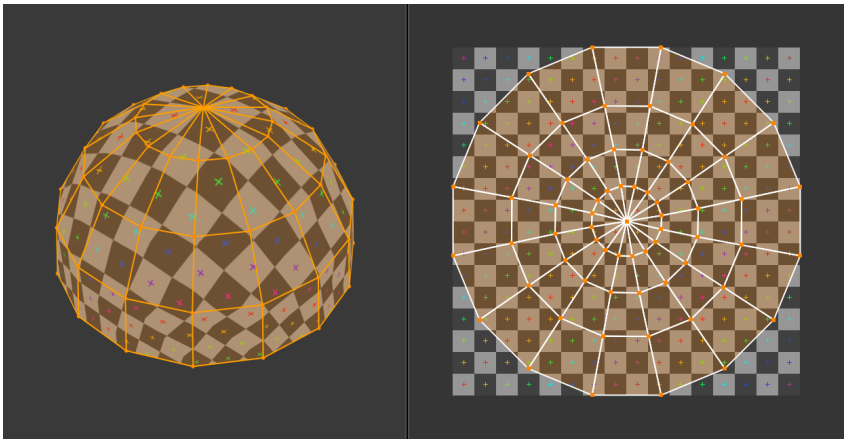
Historická poznámka: *Ton Roosendaal* vlastnil od roku 1988 animační studio NeoGeo, které se stalo populárním po celé Evropě. V roce 1994 začal vyvíjet software Blender licencovaný jako open source pod značkou Not a Number, která byla dceřinou společností NeoGeo, jelikož software, který do té doby používali byl už zastaralý. Byl to revoluční koncept softwaru, protože v té době stály všechny programy pro tvorbu 3D grafiky tisíce dolarů.

JMonkeyEngine si vybral Blender jako primární nástroj pro editaci 3D modelů, takže je možné pomocí Scene Exploreru v SDK načíst soubory s koncovkou blend a následně je automaticky konvertovat do binárních souborů enginu. Vysoká nabídka pluginů pro Blender také zajišťuje pokrytí a následnou konverzi všech rozšířených formátů modelů od jiných nástrojů pro tvorbu 3D scén.

Využití souboru blend je jednoduché, ale uživatel si musí hlídat cestu k texturám, které model využívá, aby se tomuto problému vyhnul byla do SDK implementována podpora Ogre Mesh, který byl vyvinut jako plugin do Blenderu. Ogre Mesh ukládá modely do dvou souborů ve formátu XML. První soubor obsahuje model, jeho kostru a animace. Druhý soubor popisuje mapování textur a materiály s využitím relativních cest k texturám. SDK následně umí soubory formátu Ogre Mesh konvertovat do binárních souborů.

4.3.1 UV Mapování Textur

UV Mapování Textur Je jednou z nejpoužívanějších a nejflexibilnějších technik k texturování 3D modelů. U této techniky je prostorový trojrozměrný model definovaný body X, Y, Z automaticky rozložen na plochý dvourozměrný model a uložený jako bitmapa, u které každý pixel U, V reprezentuje jeden bod v trojrozměrném prostoru X, Y, Z.



Obr. 4 Transformace modelu na bitmapu
Zdroj: [1]

Na obrázku č. 4 je vidět plocha 3D modelu, která se liší od bitmapového prostoru reprezentovaného body U, V. Tento rozdíl je způsoben mapováním 3D částí modelu do 2D plochy. Na takto mapované modely je možné kreslit a změny se v reálném čase projevují na 3D modelu.

Procedurální textury, které se opakují a jejich okraje do sebe zapadají se využívají např. při texturování terénu, ale např. pro texturování kůže na lidském těle nejsou vhodné. Procedurálními texturami nedocílíme efektu zvrásčení kůže, proto se zde používá UV mapování textur. Takový model můžeme opět uložit jako blend anebo exportovat pomocí Ogre Mesh do JMonkeyEnginu a pomocí SDK převést na binární soubor. *„UV mapování je nezbytné v herní enginu nebo v jakékoliv jiné hře. Je to de facto standard pro aplikaci textur na modely. Téměř každý model, který najdete ve hrách je texturován pomocí UV-Mapování*“ [1]

4.4 Nifty GUI

Nifty GUI [14] je Java knihovna sloužící k vytváření interaktivního uživatelského rozhraní dále jen GUI, které je optimalizované k integraci s různými grafickými systémy jako je JME3, LWJGL, JOGL a další. Definice vzhledu a chování GUI je uložena v XML souboru používající vlastní XSD anebo napsána přímo v Javě. V aplikaci, která má být touto prací vytvořena, bude použito implementace pomocí Java kódu, protože při velkém množství elementů v GUI je snadnější a přehlednější správa jednotlivých ID, které jsou unikátní pro každý element. Každá obrazovka tvořící GUI, která má být interaktivní, musí mít přiřazený ovladač. Ovladač je třída,

která implementuje rozhraní ScreenController, následně je možné v takovéto třídě zpracovávat různé události vyvolané interakcí s GUI pomocí anotací nad metodami. Na obrázku č. 5 vidíme základní inicializace Nifty Gui s nastavením výchozího stylování vzhledu a mapování tlačítek. Tyto dva xml soubory jsou součástí knihovny, takže se nemusí explicitně definovat.

```
NiftyJmeDisplay niftyDisplay = NiftyJmeDisplay.newNiftyJmeDisplay(
    assetManager, inputManager, audioRenderer, guiViewPort);
_nifty = niftyDisplay.getNifty();
_nifty.loadStyleFile("nifty-default-styles.xml");
_nifty.loadControlFile("nifty-default-controls.xml");
guiViewPort.addProcessor(niftyDisplay);
```

Obr. 5 Inicializace Nifty v JME3
Zdroj: Vlastní tvorba

Na obrázku č. 6 je vidět implementace rozhraní ScrenController, které obsahuje metody bind, onStartScreen a onEndScreen. Metoda bind je volána při vytvoření obrazovky, metoda onStartScreen při přepnutí na obrazovku obsahující tento ovladač a onEndScreen metoda při opouštění této obrazovky. Anotace NiftyEventSubscriber nám označuje metody, které budou volány při interakci uživatele s GUI. Metoda mySlider bude zavolána, pokud uživatel provede interakci s elementem posuvník, který má ID Slider. Metoda myButtons používá v anotaci pattern, pomocí kterého můžeme přiřadit jednu metodu k více tlačítkům s různým ID. V tomto konkrétním případě bude metoda zavolána při interakci se všemi tlačítky jejichž ID začíná textem btn.

```
public class MyClass implements ScreenController
{
    @NiftyEventSubscriber(pattern = "btn.*")
    public void myButtons(String id, ButtonClickedEvent event){}

    @NiftyEventSubscriber(id = "Slider")
    public void mySlider(String id, SliderChangedEvent event){}

    @Override
    public void bind(final Nifty nifty, final Screen screen){}

    @Override
    public void onStartScreen(){}

    @Override
    public void onEndScreen(){}
}
```

Obr. 6 Ovladač obrazovky
Zdroj: Vlastní tvorba

Využití knihovny Nifty Gui u aplikací vytvořené v JMonkeyEnginu není nutné, je možné použít novější externí knihovnu Lemur Gui, která je napsána přímo pro

JMonkeyEngine a využívá pouze jeho komponenty. Lemur Gui není oproti Nifty Gui součástí JMonkeyEngine knihovny, takže je nutné ho do projektu zahrnou zvlášť. Lemur Gui zatím nenabízí takové množství funkcí jako Nifty Gui. Uživatelské rozhraní je možné vytvořit vlastní s použitím připravených komponent pro renderování modelů a textů do uzlu guiNode, které jsou renderovány vždy před kamerou. Na obrázku č. 7 je vidět připnutí textu ke guiNode. Pomocí instance třídy AssetManager (více v kapitole 6.3) načteme výchozí font a vytvoříme instanci třídy com.jme3.font.BitmapText, které nastavíme velikost textu a vlastní text. Následně text umístíme na střed obrazovky a připneme k uzlu guiNode. Tento uzel používá k pozicování trojrozměrný vektor (X, Y, Z) stejně jako rootNode, ale počátek (0, 0) je umístěný v levém dolním rohu a v pravém horním rohu má souřadnice (výška displeje, šířka displeje) takže 1wu se rovná jednomu pixelu. Souřadnice Z slouží k překreslování jednotlivých elementů přes sebe a používá se rozmezí -1 a 1.

```
BitmapFont font = assetManager.loadFont("Interface/Fonts/Default.fnt");
BitmapText text = new BitmapText(font, false);
text.setSize(font.getCharSet().getRenderedSize() * 2);
text.setText("+");
text.setLocalTranslation(
    cam.getWidth() / 2 - text.getLineWidth() / 2,
    cam.getHeight() / 2 + text.getLineHeight() / 2, 0);
guiNode.attachChild(text);
```

Obr. 7 Vytvoření vlastního textu na obrazovce
Zdroj: Vlastní tvorba

4.5 Lightweight Java Game Library

Lightweight Java Game Library [11] zkráceně LWJGL je multiplatformní knihovna, která zprostředkovává rozhraní k přímému přístupu skrze JNI (Java Native Inteface) do nativních knihoven OpenGL, Vulkan, OpenAL a OpenCL a dalších od společnosti Khronos Group, které jsou napsány v jazyce C nebo C++. Tento přístup je přímý prostřednictvím typově bezpečné a uživatelsky přívětivé vrstvy. Jedná se o nízkoúrovňový přístup, takže využití je pro nováčka mnohem jednodušší s použitím frameworků nebo herních enginů, jako je například JMonkeyEngine. Jedním ze známých využití této knihovny je populární hra Minecraft.

4.5.1 OpenGL

Open Graphics Library dále jen OpenGL je multiplatformní rozhraní pro vykreslování 2D a 3D vektorové grafiky pomocí GPU. Knihovna Lightweight Java Game Library využívá implementace JOGL, což je OpenGL implementace v jazyce Java. *„Dodavatelé hardwaru, kteří vytvářejí GPU, jsou zodpovědní za napsání implementace systému OpenGL. Jejich implementaci běžně nazýváme „ovladače", které převádějí příkazy OpenGL API do příkazů GPU."* [15] Pokud hardware není schopen implementovat všechny funkce OpenGL, dodavatel hardwaru musí tyto funkce poskytovat, proto tyto funkce dodává softwarově pomocí funkcí, které v hardwaru chybí. [15]

„Veškeré zobrazované informace na displeji jsou většinou uloženy ve „frame bufferu". OpenGL umožnuje vykreslovat přímo do paměti grafické karty skrze „Frame Buffer Object" (FBO), který je mnohem rychlejší než frame buffer." [22]

JMonkeyEngine pro vykreslování scén využívá shader technologii, která modifikuje renderovací pipelinu. Shadery jsou napsané v jazyce OpenGL Shading Language. *„Shader technologie je rozšíření OpenGL knihovny a je užitečná pro vykreslování s mnohem větší grafickou kvalitou. Tato technologie může být také použita pro náročné výpočty."* [22] JMonkeyEngine využívá výpočetní sílu shader technologie mimo grafu scény, také pro transformace geometrických objektů pomocí matic.

4.5.2 OpenAL

OpenAL je multiplatformní audio API, které tvoří softwarové rozhraní pro audio hardware. Knihovna byla vytvořena společností Loki Software v roce 2000 a po jejím zániku se stala open-source knihovnou a nadále byla vyvíjena komunitou s velkou podporou od společnosti Apple. Knihovna je navržena pro efektivní přehrávání vícekanálového trojrozměrného pozičního zvuku, zejména v počítačových hrách. OpenAL umí do her přidat realismus díky filtrování zvukového výstupu pomocí degradace zvuku v závislosti na vzdálenosti a Dopplerova efektu, který mění frekvenci zvuku v důsledku pohybu a materiálových hustot.

4.6 Java

Java [16] je programovací jazyk 3. generace po boku Pascal, C, C++, C# a dalších. Jde o objektově orientovaný programovací jazyk vyvinutý společností Sun Microsystems, kterou v roce 2010 převzala společnost Oracle. Javu je možné stáhnout ve dvou různých implementacích, Java Runtime Environment dále jen JRE a Java Development Kit JDK. Implementace JRE slouží ke spuštění programů napsaných v jazyce Java běžících na Java Virtual Machine dále jen JVM a implementace JDK slouží k vývoji aplikací v jazyce Java.

„Programy by měly napodobovat skutečný svět. Náš skutečný svět je plný objektů, jako je počítač, který používáte, stůl pod počítačem, učebnice v regálech a tak dále. Objektově orientované programy napsané v jazyce Java by tedy měly popisovat objekty a interakce mezi nimi. Java používá ve svém kódu „třídy" k definování odpovídajících objektů v reálném světe." [23]

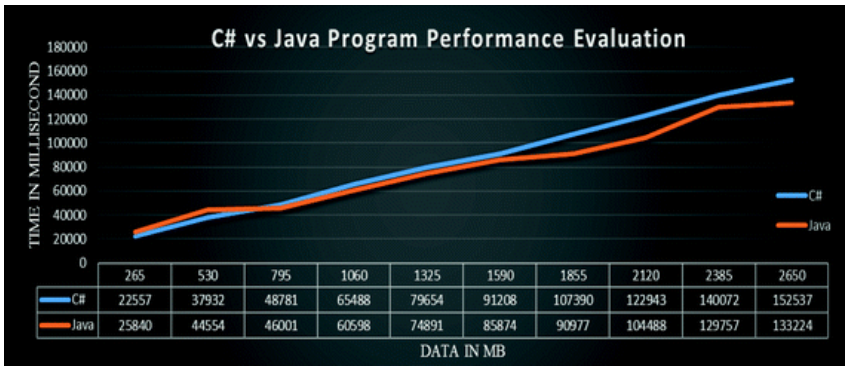
Jazyk je silně typový a relativně vysoko úrovňový jazyk. Silně typová specifikace nám rozlišuje chyby při kompilaci a chyby vzniklé při běhu programu. Kód je překládán do bajtového kódu, takže ho je možné spustit a dynamicky optimalizovat nezávisle na platformě pomocí JVM. To zahrnuje automatickou správu paměti pomocí vysoce výkonného garbage collectoru, díky čemuž jsou nepotřebná data z paměti odstraňovány automaticky. Podobně jako uvolňování paměti pomocí klíčových slov v jazykách C free nebo v C++ delete. Jazyk nepodporuje nezabezpečené volání do neindexované paměti, což vylučuje nežádoucí chování programu. [18]

4.6.1 Výkon a výhody

Vývoj počítačových her se většinou provádí pomocí programovacích jazyků C++ a C# od společnosti Microsoft, protože mají dobrý výkon a je v nich napsáno hodně herních enginů. *Mark Powell* se rozhodl toto změnit a začal vyvíjet JMonkeyEngine, aby dokázal, že i Java má dostatečný výkon na simulaci reálné fyziky a vykreslování atraktivních scén.

Porovnávací testy jazyka C# a Java při práci s velkými daty ukazují, že C# je rychlejší při začátku aplikace, ale při zvětšování objemu dat Java vykazuje lepší

chování a potřebuje méně procesorového času ke zpracování dat než C#, viz. obrázek č. 8. [21]



Obr. 8 Porovnání výkonu C# a Java
Zdroj: [21]

Just In Time zkráceně JIT kompilace kódu je dynamická optimalizační metoda Java Virtual Machine, která optimalizuje Java bytový kód při běhu aplikace. Tato metoda umožnuje přeložit bytový kód do strojového kódu, který bude lépe optimalizovaný pro daný procesor a využije všechny jeho dostupné instrukce. Přístup JIT takto optimalizuje jenom metody aplikace, které jsou zrovna potřeba. Protože JIT optimalizace je volána při každém volání metody, tak se tyto optimalizované metody ukládají do cache, takže jednou kompilovaný kód se už nekompiluje, čímž dochází k urychlení běhu aplikace. V tabulce č. 4 je ukázka optimalizace kódů bodově rozepsaná.

Kroky optimalizace	Změna kódů	Popis

Počáteční stav	public void foo() {
	y = b.get();
	...do stuff...
	z = b.get();
	sum = y + z;}	

1. Úprava finální metody	public void foo() {
	y = b.value;
	...do stuff...
	z = b.value;
	sum = y + z;}
		b.get() byl nahrazen b.value protože přistupování k proměnným přímo má menší latenci, než volání pomocí funkce

2. Odstranění redundantního načítání	public void foo() {
	y = b.value;
	...do stuff...
	z = y;
	sum = y + z;}
		z = b.value bylo nahrazeno z = y latence bude snížena, protože přistupování k lokální proměnné je rychlejší

3. Kopírování proměnných	public void foo() {
	y = b.value;
	...do stuff...
	y = y;
	sum = y + y;}
		z = y byl nahrazen y = y, protože není potřeba vytvářet nové proměnné, když se hodnoty rovnají

4. Odstranění nepotřebného kódu	public void foo() {
	y = b.value;
	...do stuff...
	sum = y + y;}
		y = y je nepotřebný kód, takže byl odstraněn

Tab. 4 JIT Optimalizace kódu
Zdroj: [17]

Aplikace napsané v jazyce Java se oproti jazyku C++ mnohem snadněji debugují. Pomocí nástrojů obsažených ve vývojových prostředích je k dispozici přehledný výpis trasy kódu, který vedl k selhání aplikace. Pomocné programy od společnosti Oracle, Java Flight Recorder a Java Mission Control dokáží zobrazit detailní přehled o chování spuštěné aplikace, zobrazí Hot Spoty, což jsou často volané metody, vytížení jednotlivých vláken a správu paměti při běhu aplikace. Pomocí těchto informací je možné aplikaci ručně optimalizovat a docílit tak většího výkonu.

5 Počítačové hry

Hraní počítačových her je dobrovolná činnost, proto je potřeba motivovat hráče a zvětšovat jeho zájem k delšímu a opětovnému hraní hry. Klíčové je zde hráče průběžně odměňovat drobnými úspěchy, které ho popohání k získávání dalších odměn. Tyto principy se začaly v posledních letech používat i v reálném světe v neherních odvětví a nazývají se Gamifikace. Jedná se o techniku marketingu, která vede ke zvyšování zájmu klientů prostřednictvím herních mechanik, do té doby používaných pouze ve hrách.

Počítačové hry se dělí na dvě velké kategorie. AAA hry, které jsou vyvíjeny velkými společnostmi a mají milionové rozpočty a Indie hry vyvíjené skupinami nadšenců s nízkým rozpočtem. Obě tyto kategorie mají společnou strategii, rozdělit rozpočet na vývoj a marketing, aby bylo dosaženo největších zisků. Snížit náklady na vývoj hry je možné, ale nesmí to negativně ovlivnit kvalitu hry do takové míry, že i přes větší rozpočet pro marketing budou nižší prodeje. Velké množství dnešních Indie her se dostává do takzvaného předběžného přístupu. Tato taktika umožnuje vývojářským studiím získat finance potřebné k dokončení svého herního titulu,

zpropagovat ho a zároveň získat zpětnou vazbu od hráčů. Předběžný přístup je ale dvousečná zbraň, některé chyby rozpracované hry se promíjí, avšak některé mohou způsobit velmi špatné hodnocení a následné odmítnutí titulu celou komunitou. Alternativou předběžného přístupu u AAA her je uzavřené Beta testování, do kterého má přístup výběr z lidí, kteří si hru předobjednali.

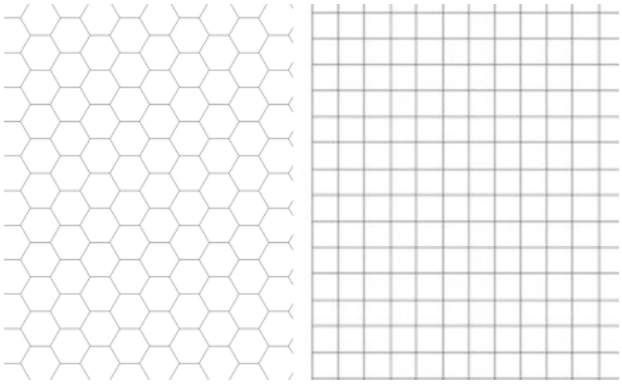
Sellar [20] ve své práci uvádí Clantonovo [4] rozdělení *„Human Computer Interaction (HCI) rozděluje počítačové hry do tří úrovní. Herní rozhraní (ovládání a uživatelské rozhraní), herní mechaniky (fyzika, denní cyklus a jiné zákony z reálného světa) a herní prožitek (prozkoumávání a dosahování herních cílů)“* [20] Míru času, jakou studio do těchto tří kategorií věnuje se většinou odvíjí od žánru daného titulu.

5.1 Hra na hrdiny

Hra na hrdiny, anglicky role-playing game zkracované na RPG. Jedná se o druh hry, ve které se hráč vžívá do role fiktivní postavy. Jeho chování je omezováno pravidly a možnostmi dané hry. Tyto hry jsou založené na vyprávění příběhu, který je hráči přímo ovlivňovány. Postavy hráčů průběžně získávají lepší statistiky. Pravidla mohou být v průběhu dovytvářena a rozšiřována, proto musí být řízena inteligentním prvkem, který dohlíží na chování postavy a na základě činů poupravuje podobu RPG světa.

5.1.1 Klasické RPG

Ke hraní klasických RPG her se nejčastěji používají různé variace kostek, tužky a papír. Tento žánr v minulosti proslavila velmi úspěšná hra *Dungeons and Dragons*. Roli inteligentního prvku zde zastupuje člověk, který je nejčastěji nazýván „Pán Jeskyně“ neboli PJ, který obstarává vyprávění příběhu, rozhodování postav, které nejsou ovládány hráči a reakce světa na činy hráčů. Faktor náhody v RPG hře ovlivňuje hod kostkou, kde větší číslo znamená větší štěstí. Každá herní postava má své statistiky jako je například síla, obratnost či inteligence a podle těchto statistik a hodu kostkou se PJ rozhodne, zda se hráčovi činy povedly. Prostor, ve kterém se děj odehrává je většinou znázorňován na čtverečkovaném nebo hexovém herním plánu viz. obrázek č. 9, který PJ připravuje před každou hrou.



Obr. 9 Hexový a čtverečkovaný herní plán
Zdroj: Vlastní tvorba

5.1.2 Počítačové RPG

Počítačové RPG hry jsou podobné stolním RPG hrám, avšak herní mechanismy jsou omezeny předpřipravenými scripty, které umožnují hernímu světu reagovat na hráčovi akce. Většinou se jedná o větvení dialogů a umělou inteligenci počítačem řízených postav, které se označují slovem NPC z anglického NonPlayer Control. Každá z těchto NPC má přidělený vzorec chování a sled akcí které cyklicky provádí. Velkou nevýhodou vůči stolním hrám, je omezení hráčova rozhodnutí. Může dělat věci, které byly předem připravené a pohybovat se po vytyčeném prostoru, proto jsou kladeny vysoké požadavky na výtvarníky a scénáristy. Musí vytvořit nelineární příběh, který je vyprávěný pomocí úkolů či dialogů s NPC.

V moderních hrách není důležitá pouze hlavní dějová linie, ale také vedlejší úkoly. Pomohou hráči lépe se vcítit do prostředí ve kterém se děj odehrává. Hráč za plnění úkolů dostává zkušenosti, pomocí kterých si vylepšuje svoji herní postavu, tak jako u stolních her. Největší rozmach RPG her byl v 90. letech 20 století.

V posledních letech začal být populární žánr odvozený od RPG s názvem massively multiplayer online role-playing game zkráceně MMORPG, v českém jazyce online RPG s obrovským počtem lidí. Tento žánr upouští od nelineárnosti příběhu, ale dává hráčů velkou svobodu ve způsobech, jakým mohou mezi sebou interagovat.

6 Vývoj aplikace v JME3

6.1 Třída SimpleApplication

Základní třída v JMonkeyEnginu je com.jme3.app.SimpleApplication rozšiřující třídu LegacyApplication. SimpleApplication je rozšiřována vygenerovanou centrální třídou Main, která je vždy výchozím bodem aplikace v nově vytvořeném projektu pomocí SDK. Tato generalizace umožnuje přístup k základním funkcím hry jako je graf scény (rootNode), správce assetů (assetManager), uživatelské rozhraní (guiNode), správa vstupů od uživatele (inputManager), správce zvuku, simulace fyziky a ke kameře (více o těchto funkcích v následujících kapitolách). Ve třídě Main stačí zavolat metodu app.start() ke spuštění a app.stop() k vypnutí aplikace. *„Každá hra (přímo nebo nepřímo) rozšiřuje SimpleApplication právě jednou centrální třídou aplikace. Pokud je třeba přístup k některým funkcím SimpleApplication z jiné třídy, rozšiřuje se druhá třída o AbstractAppState.“* [7]

```
public class Main extends SimpleApplication {

    public static void main(String[] args){
        Main app = new Main();
        app.start();
    }

    @Override
    public void simpleInitApp() {
        /* Inicializace herní scény */
    }

    @Override
    public void simpleUpdate(float tpf) {
        /* (volitelné) Kód pro interakci, který bude proveden při každé updatovací smyčce */
    }

    @Override
    public void simpleRender(RenderManager rm) {
        /* (volitelné) Rozsáhle modifikace frame bufferu a grafu scény */
    }
}
```

Obr. 10 Inicializace třídy Main
Zdroj: Vlastní tvorba

Při inicializaci nového projektu bude vytvořena třída Main tak jak je zobrazeno na obrázku č. 10, která implementuje abstraktní metodu simpleInitApp ze třídy SimpleApplication, pomocí které je inicializován výchozí stav aplikace (viz. kapitola 6.2) po spuštění. Metody simpleUpdate a simpleRender jsou volány při každém aktualizačním cyklu aplikace. Rychlost cyklu u simpleUpdate je závislá od

rychlosti zařízení, na kterém je aplikace spuštěna, proto má parametr typu float, který udává čas v milisekundách od posledního průběhu. SimpleRenderer je volán po dokončení simpleUpdate a dostává jako parametr RenderManager sloužící k pokročilým modifikacím frame buffer a grafu scény .

Třída LegacyApplication, která je použita k rozšíření třídy SimpleApplication představuje aplikaci pro vykreslování 3D scény v reálném čase. Jelikož se jedná pouze o vykreslení 3D scény bez možnosti pohybu a interakce, tak je tato třída většinou nepoužívána k vytvoření vlastní hry. Tato třída inicializuje:

1. viewPort který slouží pro zobrazování grafu scény a je k němu možné registrovat pokročilé post procesorové filtry.
2. cam výchozí kamera poskytující perspektivní projekci scény.
3. settings používající objekt AppSettings pro specifikování šířky a výšky okna, barevnou hloubku, z-buffer, vzorník vyhlazování hran a další.
4. assetManager sloužící k správě cest a načítání assetů (modelů, textur, materiálů, zvuků atd.)
5. audioRender používající se k přístupu do JME audio systému.
6. listener objekt reprezentující zvukový přijímač pro JME audio systém.
7. inputManager starající se o odposlouchávání vstupů od uživatele (myš, klávesnice, ovladač a další) a viditelnost myši.
8. stateManager spravující jednotlivé stavy aplikace, které implementují rozhraní AppState, jako je například fyzika.

Třída obsahuje metody potřebné pro základní chod aplikace (start, stop, restart a setPauseOnLostFocus). Restart slouží k načtení nového nastavení potom co bylo změněno, viz. obrázek č. 11, popisující přepnutí aplikace na celou obrazovku. Metoda setPauseOnLostFocus přijímá jako parametr boolean a slouží k upravení chování okna aplikace. Pokud uživatel přepne do Windows nebo jiné aplikace, tak výchozí nastavení pozastaví aplikaci. Nastavení na hodnotu false se hodí, pokud se vyvíjená hra hraje v reálném čase nebo je pro více hráčů.

```
public void toggleToFullscreen() {
  GraphicsDevice device = GraphicsEnvironment.getLocalGraphicsEnvironment().getDefaultScreenDevice();
  DisplayMode[] modes = device.getDisplayModes();
  int i=0; // note: there are usually several, let's pick the first
  settings.setResolution(modes[i].getWidth(),modes[i].getHeight());
  settings.setFrequency(modes[i].getRefreshRate());
  settings.setBitsPerPixel(modes[i].getBitDepth());
  settings.setFullscreen(device.isFullScreenSupported());
  app.setSettings(settings);
  app.restart(); // restart the context to apply changes
}
```

Obr. 11 Změna velikosti okna
Zdroj: [7]

Třída SimpleApplication je určená k rozšíření centrální třídou Main, protože inicializuje základní funkcionalitu hry, uzly rootNode a guiNode a létací kameru flyCam. RootNode je základní uzel 3D scény, ke kterému jsou připojeny všechny ostatní Spatials (geometrie a uzly). GuiNode je uzel 2D scény, který je vždy vykreslen před kamerou, ale mohou být k němu připojeny i 3D modely, které vždy uvidíme ze stejné strany. FlyCam je výchozí kamera z pohledu první osoby, která má mapované ovládaní pohybu pomocí WASD a rotace pomocí myši. SimpleApplication umožnuje zobrazit informace o vykreslování na displeji při jejím běhu, pokud je tento stav do aplikace přidán. Používá se převážně jenom u vývoje a ladění hry, a poskytují vývojáři informace o množství Spatials ve frame buffer, množství načtených textur, shaderů, počtu trojúhelníků, vrcholů a uniformů anglicky uniforms (předdefinované proměnné, používané pro výpočet shaderů) ve scéně viz. obrázek č. 12. Písmeno S znamená počet změněných při posledním snímků, F počet použitých při posledním snímku a M počet nahraných v OpenGL paměti.

```
FrameBuffers (M) = 6
FrameBuffers (F) = 6
FrameBuffers (S) = 9
Textures (M) = 46
Textures (F) = 44
Textures (S) = 80
Shaders (M) = 25
Shaders (F) = 22
Shaders (S) = 33
Objects = 238
Uniforms = 223
Triangles = 398967
Vertices = 415877
```

Obr. 12 Statistika vykreslované scény
Zdroj: Vlastní tvorba

SimpleApplication v základním konstruktoru inicializuje 4 stavy (StatsAppState, FlyCamAppState, AudioListenerState a DebugKeysAppState), ale je možné si do konstruktoru předat vlastní výchozí stavy hry viz. obrázek č. 13.

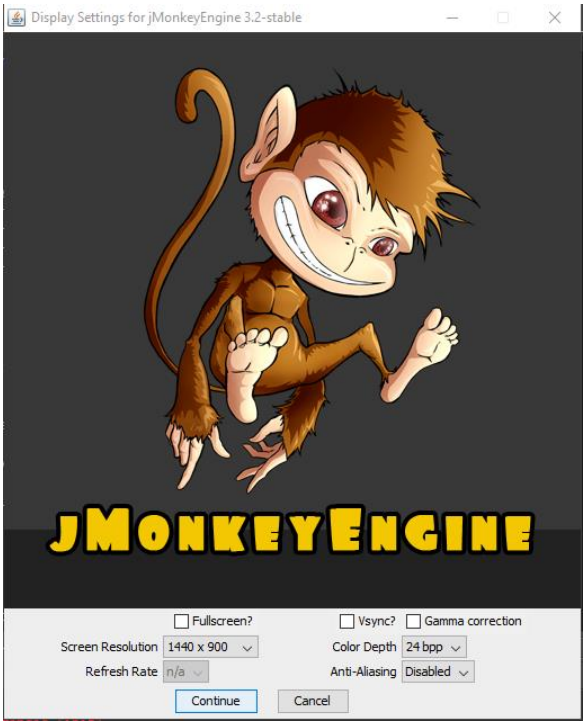
1. StatsAppState – Zobrazení statistik vykreslování hry

2. FlyCamAppState – Výchozí létací kamera pro pohyb ve 3D scéně
3. AudioListenerState – Svázání kamery se zvukovým přijímačem aplikace
4. DebugKeysAppState – Mapování tlačítek C a M k zobrazení informací do konzole o lokaci kamery (viz kapitola 6.5) a využití operační paměti

```
//Main.java
public Main()
{
    super(new StatsAppState(), new AudioListenerState());
}

//SimpleApplication.java
public SimpleApplication() {
    this(new StatsAppState(), new FlyCamAppState(),
      new AudioListenerState(), new DebugKeysAppState());
}

public SimpleApplication( AppState... initialStates ) {
    super(initialStates);
}
```

Obr. 13 Inicializace stavů
Zdroj: Vlastní tvorba

Výchozí implementace aplikace po vytvoření projektu zobrazí při každém spuštění okno, ve kterém si uživatel vybere nastavení viz. obrázek č. 14. Zobrazování okna lze vypnout, pokud bude vytvořeno vlastní nastavení aplikace a toto nastavení předáno aplikaci před spuštěním viz. obrázek č. 15.



Obr. 14 Nastavení spuštěné aplikace
Zdroj: Vlastní tvorba

```
public static void main(String[] args) {
  Main app = new Main();

  AppSettings settings = new AppSettings(true);
  settings.setResolution(640,480);
  // Ostatní možnosti nastavení

  app.setSettings(settings);
  app.start();
}
```

Obr. 15 Vlastní výchozí nastavení
Zdroj: Vlastní tvorba

6.2 Stavy aplikace

Stavy aplikace z anglického Application States jsou rozhraní AppState v JMonkeyEnginu umožňující přidávat do hry novou logiku a mechanismy. Každá aplikace v JMonkeyEnginu obsahuje právě jeden AppStateManager, dále jen stateManager, který je inicializován ve třídě LegacyApplication. StateManager obsahuje list všech AppState, které jsou k aplikaci momentálně připojeny a má 4 hlavní metody attach, detach, hasState a getState. Metody attach a detach slouží k připojování a odpojování jednotlivých instancí tříd, implementujících rozhraní AppState. Metoda hasState vrací boolean a informuje o připojení instance do stateManagera. Metoda getState(MyState.class) vrací instanci dané třídy připojené do stateManagera, pomocí které může být zavolána metoda konkrétního stavu. Časté využívaní metody getState(MyState.class) k vyvolávání metod navráceného stavu, signalizuje špatné navržení struktury aplikace.

„AppStates jsou velmi užitečné pro změnu nebo pozastavení / odpojení celých sad jiných AppStates. Například InGameState (načte herní GUI, aktivuje mapování, inicializuje herní obsah a spustí herní smyčku) versus MainScreenState (zastaví herní smyčku, uloží obsah hry, přepne na menu GUI a upraví mapování vstupů)“ [7]

Rozhraní AppState je v JMonkeyEnginu implementováno ve dvou třídách, starší AbstractAppState a novější BaseAppState. BaseAppState oproti AbstractAppState nemusí na začátku každé převzaté (override) metody volat funkci super() a navíc obsahuje metody, které jsou volány při pozastavení a opětovném povolení daného stavu, ve kterým je možné provádět vlastní kód. Využitím AppState je možné docílit odlišného chování aplikace v různých fázích a s minimální zátěží na

procesor. Na obrázku č. 16 je vidět základní struktura třídy, která rozšiřuje abstraktní třídu BaseAppState.

```
public class MyState extends BaseAppState {
    @Override
    protected void initialize(Application app) {
        //Blok který se provede při připojení stavu do stateManageru.
        //stateManager.attach(this)
    }

    @Override
    protected void cleanup(Application app) {
        //Blok který se provede při odpojení stavu ze stateManageru
        //stateManager.detach(this)
    @Override
    protected void onEnable() {
        //Blok který se provede při povolení stavu
        //this.setEnabled(true)
    }

    @Override
    protected void onDisable() {
        //Blok který se provede při pozastavení stavu
        //this.setEnabled(false)
    }

    @Override
    public void update(float tpf) {
        //(volitelný) stejný jako simpleUpdate ve třídě Main.java
    }
}
```

Obr. 16 Struktura AppState třídy
Zdroj: Vlastní tvorba

Využití AppState je vhodné pro rozdělení jednotlivých bloků funkcí do různých tříd, například přepínání jednotlivých editačních nástrojů terénu nebo využívat jiné mapování tlačítek ve hře a v menu. V jednotlivých AppState se nachází přístup ke všem proměnným centrální třídy Main, viz. obrázek č. 17. Na obrázku je zobrazena práce se stavy aplikace. Při inicializaci je inicializován další stav, který bude odpojen společně se svým rodičem pomocí metody cleanup. U metod onEnable a onDisable je ukázáno přidávání a odstraňování mapování tlačítek při pozastavování a spouštění tohoto stavu a zároveň pozastavení stavu potomka. Mapování není třeba přidávat v inicializaci a odstraňovat při čištění, protože metoda onEnable je volána okamžitě po inicializaci a metoda onDisable je volána před čištěním stavu. Obsah metod intialize a cleanup je možné vložit pouze do metod onEnable a onDisable. Funkcionalita bude v určitých případech stejná, ale není to správné použití třídy.

```
public class MyState extends BaseAppState {
    private Main _app;
    private AppStateManager _stateManager;
    private InputManager _inputManager;
    private TerrainState _terrain;

    @Override
    public void initialize(final Application app)
    {
        _app = (Main) app;
        _stateManager = _app.getStateManager();
        _inputManager = _app.getInputManager();

        _terrain = new TerrainState();
        _stateManager.attach(_terrain);
    }

    @Override
    protected void cleanup(Application app) {
        _stateManager.detach(_terrain);
    }

    @Override
    protected void onEnable() {
        _inputManager.addMapping(...kód); //přidaní mapování při povolení
        _terrain.setEnabled(true);
    }

    @Override
    protected void onDisable() {
        _inputManager.removeMapping(...kód); //odstranění mapování při pauze
        _terrain.setEnabled(false);
    }
}
```

Obr. 17 Inicializace AppState
Zdroj: Vlastní tvorba

Využívat AppState pro řízení jednotlivých Spatials ve scéně není vhodné, jelikož AppState jsou navrženy pro řízení herní logiky a komplexních herních mechanismů. Pro definování chování jednotlivých Spatials slouží rozhraní com.jme3.scene.control.Control, které se k jednotlivým Spatial přidávají pomocí spatial.addControl(myControl). Třída implementující rozhraní Control obsahuje metodu update, která je stejná jako metoda centrální třídy Main simpleUpdate. Každá instance třídy Spatial může mít několik tříd Control najednou, to umožňuje vytvářet komplexní umělou inteligenci pro NPC se zachováním přehlednosti kódu díky rozdělení jednotlivých částí chování do několika tříd. S využitím návrhového vzoru kompozice je možné mít jednu třídu Control, která pomocí metody setEnabled(boolean), stejně jako u AppState zapíná či vypíná jednotlivé vzory chování podle potřeby.

6.3 Herní assety

Multimediální soubory jako jsou modely, materiály, textury, scény, shadery, zvukové soubory a fonty se ve hrách nazývají assety. JMonkeyEnginu obsahuje pro správu a optimalizaci assetů na různých platformách AssetManager. Výchozí složka, do které se při vývoji vkládají assety je MojeHra/assets a načítají se pomocí relativní cesty. Při zabalování hry do kompilovaných souborů (build) je složka asset zabalena do souboru asset.jar, pomocí tohoto zabalení se cesty k jednotlivým assetům definované v kódu stávají platformě nezávislé. JMonkeyEngine obsahuje výchozí assety, které obsahují základní materiály, materiálové definice, shadery a textury. Výchozí assety se načítají z relativní cesty začínající slovem Common a nacházejí se zabalené v jar souborech knihovny JMonkeyEnginu.

Načítaní assetů skrze AssetManager optimalizuje běh aplikace použitím cache uložené v instanci AssetManagera, takže jednou načtené assety se nenačítají znovu, pokud jsou nahrané v paměti. V tabulce č. 5 je vyobrazena struktura souborů ve složce assets. Struktura souborů není závislá a může být libovolně změněna. Formáty souborů, jež jsou uvedeny v tabulce, jsou při zabalování hry zahrnuty do assets.jar.

Cesta:	Formát souborů:
MojeHra/assets/Interface/	font, jpg, png, xml
MojeHra/assets/MatDefs/	j3md
MojeHra/assets/Materials/	j3m
MojeHra/assets/Models/	j3o
MojeHra/assets/Scenes/	j3o
MojeHra/assets/Shaders/	j3f, vert, frag
MojeHra/assets/Sounds/	ogg, wav
MojeHra/assets/Textures/	jpg, png, tga

Tab. 5 Složka assets
Zdroj: [7]

Assety nemusí být nutně načítány pouze ze složky assets, ale je možno zaregistrovat novou lokaci, ze které mohou být načteny. Na obrázku č. 18 je uvedeno načítání assetů z lokálního uložiště pomocí ZipLocatoru umožňující přístup k lokálním souborům zip nebo HttpZipLocatoru k zip souborům uložených v síti. K těmto souborům se přistupuje stejným způsobem jako k assetů ve složce assets.

```
//načtení z lokálního uložiště
assetManager.registerLocator("town.zip", ZipLocator.class);
Spatial scene = assetManager.loadModel("main.scene");
rootNode.attachChild(scene);

//načtení z Internetu
assetManager.registerLocator("https://storage.googleapis.com/"
    + "google-code-archive-downloads/v2/code.google.com/"
    + "jmonkeyengine/wildhouse.zip", HttpZipLocator.class);
Spatial scene = assetManager.loadModel("main.scene");
rootNode.attachChild(scene);
```

Obr. 18 Načítání assetů
Zdroj: Vlastní tvorba

Soubory, které nejsou vypsané v tabulce č. 5 nebudou automaticky zabalené do asset.jar pomocí zabalovacího scriptu, proto např. soubor blend je nutno pomocí SDK konvertovat do binárního souboru j3o. *„Pokud z různých důvodů nepoužíváte SDK, ale jiný vývojový nástroj, můžete přesto převádět modely do j3o formátu. Načtete model pomocí AssetManager.loadModel() jako Spatial. Potom uložte Spatial jako j3o pomocí třídy BinaryExporter“* [7]

Všechny JMonkeyEngine třídy implementující rozhraní com.jme3.export.Savable mohou být uloženy jako binární data. Vlastní třídy mohou být uloženy také pokud implementují rozhraní Savable, tak jako je zobrazena ukázková struktura na obrázku č. 19. Metoda read u třídy InputCapsule přijímá parametry (název proměnné, výchozí hodnota) a vrací hodnotu. Metoda write u třídy OutputCapsule přijímá parametry (hodnotu, název proměnné, výchozí hodnota).

```
public class MySavableClass implements Savable {
    private int      intValue;
    private float    floatValue;
    private Material jmeObject;

    public void write(JmeExporter ex) throws IOException {
        OutputCapsule capsule = ex.getCapsule(this);
        capsule.write(intValue,   "intValue",   1);
        capsule.write(floatValue, "floatValue", 0f);
        capsule.write(jmeObject,  "jmeObject",  new Material());
    }

    public void read(JmeImporter im) throws IOException {
        InputCapsule capsule = im.getCapsule(this);
        intValue   = capsule.readInt(    "intValue",   1);
        floatValue = capsule.readFloat(  "floatValue", 0f);
        jmeObject  = capsule.readSavable("jmeObject",  new Material());
    }
}
```

Obr. 19 Struktura vlastní Savable třídy
Zdroj: Vlastní tvorba

Pomocí třídy com.jme3.export.binary.BinaryExporter mohou být třídy převedeny do binárních dat a následně pomocí com.jme3.export.binary.BinaryImporter mohou být tato data načtena. JMonkeyEngine obsahuje také starší způsob ukládání dat pomocí com.jme3.export.xml.XMLExporter a com.jme3.export.xml.XMLImporter mohou být data také uložena a načtena, ale tento způsob byl nahrazen, protože pracuje pomalu při běhu aplikace.

6.4 Aktualizační smyčka

Všechno, co se v aplikaci děje je prováděno během inicializace anebo vyvoláno z aktualizační smyčky. Aktualizační smyčka aplikace běží od spuštění aplikace až po její ukončení. U centrální třídy je aktualizační metoda nazvána simpleUpdate a u jednotlivých stavů je nazvána update. Každý aktualizační cyklus trvá určitý čas podle náročnosti kódu na procesorový čas. Proměnná vstupující do každé aktualizační metody, je parametr typu float označovaný jako Time per frame zkráceně tpf, česky čas pro snímek dále jen tpf. Tpf označuje kolik milisekund uplynulo od posledního updatu. Pro zajištění plynulého běhu aplikace je tento parametr podstatný. Aplikace by tedy na pomalejších zařízení běžela pomaleji a na rychlejších rychleji. V aktualizační smyčce se parametrem tpf násobí všechny proměnné udávající hodnotu vzdálenosti, síly a dalších, aby v aplikaci byla změna za sekundu vždy stejná. Např. pohyb myši po obrazovce by měl proměnlivou rychlost závislou na vykreslovací frekvenci aplikace, pokud by parametr tpf nebyl použit. Na obrázku č. 20 je znázorněna rotace geometrie box s využitím tpf.

```
@Override
public void simpleUpdate(float tpf) {
    box.rotate(0, 10*tpf, 0);
}
```

Obr. 20 Využití aktualizační smyčky
Zdroj: Vlastní tvorba

6.5 Kamera

Výchozí zobrazovací zařízení v grafu scény je kamera třídy com.jme3.renderer.Camera s definovanou výškou a šířkou odpovídající rozměrům okna aplikace. Scéna je vykreslována perspektivně s pozorovacím úhlem 45° a

dynamickým poměrem stran. Vykreslovány jsou Spatials vzdálené od 1wu do 1000wu. Výchozí umístění je Vector3f (0, 0, 10) a je orientována ve směru, který je definován jako opačný k jednotkovému vektoru Z = Vector3f (0, 0, -1). Přidáním stavu DebugKeysAppState získáme užitečný nástroj pro ladění kamery. Zmáčknutím tlačítka C je do konzole vývojového prostředí vypsána informace o pozici, rotaci a směru kamery viz. obrázek č. 21.

```
Camera Position: (-31.244484, 129.11, -21.01324)
Camera Rotation: (0.004087, 0.98713, -0.15797317, 0.025)
Camera Direction: (0.049151033, -0.31208092, -0.9487833)
//cam.setLocation(new Vector3f(-31.244484f, 129.11f, -21.01324f));
//cam.setRotation(new Quaternion(0.004087, 0.98713, -0.15797317, 0.025));
```

Obr. 21 Informace o transformací kamery
Zdroj: Vlastní tvorba

Třída Camera neumožnuje žádný pohyb ani ovládání, proto ji rozšiřují dvě třídy com.jme3.input.FlyByCamera a com.jme3.input.ChaseCamera. Výchozí konstruktor třídy SimpleApplication inicializuje stav FlyCamAppState využívající se při pohledu první osoby, který třídě Camera přidává možnosti ovládaní skrze tlačítka WASD nebo směrových šipek a rotaci skrze pohybu myši. Pomocí metody setDragToRotate je zviditelněna myš na obrazovce a pro rotaci je využita jen pokud je držené pravé tlačítko myši. Třídou ChaseCamera je vytvořeno chování kamery z pohledu třetí osoby, která následuje pohybující se Spatial. Pravé tlačítko myši je zde využito k rotování kamery kolem následovaného Spatial objektu. Při vykreslování naprogramovaných cutscén (mini film) je ChaseCamera využita k vytvoření hladkého pohybu kamery kolem sledovaného Spatial. Inicializace třídy ChaseCamera je vyobrazena na obrázku č. 22, před inicializováním musí být vypnut stav FlyByCamera, jinak by se o pohyb kamery staraly dvě třídy, což by vedlo k nežádoucímu chování.

```
_app.getFlyByCamera().setEnabled(false);
ChaseCamera chaseCam = new ChaseCamera(_app.getCamera(), spatial, _app.getInputManager());
```

Obr. 22 Inicializace třídy ChaseCamera
Zdroj: Vlastní tvorba

6.6 Uživatelské vstupy

JMonkeyEngine umožnujeme ovládání aplikace pomocí různých vstupních zařízení (myš, klávesnice nebo joystick). Třída com.jme3.input.InputManager je

inicializována ve třídě LegacyApplication a stará se o zpracování vstupů od uživatele. Akce vyvolané interakcí se vstupním zařízením jsou implementovány v listerenu, česky naslouchači. Každá akce má vlastní událost, která jí spouští, zvanou jako trigger, česky spouštěč. JMonkeyEngine obsahuje spouštěče pro zmáčknutí tlačítka KeyTrigger na klávesnice, MouseButtonTrigger na myši a JoyButtonTrigger na ovladači. InputManager také umí odposlouchávat pohyb myši pomocí MouseAxisTrigger a ovladače pomocí JoyAxisTrigger. Každý spouštěč přijímá parametr typu int, udávající id konkrétního tlačítka nebo osy. Třídy MouseInput, KeyInput a JoyInput obsahují konstanty jednotlivých dostupných tlačítek a os. Všechny výše zmíněné třídy se nacházejí v balíčku com.jme3.input.controls.

Listenery jsou rozděleny do dvou typů ActionListener pro okamžité akce a AnalogListener pro plynulé akce. *„Oba vstupní listenery neví, jaký konkrétní tlačítko bylo zmáčknuto, vědí pouze název vyvolané akce.“* [7] ActionListener je používán k odposlouchávání zmačknutých nebo uvolněných tlačítek přijímající parametry jméno akce, boolean, zda bylo tlačítko zmáčknuto nebo uvolněno a float Time per frame. AnalogListener je používán k odposlouchávání tahů myší a úhlů páček ovladače. Přijímá parametry jméno akce, float určující sílu a float Time per frame. *„MouseAxis a JoyAxis se spouštění podél osy X (vpravo/vlevo) nebo osy Y (nahoru/dolů). Tyto spouštěče obsahují extra parametrem typu boolean pro zápornou polovinu osy, Napsaná hodnota (true) je pro kladnou osu a (false) pro zápornou.“* [7]

Na obrázku č. 23 je znázorněná implementace AnalogListeneru a ActionListeneru, přiřazení mapovaní při inicializaci vlastního stavu a odstranění při čištění stavu. Pokud uživatel zmáčkne tlačítko mezerník na objektu Spatial je vyvolána metoda jump a při pohybu myši po ose x do kladných čísel je vyvolána relativní rotace objektu Spatial. Pokud je AnalogListener využit pro odposlouchávání zmáčknutých tlačítek, zjistíme tak čas, po jakou bylo tlačítko drženo.

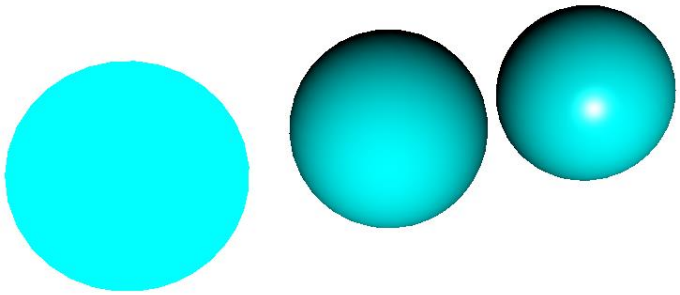
```java
public class MyState extends BaseAppState
{
    private static final String JUMP = "moveJump";
    private static final String ROTATE_XPLUS = "spatialRotate";
    private InputManager inputManager;
    @Override
    protected void initialize(Application app){
        inputManager = app.getInputManager();
        inputManager.addMapping(JUMP, new KeyTrigger(KeyInput.KEY_SPACE));
        inputManager.addMapping(ROTATE_XPLUS, new MouseAxisTrigger(MouseInput.AXIS_X, true));
        inputManager.addListener(actionListener, JUMP);
        inputManager.addListener(analogListener, ROTATE_XPLUS);
    }
    @Override
    protected void cleanup(Application app){
        inputManager.deleteMapping(JUMP);
        inputManager.deleteMapping(ROTATE_XPLUS);
        inputManager.removeListener(actionListener);
        inputManager.removeListener(analogListener);
    }
    private final ActionListener actionListener = new ActionListener(){
        @Override
        public void onAction(String name, boolean isPressed, float tpf)
        {
            if(name.equals(JUMP) && isPressed){
                //akce vyvolána jednou při zmáčknutí tlačítka
                spatial.jump();
            }
        }
    };
    private final AnalogListener analogListener = new AnalogListener()
    {
        @Override
        public void onAnalog(String name, float value, float tpf)
        {
            if(name.equals(ROTATE_XPLUS)){
                //akce při pohybu myší na ose X do kladných čísel
                spatial.rotate(0f, value, 0f);
            }
        }

    };
}
```

Obr. 23 Implementace ActionListener a AnalogListener
Zdroj: Vlastní tvorba

6.7 Materiály

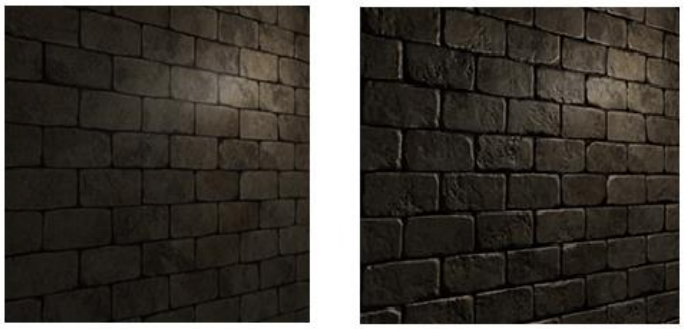
Graf scény by bez definovaných materiálů přidaných ke každé geometrii byl prázdný, protože OpenGL by nemělo informace o tom jak danou geometrii vykreslit. Všechny uložené materiály mají koncovku j3m a materiálové definice j3md. Všechny výchozí materiálové definice jsou uloženy v relativní cestě Common/MatDefs/Misc/ a musí u nich být nastavena barva nebo textura pro vykreslení ve scéně. Pro vykreslení modelu, který nereaguje na světla scény a nemá žádné zastínění se v JMonkeyEnginu používá materiálová definice Unshaded.j3md. Geometrie reagující na světlo jsou vytvořeny pomocí materiálové definice Lighting.j3md, tyto materiály

potřebují ve scéně světelný zdroj jinak nejsou viditelné. Tomuto materiálu může být nastavena difuzní textura, normálová mapa, světelná mapa a průhlednost. Na obrázku č. 24 jsou zobrazeny 3 geometrie s použitými předdefinovanými materiály, zleva Unshaded, Lighting a Lighting s nastavenou odrazivostí světla pomocí parametru Shininess.

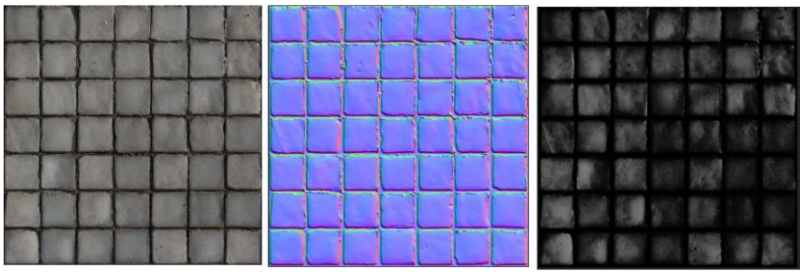


Obr. 24 Porovnání materiálů v JMonkeyEnginu
Zdroj: [vlastni]

Normálovou a světelnou mapu je možné vygenerovat např. pomocí SDK nebo programu Blender. Normálová mapa vytváří iluzi nerovnoměrného povrchu geometrie, pomocí rozdílných normálových vektorů, které ovlivňují lom světla a zastínění pro každý pixel. Světelná mapa definuje množství odraženého světla pro každý pixel v závislosti na normálovém vektoru. Na obrázku č. 25 je vidět rozdíl na geometrie bez normálové mapy (vlevo) a s normálovou mapou (vpravo). Na obrázku č.26 je zobrazena difúzní textura, normálová a světelná mapa, které jsou v OpenGL využity k vytvoření realisticky vypadajícího povrchu.

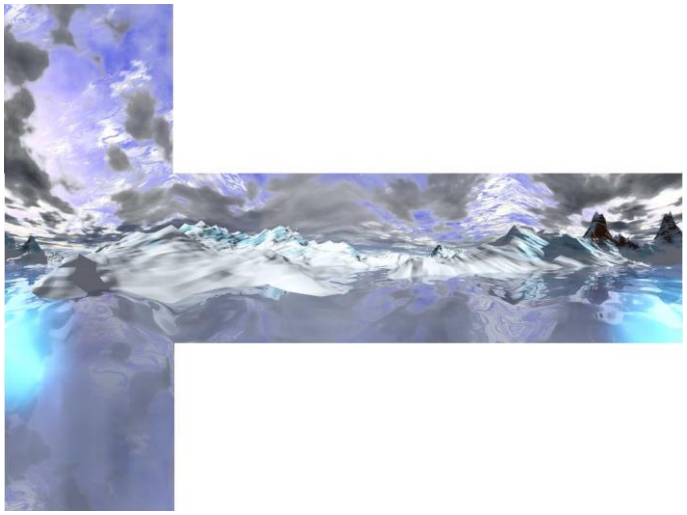


Obr. 25 Material s a bez normálové mapy
Zdroj: [15]



Obr. 26 Difúzní textura, normálová a světelná mapa
Zdroj: Vlastní tvorba

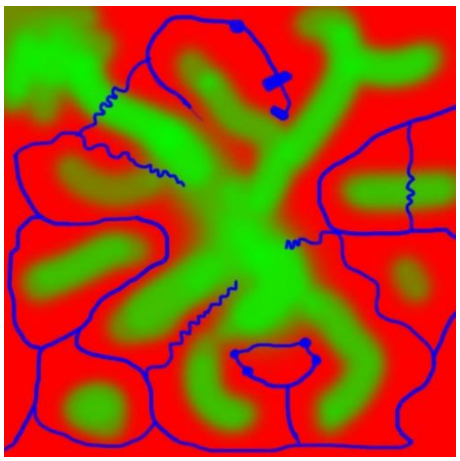
JMonkeyEngine obsahuje ještě další dvě základní definice materiálů, Sky.j3md a Terrain.j3md/TerrainLighting.j3md. Sky.j3md je materiál, kterému je do parametru TextureCubeMap nastavena textura oblohy. Ukázková textura viz. obrázek č. 27.



Obr. 27 SkyBox textura
Zdroj: [7]

Obloha, která je v herních enginech nazvána SkyBox, je Spatial připojený ke kameře. Pozice je vždy stejná jako pozice kamery, ale rotace je statická a vytváří kolem kamery krychli. SkyBox musí mít nastavený vlastnosti setQueueBucket(Bucket.Sky) a setCullHint(CullHint.Never). Nastavení CullHint.Never způsobí, že se jeho polygony budou vykreslovat vždy a nikdy nebudou skryty z důvodu optimalizace běhu aplikace a Bucket.Sky informuje OpenGL, že Spatial má mít v Z-Bufferu nastavenou hodnotu na nekonečno, takže SkyBox bude vykreslený až za všema ostatníma Spatials. Terrain.j3md umožnuje definovat materiál s až 11 texturami/mapami v libovolné kombinaci difuzních textur a normálových map. Tento materiál se používá primárně pro terén. Textury jsou vykresleny na základě

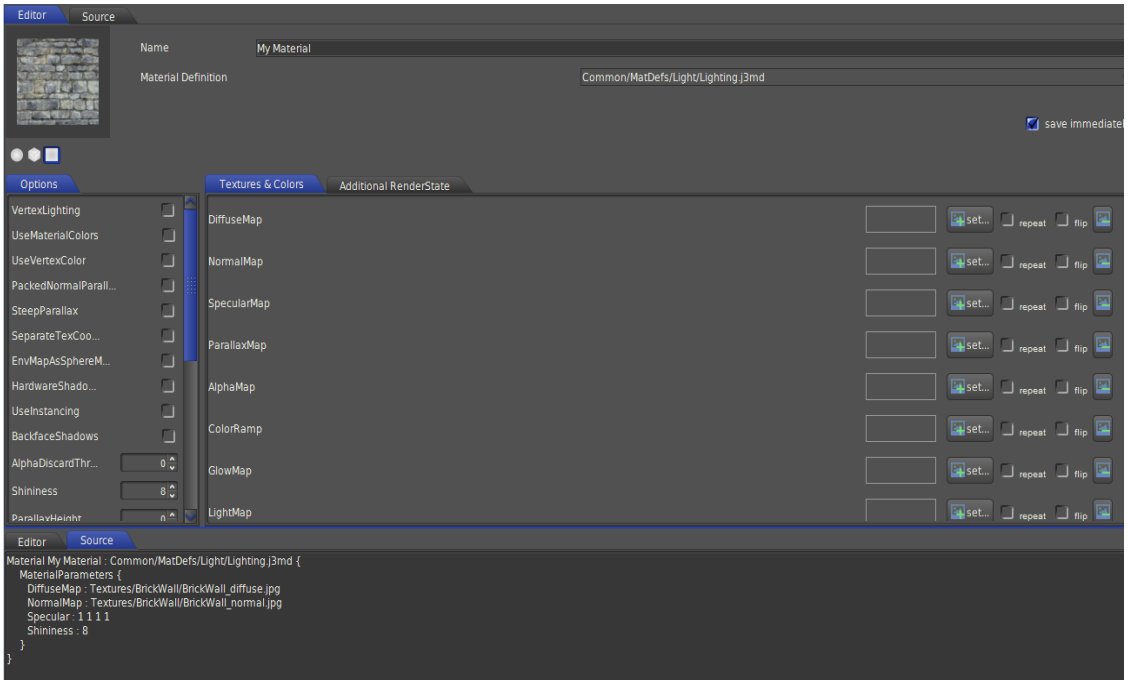
až třech alpha map, které definují jejich rozložení na terénu. Texturování tak velké geometrie, jako je terén, je použitím textur s opakujícím se vzorem výpočetně podobně náročné jako menší geometrie. Na obrázku č. 27 je zobrazena alpha mapa, pomocí které budou rozděleny textury na terén. Každý kanál RGBA označuje jednu texturu a díky tomu je možný plynulí přechod mezi texturami. Definice materiálu TerrainLighting.j3md je stejná jako Terrain.j3md, ale přebírá navíc vlastnosti Lighting.j3md materiálu. Jednotlivé materiály je možné upravovat programově, viz obrázek č. 28, anebo vytvořením vlastní materiálové definice pomocí grafického rozhraní v SDK nebo jazyka OpenGL Shading Language viz obrázek č. 29.



Obr. 28 Alpha mapa terénu
Zdroj: [7]

```
Geometry sphare1 = new Geometry("normal sphere", new Sphere(32, 32, 1f));
Material mat1 = new Material(assetManager,"Common/MatDefs/Light/Lighting.j3md");
mat1.setFloat("Shininess", 0f); // [1,128]
mat1.setBoolean("UseMaterialColors", true);
mat1.setColor("Ambient", ColorRGBA.Black);
mat1.setColor("Diffuse", ColorRGBA.Cyan);
mat1.setColor("Specular", ColorRGBA.White);
sphare1.setMaterial(mat1);
rootNode.attachChild(sphare1);

Geometry sphare2 = new Geometry("Smooth sphere", new Sphere(32, 32, 1f));
Material mat2 = new Material(assetManager,"Common/MatDefs/Light/Lighting.j3md");
mat2.setBoolean("UseMaterialColors", true);
mat2.setColor("Ambient", ColorRGBA.Black);
mat2.setColor("Diffuse", ColorRGBA.Cyan);
mat2.setColor("Specular", ColorRGBA.White);
mat2.setFloat("Shininess", 100f); // [1,128]
sphare2.setMaterial(mat2);
sphare2.move(2.5f, 0, 0);
rootNode.attachChild(sphare2);
```

Obr. 29 Programová úprava materiálu
Zdroj: Vlastní tvorba



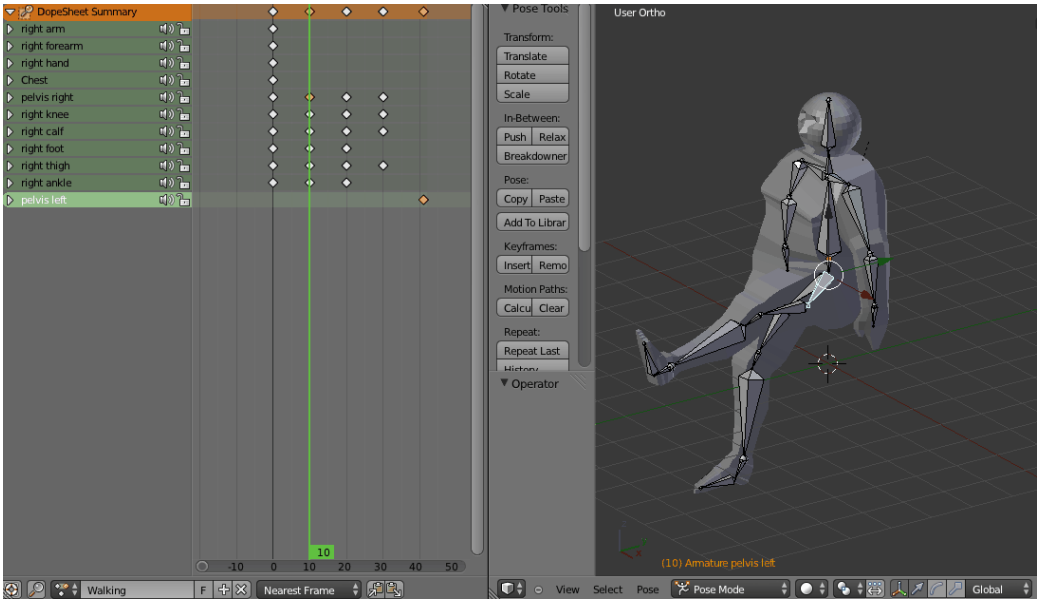
Obr. 30 Vlastní tvorba materiálu pomocí SDK
Zdroj: Vlastní tvorba

6.8 Animace a kostra modelu

Ve 3D hrách se využívá kosterní animace k rozpohybování primárně postav, ale je možné použít tento princip k rozpohybování jakékoliv 3D modelu, například pohyb větví stromů. Vytvoření animací je velmi náročná část při vývoji her, jelikož výsledné animace mohou působit jako robotický pohyb nebo postava vypadá při chůzi jako když plave. Proto velká studia vyvíjející AAA tituly používají k vytvoření animací metodu Motion Capture. Objekt je sledovaný pomocí několika kamer a má na sobě senzory. Poloha těchto senzorů je vzorkována několikrát za sekundu, jedná se ale o velmi komplexní a drahou technologii. Animace pro Indie tituly využívají pouze animace vytvořené pomocí nástroje pro práci s 3D modely jako je Blender.

Animace se skládá ze tří základních prvků, které musí být definované (kostra kůže, klíčové snímky). Kostra modelů se skládá z jednotlivých kostí. Definované kosti tvoří hierarchii rodič-potomek, takže pohyb jedné kosti ovlivňuje pohyb kosti druhé (např. pohyb paže ovlivňuje polohu zápěstí). Kůže je polygon (viditelná část), jehož pohyb a tvar je ovlivněný pohybem kostí. Jeden polygon může být ovlivněn pohybem více kostí (neviditelná část). Každý polygon kůže může být ovlivněný různě pohybem jedné kosti. (např. pohyb ruky ovlivní všechny polygony na ruce,

mírně ovlivní polygony na hrudi, ale nohy neovlivní vůbec). Klíčový snímek definuje polohu kostí. Jedna animace je tvořena sekvencí klíčových snímků. Na obrázku č. 31 je vlevo zobrazený seznam kostí (zelené řádky), klíčové snímky (bílé tečky) a vpravo model s definovanými kostmi, kde hlavní kost začíná v bederní oblasti zad a všechny ostatní kosti jsou potomci této hlavní kosti. Engine pro přehrávání animací se následně stará o plynulí přechod kostí do polohy definovaných v klíčových snímcích. JMonkeyEngine využívá GPU k výpočtu pohybu kůže.



Obr. 31 Animace klíčové snímky
Zdroj: [1]

JMonkeyEngine poskytuje pro správu animací dvě základní třídy a jedno rozhraní (AnimControl, AnimChannel a rozhraní AnimEventListener), pomocí kterých je možné používat animace. Pokud byly v načteném modelu nalezeny správně vytvořené animace, dostaneme instanci třídy com.jme3.animation .AnimControl zavoláním metody Spatial.getControl(AnimControl.class). Správně vytvořené animace je možné zkontrolovat i pomocí Scene Exploreru, ve kterém jsou vidět názvy animací s možností jejich přehrání a jednotlivé kosti kostry, viz. obrázek č. 32. Pokud není použité JMonkeyEngine SDK, je možné získat seznam animací zavoláním metody AnimControl.getAnimationNames(), která vrací kolekci Stringů s názvy dostupných animací.

Obr. 32 Animace ve Scene Exploreru
Zdroj: Vlastní tvorba

Přehrávání animací je možné pomocí třídy com.jme3.animation.AnimChannel. Instanci této třídy získáme zavoláním metody AnimControl.createChannel(). Jeden model může mít více animačních kanálů umožňující přehrávat několik animací zároveň. Model uvedený na obrázku č. 32 obsahuje animace ovládající celou kostru modelu, proto stačí vytvořit pouze jeden animační kanál, kterému bude měněna přehrávaná animace. Rozhraní com.jme3.animation.AnimEventListener slouží k notifikaci při dokončení nebo změně animace. Obsahuje metody onAnimCycleDone a onAnimChange, ve kterých dostaneme jako parametr instanci třídy AnimControl a AnimChannel a název dokončené animace. Animaci vyvoláme metodou AnimChannel.setAnim přijímající parametr String s názvem animace a druhý volitelný parametr typu float určující, jak dlouho se nová animace bude překrývat se starou animací na stejným kanále pro vytvoření plynulého přechodu. Animaci je možné nastavit smyčku, ve které bude animace přehrávaná pomocí metody AnimChannel.setLoopMode. V ukázce na obrázku č. 33 je zobrazena implementace přehrávání dvou animací znázorňující nečinnost v náhodném pořadí.

```
public class MyAnimControl implements AnimEventListener
{
    private final Spatial _model;
    private AnimChannel _channel;
    private AnimControl _control;

    public MyAnimControl(Spatial model)
    {
        _control = model.getControl(AnimControl.class);
        if (_control != null)
        {
            _control.addListener(this);
            _channel = control.createChannel();
            _channel.setAnim("IdleA");
        }
    }

    @Override
    public void onAnimCycleDone(AnimControl control, AnimChannel channel, String animName)
    {
        if(new Random().nextBoolean())
        {
            _channel.setAnim("IdleA");
        }else
        {
            _channel.setAnim("IdleB");
        }
    }
    @Override
    public void onAnimChange(AnimControl control, AnimChannel channel, String animName){}
}
```

Obr. 33 Implementace animace
Zdroj: Vlastní tvorba

V RPG hrách, ale i jiných žánrech je potřeba přidávat na animované postavy jiné geometrie znázorňující zbraň, zbroj a další. Pokud by přidaná geometrie znázorňující např. klobouk by byla umístěna relativně od polohy modelu, tak při přehrávání animace např. běhu by byla stále na stejném místě. Tento problém řeší třída com.jme3.animation.SkeletonControl umožňující získat jednotlivé kosti z kostry modelu, ke kterým je možné přidat Spatials, takže geometrie klobouku bude kopírovat pohyb hlavy při animaci běhu. Na obrázku č. 34 je ukázáno, jak přidat klobouk ke kosti hlava („headBone“), aby kopíroval pohyby animované postavy.

```
public void equipHat(Spatial model)
{
    SkeletonControl skeletonControl = model.getControl(SkeletonControl.class);
    if(skeletonControl != null)
    {
        Spatial hat = assetManager.loadModel("Models/attachment/Hat.j3o");
        skeletonControl.getAttachmentsNode("headBode").attachChild(hat);
    }
}
```

Obr. 34 SkeletonControl ukázka
Zdroj: Vlastní tvorba

6.9 Ray Casting

JMonkeyEngine využívá k interakci techniku vyslání paprsku z anglického Ray casting (nejedná se o Ray tracing). Paprsek třídy com.jme3.math.Ray je vyslaný z bodu A ve směru B s limitovanou vzdáleností. V JMonkeyEnginu obsahují všechny geometrie metodu collideWith, která přijímá jako parametr instance třídy Ray a com.jme3.collision.CollisionResults. Metoda collideWith zjistí kolize paprsku s geometrií a všemi jeho potomky. Např. pokud je cílem zjistit kolizi paprsku se všemi objekty ve scéně, metodu collideWith zavoláme nad uzlem rootNode. Třída CollisionResults implementuje rozhraní java.lang.Iterable<T extends Object>, takže nad instancí CollisionResults může být zavolán foreach cyklus a vracející instance třídy com.jme3.collision.CollisionResult znázorňující jednotlivé kolize paprsku s geometriemi. Třída CollisionResults obsahuje metody k navrácení nejbližší kolize getClosestCollision vracející CollisionResult na indexu 0, nejvzdálenější kolize getFarthestCollision vracející CollisionResult na indexu početKolizí-1 nebo getCollision(int index) vracející kolizi na předaném indexu. Všechny tyto metody před vrácením CollisionResult srovnávají výsledky podle vzdálenosti, pokud už nebyly dříve seřazeny. Třída CollisionResult obsahuje tři základní metody getGeometry pro získání geometrie, getDistance pro získání vzdálenosti od počátku paprsku a getContactPoint pro získání bodu průniku (Vector3f) od počátků scény uzlu (rootNode).

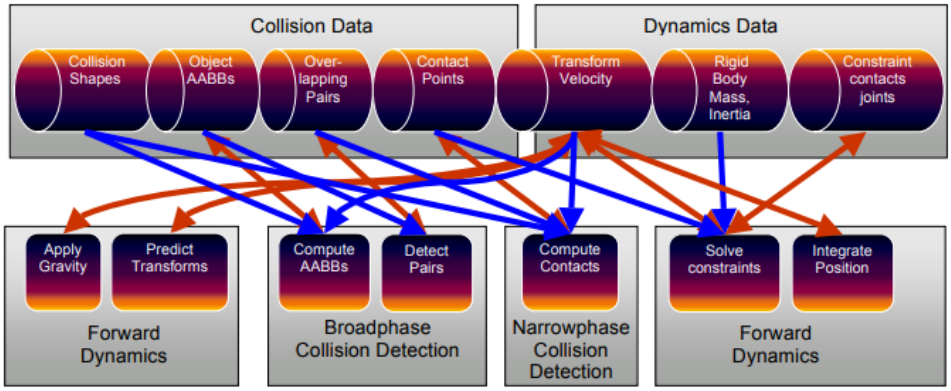
Na obrázku č. 35 jsou zobrazeny dvě ukázky implementace Ray castingu. V první ukázce je testování průniku paprsku vyslaného ze středu okna vykreslující scénu a následné vypsání všech průniků. Ve druhé ukázce je testování průniku paprsku od souřadnic myši do scény ve směru kamery. Pomocí instance třídy inputManager získáme 2D souřadnice polohy myši metodou getWorldCoordinates, kde jako druhý parametr je použita Z-tová souřadnice. Odečtením souřadnice se Z-tovou hodnotou 0 od souřadnice se Z-tovou hodnotou 1 a jejím normalizováním získáme směr, kterým bude paprsek vyslán.

```
//1. ukázka interakce středu obrazovky se scénou
Ray ray = new Ray(camera.getLocation(), camera.getDirection());
CollisionResults results = new CollisionResults();

rootNode.collideWith(ray, results);
for (CollisionResult result : results)
{
    System.out.println("Contact: " + result.getContactPoint()
        + " Geometry Name: " + result.getGeometry().getName()
        + " Distance: " + result.getDistance());
}

//2. ukázka interakce kurzoru se scénou
Vector2f cursorLoc = inputManager.getCursorPosition()
Vector3f rayStartLoc = camera.getWorldCoordinates(cursorLoc, 0);
Vector3f mouseFarLoc = camera.getWorldCoordinates(cursorLoc, 1);
Vector3f rayDirection = mouseFarLoc.subtractLocal(rayStartLoc).normalizeLocal();

Ray ray = new Ray(rayStartLoc, rayDirection);
CollisionResults results = new CollisionResults();

rootNode.collideWith(ray, results);
for (CollisionResult result : results)
{
    System.out.println("Contact: " + result.getContactPoint()
        + " Geometry Name: " + result.getGeometry().getName()
        + " Distance: " + result.getDistance());
}
```

Obr. 35 Ray Casting implementace
Zdroj: Vlastní tvorba

6.10 Fyzika a kolize

JMonkeyEngine využívá pro simulaci fyziky mírně přizpůsobenou knihovnu JBullet, což je Java verze open source knihovny Bullet Physics SDK distribuované pod licencí ZLib napsané v jazyce C++ optimalizované pro využití ve VR, video hrách, visuálních efektech a simulaci robotů. Nabízí *„diskrétní a spojitou detekci kolizí včetně paprskového a konvexního testu. Podpora konvexních a konkávních sítí včetně všech základních tvarů. Rychlé a stabilní řešení fyziky pevných těles, vozidel, postav a ragdoll těles“*. [3] Ragdoll postava jindy označovaná jako hadrový panáček, je simulací pohybu postavy (např. při pádu), která je definovaná pomocí kostí a vymezeného úhlu pohybu jednotlivých kostí. Ragdoll mechaniku je možné využít i u jiných těles než jen postav, např. tlusté lano, bude mít omezené uhly tak, aby nebylo možné na něm vytvořit ostrý uhel ohybu při pohybu, který by působil nereálně. Na obrázku č. 36 je zobrazena pipeline sloužící k simulaci fyziky u JBullet knihovny



Obr. 36 Pipeline knihovny JBullet
Zdroj: [3]

Fyzika se inicializuje pomocí stavu, který musí být připojen ke stateManageru aplikace. Stav com.jme3.bullet.BulletAppState je součástí JMonkeyEnginu. Po inicializovaní a přidání stavu stačí vytvořit ke každé geometrii kolizní model (popis níže). Následně je vytvořena instance třídy com.jme3.bullet.control .RigidBodyControl přijímající v konstruktoru kolizní model a float parametr definující hmotnost geometrie. Pokud je geometrie statická, je jí nastavena hmotnost 0. Instance třídy RigidBodyControl je následně předána geometrii pomocí metody addControl, což umožní stavu, obsluhující fyziku manipulovat s geometrií. Posledním krokem je geometrii zaregistrovat do fyzického prostoru stavu pomocí metody BulletAppState.getPhysicsSpace().add přijímající jako parametr geometrii. Spatials v JMonkeyEnginu mohou být statické, kinetické a dynamické. V tabulce č.6 je zobrazen jejich přehled a vlastnosti.

Typ:	Statické	Kinematické	Dynamické
Příklad:	Nepohyblivá překážka, podlaha, budova, …	Dálkově ovládané předměty: Vzducholoď, meteorit, výtah, dveře, dálkově řízené NPC pomocí kontroléru, …	Interaktivní předměty: bedny, padající sloupy, kosmická loď s nulovou gravitací, …
Má hmotnost?	Ne, 0.0f	Ano,> 0.0f	Ano,> 0.0f
Jak se pohybuje?	nepohybuje	setLocalTranslocation();	setLinearVelocity(); applyForce(); setWalkDirection(); pro CharacterControl
Jak se umístí do scény?	setPhysicsLocation(); setPhysicsRotation ();	setLocalTranslation (); setLocalRotation ();	setPhysicsLocation (); setPhysicsRotation ();
Může se pohybovat a tlačit ostatní?	Ne	Ano	Ano

Je ovlivněn silami? (Spadne, když je ve vzduchu?)	Ne	Ne	Ano
Jak aktivovat toto chování?	setMass (0f); setKinematic (false);	setMass (1f); setKinematic (true);	setMass (1f); setKinematic (false);

Tab. 6 Statické, Kinematické, Dynamické objekty ve fyzice
Zdroj: [7]

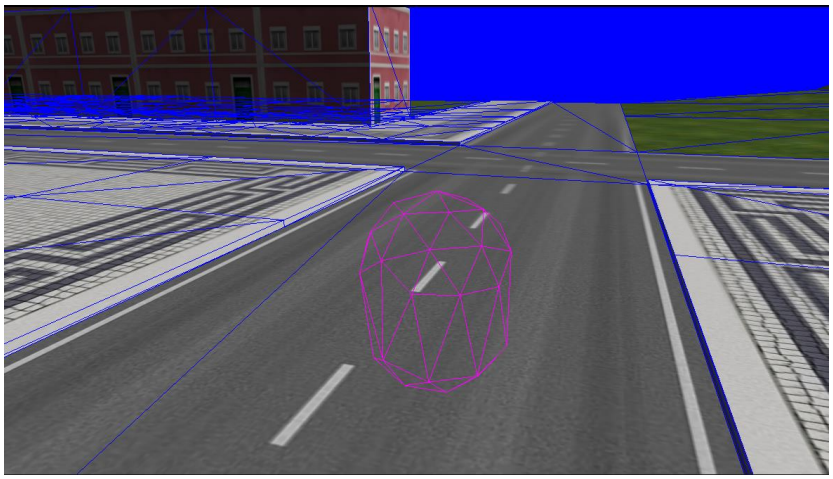
Kolizní model geometrie je možné vytvořit několika způsoby. Geometrie může obsahovat kolizní model už předem definovaný nebo vygenerovaný za běhu aplikace. Vytvoření předem definovaného modelu je vhodné u scény, u které neočekáváme dynamické změny. Kolizní model je možné vytvořit pomocí programu Blender nebo ve Scene Exploreru. Z důvodu optimalizace aplikace je lepší při načítání scény nejdříve všechny geometrie přidat k uzlu a následně nechat vygenerovat kolizní model k danému uzlu. JMonkeyEngine obsahuje několik druhů metod, pomocí kterých je možné vygenerovat kolizní model geometrie. Mezi dvě základní metody patří CollisionShapeFactory.createMeshShape(geometrie), pomocí které je možné vygenerovat kolizní model přesně kopírující geometrii předanou v parametru anebo pomocí new CapsuleCollisionShape(1f, 4f) přijímající parametry rádius a výšku, která vytvoří elipsoid. CapsuleCollisionShape se používá primárně pro kolizní model postav, jelikož se jedná o primitivní tvar, který je snadný pro výpočet a vůči válcovitému tvaru netrpí na zasekávání herní postavy o mírné nerovnosti prostředí. Kolizní model není generovaný automaticky při přidání Spatial do fyzického prostoru, aby bylo možné přidat např. nekolizní vegetaci na kolizní terén. Spatial, který odebíráme ze scény je nutné ručně odebrat také z fyzického prostoru, jinak by geometrie nebyla viditelná, ale fyzický model by stále byl přítomný v prostoru.

Na obrázku č. 37 je zobrazena ukázka kódu inicializující scénu terénu a hráče. K terénu je vygenerován kolizní model a k hráči kolizní elipsoid, ke kterému je možné v budoucnu přidělit kameru. Následně je přiřazen vytvořený objekt do fyzického prostoru aplikace. Všechny metody a třídy jsou popsány výše.

Na obrázku č. 38 je vykreslená scéna pomocí kódu z obrázku č. 37 se zobrazenými kolizními modely, čehož jde dosáhnout pomocí metody setDebugEnabled(true) u třídy BulletAppState.

```java
public class MyStateWithBullet extends BaseAppState
{

    AppState bulletAppState;
    AssetManager assetManager;
    AppStateManager stateManager;
    Node rootNode;
    Node terrain;
    CharacterControl player;

    @Override
    protected void initialize(Application app){
        bulletAppState = new bulletAppState();
        assetManager = app.getAssetManager();
        stateManager = .app.getStateManager();
        rootNode = app.getRootNode();
        //přidáme stav fyziky
        stateManager.attach(bulletAppState);
        //načteme terén a vytvoříme kolizní model
        terrain = assetManager.loadModel("Models/Terrain.j3o");
        CollisionShape terShape = CollisionShapeFactory.createMeshShape(terrain);
        RigidBodyControl terBody = new RigidBodyControl(terShape, 0);
        sceneModel.addControl(terBody);
        //vytvoříme kolizní model hráče, ke kterému může být připojena kamera
        CapsuleCollisionShape capsuleShape = new CapsuleCollisionShape(1f, 4f);
        player = new CharacterControl(capsuleShape, 80f);
        //přidáme Spatials do fyzického prostoru
        bulletAppState.getPhysicsSpace().add(terrain);
        bulletAppState.getPhysicsSpace().add(player);
        //přidáme terén do scény
        rootNode.attachChild(terrain);
    }
    @Override
    protected void cleanup(Application app){
        //při ukončení tohoto stavu, všechno odinicializujeme
        bulletAppState.getPhysicsSpace().remove(terrain);
        bulletAppState.getPhysicsSpace().remove(player);
        rootNode.detach(terrain);
        stateManager.detach(bulletAppState);
    }
}
```

Obr. 37 Ukázka inicializace fyziky
Zdroj: Vlastní tvorba



Obr. 38 Zobrazení kolizních modelů
Zdroj: Vlastní tvorba

Knihovna JBullet obsahuje také třídu com.jme3.bullet.control.GhostControl sloužící k detekci kolizí a průniků mezi fyzickými objekty. Objekt třídy GhostControl je nehmotný a neviditelný. Využití má při detekci zón, ve kterých se Rigidní objekty nachází. Např. detekce radioaktivní zóny, ve které hráči ubývají životy.

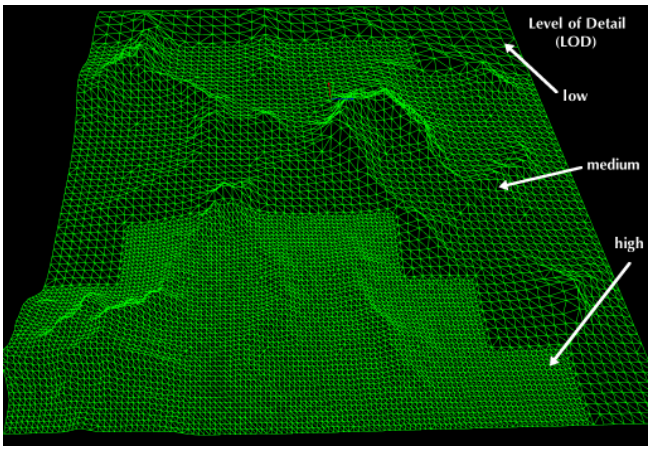
„Fyzický prostor běží standartně na 60 FPS. Tato rychlost nezávisí na skutečné rychlosti vykreslování snímků a nijak se jí nepřizpůsobuje. Místo toho, když je skutečná hodnota FPS větší než 60, systém zobrazuje interpolované pozice fyzických objektů. Když je rychlost FPS nižší než 60, budou fyzikální veličiny zvětšeny, aby nahradili chybějící výpočty.“ [7]

6.11 Generování terénu

JMonkeyEngine využívá ke generování terénu Geo Mipmapping. Jedná se o optimalizovanou technologii vykreslování terénu, pomocí rozdělování na menší části, u kterých je následně možné dynamicky snižovat úroveň detailů. *„Protože velká část terénu není viditelná (nenachází se v zobrazovaném objemu), není potřeba aby byla renderovaná, takže by se mělo předejít zbytečným výpočtům. Proto je datová struktura terénu rozdělena do struktury, nazvané Quadtree“* [5]

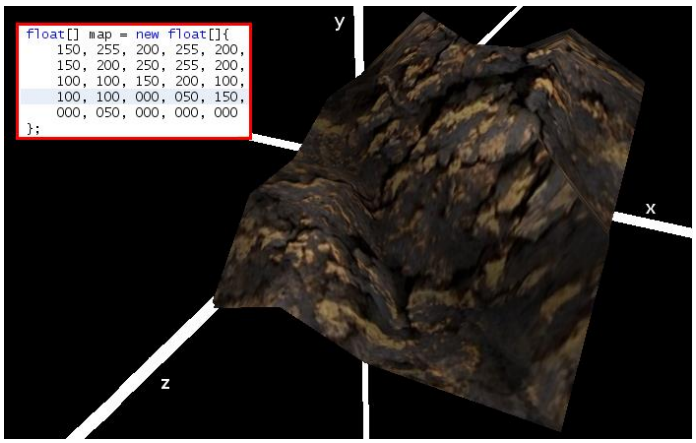
Quadtree princip je implementovaný pomocí tříd TerrainQuad a TerrainPatch v balíčku com.jme3.terrain.geomipmap. TerrainQuad konstruktor definuje mimo jiné parametry také jeho velikost a velikost jednotlivých TerrainPatch. Velikosti musí být po odmocnění celé číslo typu Integer. (Příklad: Velikost TerrainQuad je 1024 a velikost TerrainPatch je 128, tak hlavní TerrainQuad je rozdělen na 4 další TerrainQuad a ty jsou následně rozdělený na TerrainPatch). Tyto třídy poskytují prostředky ke správě úrovně detailů terénu. TerrainPatch, kterému byla snížena úroveň detailů informuje okolní TerrainPatch, na které má uložený odkaz, aby aktualizovali svojí geometrii. Bez tohoto informování by se mohl terén vykreslovat chybně s dírami. Snižování detailu funguje pomocí upravování vyrovnávací paměti indexů trojúhelníků, takže celá geometrie nemusí být znova načítána do paměti GPU. Jednotlivý TerrainPatch si drží skutečnou geometrii terénu, proto jí není nutné při zvyšování úrovně detailů opět znovu načítat. Na obrázku č. 39 je zobrazen úbytek detailů v závislosti na vzdálenosti kamery od jednotlivých TerrainPatch. Podpora

jeskyní a útesů při vykreslování terénu pomocí Geo Mipmapping algoritmu není v JMonkeyEnginu implementována, ale je plánovaná.



Obr. 39 Dynamická úroveň detailů terénu
Zdroj: [7]

Data definující výšku terénu jsou obecně uloženy jako floatové pole hodnot v rozsahu od 0 do 255. Na obrázku č. 40 je zobrazený převod floatového pole do mapy.



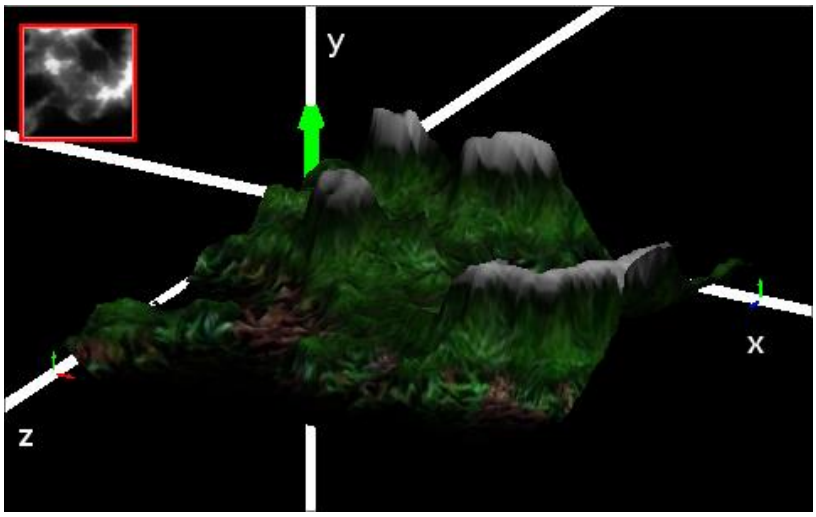
Obr. 40 Převod Float pole na mapu
Zdroj: [7]

K vygenerování těchto dat se v JMonkeyEnginu využívají dva základní principy. První princip je FractalHeightMapGrid využívající fraktálovou knihovnu k vytvoření komplexního, avšak opakovaného vzoru za běhu aplikace. Tvar terénu je možné modifikovat pomocí několika parametrů a fraktálových post filtrů, které jsou součástí knihovny. Na obrázku č. 41 je zobrazena implementace generátoru fraktálového vzoru (Fractal noise). Následně je možné získat floatBuffer ze třídy com.jme3.terrain.noise.Basis obsahující vygenerovaný šum. Druhý princip

ImageBasedHeightMapGrid využívá 16bitové výškové mapy ve stupních šedi. Obrázek musí mít stejné rozlišení jako je velikost terénu. Barva jednotlivých pixelů reprezentuje výšku v daném bodě (čím tmavší pixel je, tím nižší terén v daném bodě bude). Na obrázku č. 42 je zobrazen příklad 16bitové výškové mapy.

```
private Basis createFractalNoise(final float weight)
{
    final FractalSum base = new FractalSum();
    base.setRoughness(1.2f);
    base.setFrequency(0.2f);
    base.setAmplitude(weight);
    base.setLacunarity(2.12f);
    base.setOctaves(8);
    base.addModulator((NoiseModulator) (float... in) -> ShaderUtils.clamp(in[0] * 0.5f + 0.5f, 0, 1));

    final FilteredBasis ground = new FilteredBasis(base);

    final PerturbFilter perturb = new PerturbFilter();
    perturb.setMagnitude(0.2f);

    final OptimizedErode therm = new OptimizedErode();
    therm.setRadius(5);
    therm.setTalus(0.011f);

    final SmoothFilter smooth = new SmoothFilter();
    smooth.setRadius(1);
    smooth.setEffect(0.1f); // 0.7

    final IterativeFilter iterate = new IterativeFilter();
    iterate.addPreFilter(perturb);
    iterate.addPostFilter(smooth);
    iterate.setFilter(therm);
    iterate.setIterations(1);

    ground.addPreFilter(iterate);

    return ground;
}
```

Obr. 41 Generátor fraktálového vzoru
Zdroj: Vlastní tvorba



Obr. 42 Mapa vygenerovaná z obrázku
Zdroj: [7]

7 Shrnutí výsledků

Hlavní cílem práce bylo vytvořit funkční koncept aplikace umožňující vytvářet statické scény, upravovat vizuální stránku post procesorových efektů, rozmisťovat počítačem řízené postavy, vytvářet příběhovou linii a následně umožnit vytvořenou scénu a příběh si zahrát. Použité soubory textur a modely jsou volně dostupná testovací data JMonkeyEnginu. Je možné přidávat a mazat uložené soubory (modely, textury a další) z datové složky aplikace, když je vypnutá.

Podmínky byly splněny a vypracovány tak, že nadále je snadné rozšiřovat aplikaci o novou funkcionalit. Velkou pozornost bylo potřeba věnovat dokumentaci JMonkeyEnginu a Nifty GUI knihovny, aby byly všechny dostupné třídy a funkce použity správně. Velkou oporou bylo i uživatelské fórum JMonkeyEnginu [8], na kterém je možné nalézt inspiraci k řešení konkrétních problémů. Mnoho informací bylo nalezeno v prezentacích *Math for dummies* [12] a *Scene graph for dummies* [19] vytvořených přímo pro JMonkeyEngine, kde bylo mnoho informací o matematických operacích s vektory a správného použití grafu scény. Pozornost byla také směrovaná na dokumentaci k LWJGL a OpenGL knihovnám, aby bylo možné správně a co nejefektivněji modifikovat vyrovnávací paměť GPU k dosažení drobných změn ve scéně.

Očekávaná náročnost vývoje aplikace pro návrh RPG her byla mnohem menší, než ve skutečnosti byla. Nifty GUI a JMonkeyEngine mají v určitých místech velice zastaralou a neaktualizovanou dokumentaci, což stěžovalo práci a nutilo hledat řešení na fórum.

Práce s Nifty GUI je hodně náročná, protože definice uživatelského rozhraní pomocí XML poskytuje velký prostor pro chybovost, z důvodu toho že psaní identifikátorů je zdvojené (XML a Java), takže může být napsáno nekonzistentně a XML definice rozhraní je velice dlouhá a nepřehledá. Následně bylo rozhodnuto použít pro tvorbu GUI definici přímo v jazyce Java a umístit identifikátory do konstant a zamezit tak vzniku chybě. Tato implementace Nifty GUI psaného přímo v Java kódu je relativně nová funkce, takže nejsou implementovány všechny dostupné funkce.

8 Závěry a doporučení

Vlastní myšlenka užitečnosti této aplikace, je prozatím postavená pouze na mých domněnkách a předešlých zkušenostech z pozice hráče a fanouška tohoto typu her. Aplikace bude nadále vyvíjená přes rozsah této práce a až bude uznáno za vhodné bude prezentována/navržena přímo vývojářům anebo formou startupu.

V budoucnu bude potřeba přizpůsobovat aplikaci novějším verzím JMonkeyEnginu, protože její vývoj je rychlí a s novou verzí vždy přichází plno užitečných a optimalizačních změn. Ukládací formát pro post procesorové efekty scény je v aktuální verzi pomocí binárních souborů, takže bude vhodné uložit konfiguraci do XML nebo textového formátu. Bude potřeba předělat uživatelské rozhraní v menu aplikace, aby nabízelo více možností pro správu datových souborů aplikace a vytvořit vlastní konfigurační soubory v datové složce a rozhraní k nastavení antialiasing, rozlišení, obnovovací frekvence a dalších. Dále bude muset být implementována nápověda po zmačknutí klávesy F1 v každém editačním okně, aby bylo uživateli vysvětleno ovládání a chování aplikace v daném editačním rozhraní. Také bude muset být implementováno více možností při tvoření příběhové linie, aby nebylo možnost začít jednu část, bez splnění předchozí části.

Uživatelské rozhraní využívající knihovnu Nifty GUI bude předěláno k využití pouze knihovny Lemur GUI, které je pro JMonkeyEngine přímo vyvinuto. Nifty GUI nabízí širokou škálu předvytvořených ovládacích prvků, ale jeho výkon značně omezuje plynulost běhu aplikace. Vysoká náročnost je zapříčiněná používáním zastaralých komponent k vykreslování rozhraní a jejich aktualizace ze strany vývojářů není plánovaná, protože Nifty GUI podporuje několik dalších implementací a nestíhají aktualizovat všechny tak, aby využívali nejnovější možnosti cílové platformy.

9 Seznam použité literatury

[1] BLENDER, Documentation team: Blender 2.79 Manual [online]. [cit. 25.3.2019] <https://docs.blender.org/manual/en/latest/index.html>

[2] Blender official, main page, [online], [cit 16.4.2019], <https://www.blender.org>

[3] Bullet Physics SDK, documentation, [online], [cit 7.4.2019], <https://github.com/bulletphysics/bullet3/tree/master/docs>

[4] CLANTON, Chuck. An interpreted demonstration of computer game design. In: CHI 98 conference summary on Human factors in computing systems – CHI '98 [online]. New York, New York, USA: ACM Press, 1998, 1998, s. 1-2 [cit. 2019-04-05]. DOI: 10.1145/286498.286499. ISBN 1581130287. <http://portal.acm.org/citation.cfm?doid=286498.286499>

[5] DE BOER, Willem H. Fast Terrain Rendering Using Geometrical MipMapping [online]. 2000, [cit. 2019-04-10]. <http://www.flipcode.com/archives/article_geomipmaps.pdf>

[6] HOHMUTH, Jens, KARING, Martin, Nifty Gui The Missing Manual, 28.12.2011, [internet], [cit 5.4.2019], <http://sourceforge.net/projects/nifty-gui/files/nifty-gui/1.3.2/nifty-gui-the-manual-1.3.2.pdf/download>

[7] JMonkeyEngine, documentation, [2019], [online], [cit 15.3.2019], <https://wiki.jmonkeyengine.org>

[8] JMonkeyEngine Forum, forum, [online], [cit 15.4.2019], <https://hub.jmonkeyengine.org/>

[9] JMonkeyEngine official, main page, [online], [cit 16.4.2019], <http://jmonkeyengine.org>

[10] KUSTERER, Ruth. JMonkeyEngine 3.0 beginner's guide: develop professional 3D games for desktop, web, and mobile, all in the familiar Java programming language. Birmingham: Packt Pub, [2013]. ISBN 9781849516464

[11] Lightweight Java Game Library official, main page, [online], [cit 16.4.2019], <https://www.lwjgl.org>

[12] Math for Dummies, presentations, jmonkeyengine.com, [online], [cit 16.4.2019], <https://wiki.jmonkeyengine.org/tutorials/math/assets/fallback/index.html>

[13] NIFTY GUI DOCS, documentation, [online], [cit 5.4.2019], <https://github.com/nifty-gui/nifty-gui/wiki>

[14] Nifty Gui official, main page, [online], [cit 16.4.2019], <https://github.com/nifty-gui/nifty-gui>

[15] OpenGL Wiki. Portal: OpenGL Concepts [online]. OpenGL Wiki, 2017 Sep 15, 04:55 UTC [cit. 29.3.2019]. <http://www.khronos.org/opengl/wiki_opengl/index.php?title=Portal:OpenGL_Concepts&oldid=14028>

[16] ORACLE official, main page, [online], [cit 17.4.2019], <https://www.oracle.com/cz/java>

[17] ORACLE DOCS, Oracle Americana, Inc, [2016], [online], [cit 20.3.2019], <https://docs.oracle.com>

[18] ORACLE JAVA SPECIFICATION, Version 12, Oracle Americana, Inc, [2019], [online], [cit. 20.3.2019] <https://docs.oracle.com/javase/specs/jls/se12/jls12.pdf>

[19] Scenegraph for Dummies, presentations, jmonkeyengine.com, [online], [cit 16.4.2019], <https://wiki.jmonkeyengine.org/tutorials/scenegraph/assets/fallback/index.htm>

[20] SELLAR, Tracey. User Experience in Interactive Computer Game Development. MASOODIAN, Masood, Steve JONES a Bill ROGERS, ed. Computer Human Interaction [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, 2004, s. 675-681 [cit. 28.3.2019]. Lecture Notes in Computer Science. DOI: 10.1007/978-3-540-27795-8_77. ISBN 978-3-540-22312-2. <http://link.springer.com/10.1007/978-3-540-27795-8_77>

[21] Smart trends in information technology and computer communications. New York, NY: Springer Berlin Heidelberg, 2017. ISBN 978-981-10-3432-9.

[22] SOBH, Tarek M. a Khaled ELLEITHY. Innovations in Computing Sciences and Software Engineering. Dordrecht: Springer, [2010]. ISBN 978-90-481-9112-3.

[23] XU, Chong-wei. Learning Java with games. New York, NY: Springer Science+Business Media, [2018]. ISBN 978-3-319-72885-8.

10 Přílohy

1) CD se zdrojovým kódem a sestavenou aplikací
2) Podklad pro zadání bakalářské práce studenta

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Horáček Viktor	Na Vyhlídce 2208, Dvůr Králové nad Labem	I1500363

TÉMA ČESKY:

Software pro návrh příběhu 3D RPG her

TÉMA ANGLICKY:

Software for story designing of 3D RPG games

VEDOUCÍ PRÁCE:

prof. RNDr. PhDr. Antonín Slabý, CSc. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je vyvinout aplikaci návrhu příběhu počítačové hry na hrdiny(RPG) pomocí JMonkeyEnginu, který umožní uživateli vytvořit hratelný příběh a amatérský vzhled 3D světa bez znalosti programování. Při modelování herního světa bude uživatel schopný editovat všechny prvky pouze pomocí předpřipravených nástrojů pomocí uživatelského rozhraní.

V textu práce budou představeny elementy software použité v aplikaci a zkušenosti s jejím vývojem.

Vlastní aplikace bude obsažena v příloze k práci.

SEZNAM DOPORUČENÉ LITERATURY:

JMonkeyEngine, documentation, [2019], [online],
<https://wiki.jmonkeyengine.org>

JMonkeyEngine official, main page, [online],
<http://jmonkeyengine.org>

KUSTERER, Ruth. JMonkeyEngine 3.0 beginner's guide: develop professional 3D games for desktop, web, and mobile, all in the familiar Java programming language. Birmingham: Packt Pub, [2013]. ISBN 9781849516464

Podpis studenta: ..... **Datum:** 7.2.2019

Podpis vedoucího práce: ..... **Datum:** 7.2.2019