VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# 3D APLIKACE PRO MOBILNÍ TELEFONY

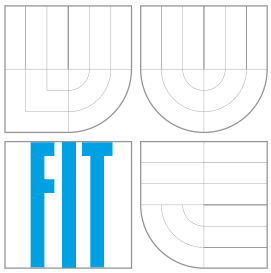
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ SLAVOTÍNEK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# 3D APLIKACE PRO MOBILNÍ TELEFONY

3D APPLICATION FOR MOBILE PHONES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ SLAVOTÍNEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2010

# Abstrakt

Práce se zabývá zobrazováním 3D grafiky na platformě Java ME. Nejprve je obecně popsána architektura této platformy. Dále je pozornost věnována popisu a srovnání API rozhraní JSR 184 a MascotCapsule. Praktická část se týká tvorby aplikace s užitím rozhraní MascotCapsule. Je popisován návrh a implemntace jednoduché hry inspirované dvojrozměrným Dyna Blasterem.



# Abstract

This bachelor thesis deals with 3D graphics representation on Java ME platform. The general architecture of this platform is described first. Consideration is given to description of API interfaces JSR 184 and MascotCapsule. Practical part is about application implemented with usage of MascotCapsule interface – simple game based on 2D Dyna Blaster.



# Klíčová slova

3D grafika, J2ME, Java ME, JSR 184, API pro 3D grafiku na Java ME mobilních zařízeních, MascotCapsule.

# Keywords

3D Graphics, J2ME, Java ME, JSR 184, Mobile 3D Graphics API for Java ME, MascotCapsule.



# Citace

Tomáš Slavotínek: 3D aplikace pro mobilní telefony, bakalářská práce, Brno, FIT VUT v Brně, 2010

# 3D aplikace pro mobilní telefony

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Aleše Láníka a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Slavotínek
19. května 2010

## Poděkování

Tímto bych chtěl poděkovat vedoucímu práce, Ing. Aleši Láníkovi za jeho podporu a rady při tvorbě tohoto textu a praktické implementaci.

# Obsah

# Kapitola 1

# Úvod

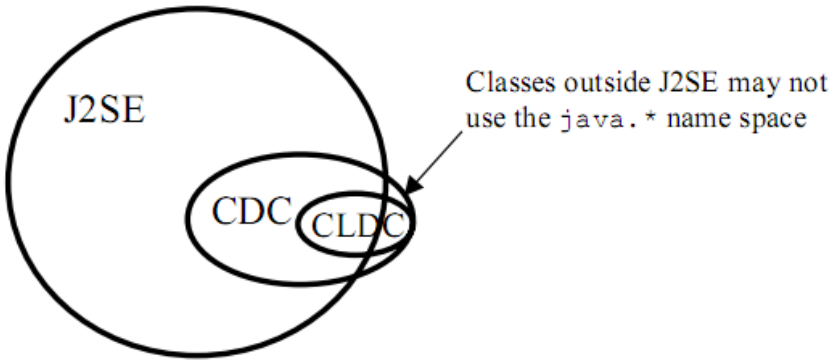
S trojrozměrnou grafikou se na platformě osobních počítačů setkáváme již zhruba 20 let. Dnes její užití vnímáme jako zcela běžné a k odvětvím jako jsou návrhové systémy, simulace či hry neodmyslitelně patří. Přičemž poslední jmenovaná kategorie je v oblasti 3D grafiky "tahounem" a tlačí schopnosti osobních počítačů a zejména grafických akcelerátorů velmi rychle kupředu. V době, kdy se první trojrozměrné aplikace pro osobní počítače objevily, byly tyto vybaveny procesorem taktovaným na několik desítek MHz, operační pamětí s kapacitou jednotek MB a displayem s rozlišením většinou $640 \times 480$ bodů. Přičemž taková sestava zabrala prostor pracovního stolu. Dnes můžeme díky miniaturizaci zařízení s podobnými parametry bez potíží nosit v kapse v podobě mobilního telefonu, PDA, audiovizuálního přehrávače či GPS navigace. Původně jednoúčelová zařízení jako telefony se tak díky procesoru pracujícím na frekvenci několika set MHz a paměti s kapacitou jednotek či desítek MB stávají multifunkčními. Stále rostoucí rozlišovací schopnost těchto zařízení umožňuje provoz multimédií a zobrazení kvalitní 2D a 3D grafiky.

Práce se zabývá možnostmi a uplatněním trojrozměrné grafiky na běžných mobilních telefonech v prostředí Java Micro Edition (Java ME). Čtenář bude stručně seznámen s architekturou platformy Java. Prostředí Java ME bude rozebráno podrobněji, včetně popisu dostupných konfigurací a profilů. Hlavním cílem práce je však popis a srovnání dvou rozhraní pro mobilní 3D grafiku – MascotCapsule a JSR 184. Po obecném seznámení s rozhraními bude následovat popis jednotlivých tříd a jejich schopností. Srovnání obou rozhraní proběhne formou příkladu, rozčleněného na řešení dílčích podproblémů doplněné slovními popisy. Po prostudování tohoto textu by čtenář měl být schopen objektivně zvolit rozhraní vhodné pro konkrétní aplikaci. Praktická část se bude zabývat návrhem a implementací komplexnější aplikace s užitím jednoho z diskutovaných 3D rozhraní.

# Kapitola 2

# Platforma Java

Java ME je jedna ze 4 dostupných Java platforem, které se navzájem liší cílovými zařízeními a tedy i vlastnostmi a schopnostmi, které vývojářům a jejich aplikacím poskytují. Rozsah a vztah jednotlivých platforem je znázorněn na obrázku 2.1.




Obrázek 2.1: Vztah plaforem Java [11]

Nejrobustnější je platforma *Java Enterprise Edition* (Java EE), která je určena pro nasazení zejména na výkonných systémech jako jsou servery. Nejrozšířenější platformou je *Java Standard Edition* (Java SE) s uplatněním na běžných stolních počítačích, noteboocích apod. Platformě Java ME se budeme blíže věnovat v následující kapitole. Platforma pro nejjednodušší zařízení jako jsou "chytré karty" nese označení *Java Card*.

## 2.1 Platforma Java ME

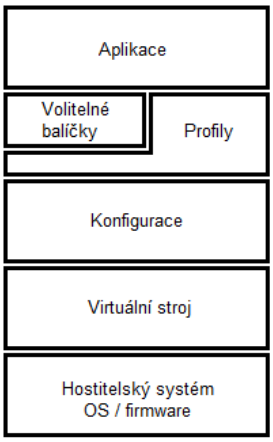
Původní název platformy byl *Java 2 Micro Edition* nebo zkráceně *J2ME*. V průběhu roku 2005 se "2" z názvu oficiálně vypustila [2]. Některé zdroje však stále používají starší označení, které je rovněž použito v původních dokumentech a specifikacích. Všechna pojmenování však označují jednu a tu samou platformu. V tomto textu budeme používat novější označení Java ME.

Platforma Java ME je určená pro použití na přenosných zařízeních, jako jsou mobilní telefony a PDA. Tato kategorie zařízení poskytuje omezené prostředky ve srovnání se stolními počítači. Omezení se týkají zejména kapacity operační a trvalé paměti, výpočetního výkonu, rozlišení displaye a možností napájení. Java ME tyto parametry zohledňuje a umožňuje efektivní tvorbu aplikací pro tuto cílovou platformu. [12], [18], [17]

Platformu Java ME lze rozložit do několika vrstev:

- Virtuální stroje
- Konfigurace
- Profily
- Volitelné balíčky

Návaznost jednotlivých vrstev a jejich postavení mezi hostitelským systémem a uživatelskou aplikací můžeme vidět na obrázku 2.2.




Obrázek 2.2: Vrstvy platformy Java ME

Jednotlivé prvky tohoto modelu stojící nad hostitelským systémem si podrobněji popíšeme. Samotný hostitelský systém se může lišit zařízení od zařízení, přičemž velmi často nejsou vzájemně kompatibilní. To se týká zejména jednodušších zřízení vybavených pouze firemním softwarem (firmware), který bývá specifický právě pro jeden konkrétní model. Výkonnější zařízení jako jsou inteligentní telefony (smartphones) a PDA můžou být vybaveny jednoduchým operačním systémem např. OS Symbian. Tyto systémy typicky bývají společné pro širší skupinu zařízení.

### 2.1.1 Virtuální stroje

Virtuální stroj (Virtual Machine, VM) je klíčový prvek pro spouštění aplikací vytvořených v jazyce Java. Spouštěná aplikace se nachází ve formě takzvaného mezikódu s označením Java bytecode. Virtuální stroj z pohledu aplikace odstiňuje hostitelský software (operační systém, firmware) a hardware. Vytváří tak pro aplikace universální prostředí. V závislosti na Java platformě je k dispozici několik rozdílných virtuálních strojů. Platformy Java SE a EE poskytují plnohodnotný stroj nazvaný Java Virtual Machine (JVM). Platforma Java ME může využívat také JVM nebo jednodušší variantu označovanou K Virtual Machine (KVM). Platforma Java Card využívá nejjednodušší stroj Card VM.

To, jaký stroj (JVM/KVM) bude Java ME používat závisí na použité konfiguraci CLD nebo CLDC (viz dále).Výkonnější mobilní zařízení používající konfiguraci CLD poskytují

plnohodnotný stroj JVM. Tento text je zaměřen na zařízení méně výkonná a tedy platformu Java ME v konfiguraci CLDC. Pro tato zařízení je virtuální stroj speciálně upraven, aby splňoval zejména tato kritéria [11]:

- Malé jádro o velikosti 40-80kB, které bude staticky uloženo v paměti.
- Dobrá přenositelnost
- Kvalitní dokumentace
- Modulárnost a přizpůsobitelnost
- Úplnost a rychlost stroje bez nutnosti obětovat jednu či druhou vlastnost.

První z kritérií se promítlo i do názvu virtuálního stroje – písmeno K v označení KVM znamená "kilo". Tento název byl zvolen, protože dostupná paměť na těchto zařízeních se počítá obvykle v kilobytech, narozdíl od stolních počítačů kde pro měření paměťových nároků používáme typicky MB. Minimální implementace vyžaduje pro provoz přibližně 128kB paměti. Paměťový prostor je rozdělen mezi vlastní jádro KVM, knihovny a paměť pro aplikaci (heap memory). Častější je však implementace s alespoň 256kB paměti, z čehož přibližně polovina je k dispozici uživatelské aplikaci.

Požadavek na přenositelnost byl rovněž splněn a v praxi se s KVM můžeme setkat na procesorech schopných zpracovat různou šířku slova (16 nebo 32 bitů) a disponujících jak redukovanou (RISC) tak kompletní (CISC) instrukční sadou [11].

### 2.1.2 Konfigurace CDC/CLDC

Konfigurace byly vytvořeny za účelem seskupit zařízení s podobnými výkonnostními a paměťovými parametry. Z pohledu software pak specifikuje zejména tyto parametry [10]:

- dostupné vlastnosti vlastního jazyka Java
- schopnosti VM
- dostupné Java knihovny a API rozhraní

Vrstva konfigurací je velmi těsně provázána s vlastním virtuálním strojem. Přihlédneme-li ke složitosti virtuálního stroje, znamenala by změna ve specifikaci konfigurace značný i zásah do implementace virtuálního stroje, což by byla velmi časově a finančně náročná operace. Proto existuje jen malý počet konfigurací, tak aby byly udržovatelné. V současné době jsou definovány dvě standardní konfigurace:

- Connected Device Configuration (CDC)
- Connected Limited Device Configuration (CLDC)

Většina z funkcionality CDC a CLDC byla převzata z Java SE. Třídy převzaté z Java SE jsou buďto naprosto shodné se svou předlohou nebo se jedná o podmnožinu dané třídy. Mimo tyto zděděné prvky jsou profily CDC a CLDC rozšířeny o funkce specifické

pro přenosná zařízení. Profil CDC je určený pro robustnější zařízení s větším množstvím paměti (obvykle více než 10MB) jako jsou vestavěné systémy, high-endová PDA či set-top boxy. Profil CLDC je zaměřen zejména na běžné mobilní telefony a levnější (low-end) PDA.[17]

CLDC se věnuje zejména těmto oblastem:

- dostupné funkce jazyka Java a virtuálního stroje
- základní Java knihovny (`java.lang.*` a `java.lang.*`)
- vstup a výstup
- síťová komunikace
- bezpečnost
- lokalizace

Naproti tomu neřeší atributy, které jsou zohledněny v profilech implementovaných nad CLDC:

- řízení životního cyklu aplikace (instalace, spouštění, odstranění)
- uživatelské rozhraní
- zpracování událostí

Tyto Java profily budou podrobněji rozebrány v následujícím textu.

### 2.1.3 Profily MIDP

Jeden ze základních cílů platformy Java je přenositelnost aplikací. Cílová zařízení platformy Java ME mohou být velmi rozdílná, lze je však rozdělit do konkrétních kategorií jako jsou vestavěné systémy, mobilní telefony a PDA, herní zařízení apod. Přenositelnost mezi jednotlivými skupinami není vyžadována a vzhledem k omezenému výkonu a prostředkům cílových zařízení by mělo její zajištění spíše negativní účinek. Důležitá je kompatibilita v rámci jedné skupiny zařízení. Aby však bylo možné ji zajistit je potřeba vlastnosti a potřeby jednotlivých skupin zařízení přesně specifikovat, což je úkolem právě Java ME profilů. Rovněž může být užitečné aby profil umožňoval provoz určité aplikace či aplikací i na obecnější skupině zařízení. Je tedy možné aby určité zařízení podporovalo i více profilů současně. Některé z nich specifické pro dané zařízení a jiné specifické pro aplikaci. [18], [17]

Na softwarové úrovni profil definuje kolekci API rozhraní a knihoven postavených nad určitou konfigurací. Poskytují tak dodatečné možnosti nad touto konfigurací, které přesně vyhovují určitému segmentu trhu.

Jedna velká skupina je tvořena *mobilními informačními zařízeními* (Mobile Information Devices, MID). Pro tato zařízení je určen profil *Mobile Information Device Profile* (MIDP) pracující nad konfigurací CLDC [1]. Tento profil je v současné době dostupný v revizi MIDP

[1] Původně stál profil MIDP nad konfigurací CLDC 1.0 – nyní i nad CLDC 1.1.

2.0, vychází z MIDP 1.0 a poskytuje zpětnou kompatibilitu. Je tedy možné spouštět aplikace původně napsané pro MIDP 1.0 v prostředí MIDP 2.0. Dnešní MID zařízení oplývají širokou řadu schopností. Profil MIDP se je nesnaží obsáhnout všechny, ale omezuje dostupná API rozhraní pouze na oblasti, které jsou opravdu nezbytné pro zachování přenositelnosti a úspěšné nasazení [15]. Zejména se jedná o tyto:

- řízení životního cyklu aplikace (instalace, spouštění, odstranění)
- uživatelské rozhraní
- zpracování událostí
- doručení a placení aplikace
- bezpečnost a podepisování aplikací
- síťová komunikace
- trvalé paměťové uložiště
- zvuk
- časovače

# Kapitola 3

# 3D grafika na mobilních zařízeních

Vývoj 3D aplikací bez jakékoliv podpory ze strany výrobců mobilních zařízení by byl obtížný a časově náročný. I přesto máme možnost setkat se s aplikacemi, které jsou schopné plnohodnotného trojrozměrného zobrazení bez jakékoliv dodatečné softwarové či hardwarové podpory od cílového zařízení. Jako příklad můžeme jmenovat grafický engine *Gem Xenotime 3D* polské společnosti Gameleons Sp z o. o., který je schopen pracovat jak na telefonech s 3D softwarovým rozhraním tak bez něj[9]. Na zmíněném enginu běží např. velmi známá akční hra Alpha Zone 3D, kterou si tak můžou zahrát i majitelé "non-3D" telefonů jako je Nokia 3510i.

I tak se ale jedná spíše o výjimku a vývojáři 3D aplikací, zejména her, většinou staví na některém z dostupných 3D rozhraní. Naprostá většina dnešních mobilních zařízení některé rozhraní podporují a tak se tvorba vlastních enginů starajících se o zobrazení "na vlastní pěst" stává historií. Stejně jako např. softwarový renderovací režim na PC, který byl vytlačen zejména (tehdy velmi rozšířeným) rozhraním Glide 3D API a rozhraními Direct3D a OpenGL. Na poli běžných mobilních zařízení není výběr rozhraní rovněž nikterak široký, pomineme-li opravdu výkonné stroje, kde se můžeme setkat i se zmíněnými rozhraními Direct3D či OpenGL věčně odpovídající podpory ze strany hardwaru. Cílem této práce je zaměřit se na zařízení běžná a aplikace tvořené v prostředí jazyka Java. Na platformě Java ME máme v současné době [1] na výběr ze dvou následujících rozhraní pro práci s 3D grafikou:

- JSR 184 (M3G)
- MascotCapsule

Obě rozhraní se navzájem v mnohém liší a jejich popisem se budeme zabývat v následujících kapitolách. Rovněž bude uvedeno srovnání, které by čtenáři mělo pomoci při výběru vhodného rozhraní pro specifickou aplikaci.

## 3.1 JSR 184

Plný název dle specifikace Java Specification Request 184 (JSR 184)[13] zní *Mobile 3D Graphics API for J2ME*$^{TM}$ nebo zkráceně *M3G*. Jedná se o volitelný balíček platformy Java ME pro práci s 3D grafikou, který bývá postaven nad konfigurací CLDC. Tato konfigurace

[1] Stav z roku 2010.

je typická pro mobilní telefony či PDA kde se setkáváme s profily MIDP 1.0 a MIDP 2.0. Jak již bylo řečeno tyto profily byly vytvořeny aby pokryly základní funkčnost zařízení a typicky bývají doplněny o řadu volitelných balíčků jako je právě JSR 184.

I přesto že je JSR 184 volitelnou součástí, jedná se o rozhraní standardizované, což znamená, že aplikace napsané s jeho užitím jsou poměrně snadno přenositelné. Musíme však při implementaci zohlednit některé faktory, jako jsou výkonové rozdíly mezi zařízeními, různá rozlišení displaye, odlišné ovládání apod.

Rozhraní poskytuje dva odlišné přístupy k práci s 3D grafikou:

- vysokoúrovňový režim (high-level, retained mode)
- nízkoúrovňový, bezprostřední režim (low-level, immediate mode)

Jak již názvy režimů napovídají, jejich odlišnost spočívá v tom, na jaké úrovni s grafikou pracujeme. Vysokoúrovňový režim umožňuje práci se scénou jako s množinou objektů. Objektem muže být např. model vytvořený předem v programu pro 3D modelování a posléze převedený do formátu *.m3g. Jako objekt rovněž chápeme zdroj světla starající se o potřebné nasvícení modelů, či kameru, kterou na scénu pohlížíme.

Nízkoúrovňový režim nám naproti tomu poskytuje možnost pracovat se scénou na daleko nižší úrovni. Jednotlivé objekty jsou vytvářeny programově z grafických primitiv až při spouštění či za běhu aplikace. Tento postup může být někdy výhodnější nebo pro konkrétní aplikaci i jediný možný.

Sestavování scény pouze jedním či druhým způsobem by bylo značně limitující. Například můžeme jako vývojáři chtít vytvořit statické prostředí hry (mapu) z grafických primitiv při spouštění aplikace a posléze ho vyplnit dynamickými předměty a pohyblivými nepřáteli, jejichž modely jsme si vytvořili včetně animací a otexturování pohodlně v 3D studiu. Tuto skutečnost si skupina pracující na JSR 184 dobře uvědomovala a umožnila nám současné použití obou postupů, čímž nám poskytla velmi silný a flexibilní nástroj na tvorbu 3D aplikací.

### 3.1.1 Třídy rozhraní JSR 184

Rozhraní JSR 184 nabízí 30 tříd. V následujícím textu se budeme zabývat jejich orientačním popisem. Podrobnosti k metodám a vlastnostem jednotlivých tříd je možné nalézt ve specifikaci *Java Specification Requests – JSR 184*[13].

#### `Graphics3D`

Třída `Graphics3D` odpovídá návrhovému vzoru singleton. Musí být svázána s objektem, na který bude probíhat kreslení, čehož dosáhneme voláním metody `bindTarget()`. Třída hraje důležitou roli, protože jejím prostřednictvím probíhá veškeré vykreslování, k čemuž nám slouží sada metod `render()` s různými parametry pro různé entity. Konkrétně umožňují kreslení celého trojrozměrného světa, *uzlů grafické scény* a jednoduchých útvarů zvaných *submeshe*, což jsou v JSR 184 *základní grafická primitiva*. Vlastní vykreslení aktuálního snímku námi připravené scény vyvoláme metodou `releaseTarget()`.

Mimo to se `Graphics3D` stará o další funkcionalitu, jako je ořezávání (clipping) scény pouze na její viditelnou část, což je důležité zejména z hlediska výkonu. Rovněž umožňuje

nastavit řadu parametrů vykreslování, jako je aplikace Antialiasingu pro vyhlazení hran objektů, Ditheringu zvyšujícího barevnou hloubku nebo použití TrueColor zobrazení.

`Object3D`

Jedná se o abstraktní třídu tvořící základ všech objektů, které se mohou stát součástí trojrozměrného světa. Dědí od ní všechny dále uvedené třídy.

`Image2D`

`Image2D` je dvojrozměrný obrázek, který může být použit jako textura na objekt, pozadí scény nebo jako sprit. JSR 184 umožňuje definovat obrázek jako měnitelný či neměnný. Obsah měnitelného obrázku může být kdykoliv modifikován, naproti tomu použití neměnného obrázku vede k lepší optimalizaci a urychlení kódu. Mimo to je uchovávána informace o rozměrech obrázku a jeho formátu. Formát může být klasický RGB, RGBA s alpha kanálem, pouze alpha kanál, kanál udávající odrazivost, případně kombinace dvou posledně jmenovaných.

`Background`

Třída určuje, jakým způsobem bude docházet k mazání scény. Je možné mazat scénu určitou barvou, vykreslením obrázku `Image2D` nebo mazání nepoužívat vůbec. K mazání dojde při volání metody `Graphics3D.clear()`.

`Transform`

Jedná se o klasickou transformační matici $4 \times 4$. Umožňuje operace jako je posunutí, rotace, změna měřítka, inverze, transponování matice a nastavení na jednotkovou matici.

`AnimationController`, `AnimationTrack` a `KeyframeSequence`

Tyto tři třídy souvisejí s realizací animací v prostředí JSR 184. `AnimationController` se stará o řízení *sekvence animací* a spravuje atributy jako je pozice, rychlost a váha animace. Sekvence animací je tvořena několika objekty třídy `AnimationTrack`. `AnimationTrack` slouží k jako prostředník mezi konkrétní sekvencí snímků `KeyframeSequence` a konkrétním `AnimationController`em. `AnimationTrack` je možné přiřadit k více objektům, čímž lze dosáhnout přehrání stejné animace řízené z jednoho místa na více objektech paralelně. Přidání či odebrání animace u konkrétního objektu provádíme voláním jeho metod `addAnimationTrack()` a `removeAnimationTrack()`.

`VertexArray`, `VertexBuffer`, `IndexBuffer` a `TriangleStripArray`

Pomocí těchto tříd můžeme vytvářet grafické objekty na nízké (bezprostřední) úrovni. Universální třída `VertexArray` uchovává pole celočíselných vektorů. Využíváme ji pro uložení základních vlastností vytvářených objektů. Mezi tyto vlastnosti patří: pozice jednotlivých vertexů, pozice textur, normály a barvy. Pro přehlednost a snadnou manipulaci uchováváme reference na jednotlivé vlastnosti uvnitř objektů třídy `VertexBuffer`. `VertexBuffer` skrývá referenci na jedeno pole `VertexArray` s informacemi o vertexech,

jedno pole s normálami a jedno pole s nastavením barev. Polí určujících pozici textur může být i více nebo naopak nemusí být použito vůbec.

`IndexBuffer` je abstraktní třída, která udává jakým způsobem budou z vetexů vytvářeny *submeshe*. Na této třídě je v současné implementaci JSR 184 založena jediná skutečná třída – `TriangleStripArray`. Jak již název napovídá, definuje vytvářené objekty jako **pás trojúhelníků**. První tři vertexy vytvoří první trojúhelník pásu. Další trojúhelník je tvořen vždy novým vertexem a dvěma posledními vertexy z předchozího trojúhelníku.

`Transformable`

Abstraktní třída definující základní transformace objektů a textur. Jsou na ní založeny dvě důležité třídy `Node` a `Texture2D`.

`Node`, `Group`, `World`, `Sprite3D`, `Camera`, `Light`, `Mesh` `MorphingMesh` a `SkinnedMesh`

Třída `Node` je opět abstraktní a představuje základ *uzlů grafické scény*. Dědí od ní všechny zbývající třídy uvedené v této sekci – tzv. *uzly* . `Camera` je uzel představující pomyslnou kameru, kterou snímáme scénu. Umožňuje nastavení perspektivní nebo paralelní projekce. Třída `Light` tvoří uzel pro simulaci různých zdrojů světla. Setkáme se zde se čtyřmi druhy světel: okolní (ambient), směrové (directional), všesměrové (omnidirectional) a bodové (spot). `Mesh` je uzel, který definuje 3D objekt jako povrch sestavený z polygonů. Povrch je definován nám již známými třídami `VertexBuffer`, `IndexBuffer` a dále popsanou třídou `Appearance`. Odvozené třídy `MorphingMesh` a `SkinnedMesh` umožňují speciální transformace. `Sprite3D` je uzel reprezentující 2D obrázek uvnitř 3D scény. Používá se např. pro tvorbu částicových efektů či náhradu plnohodnotných trojrozměrných objektů "lacinější" dvojrozměrnou variantou.

`Group` je uzel sloužící k uchování skupiny uzlů uvnitř jednoho celku. Od něj je odvozena třída `World`, která se používá k uchování scény jako celku. Viditelnou část této scény je pak možné jednorázově vykreslit dříve popsanou metodou `Graphics3D.render()`.

`Material`, `PolygonMode`, `CompositingMode`, `Fog`, `Appearence` a `Texture2D`

Tyto třídy slouží k definici atributů použitých při vykreslování objektů typu `Mesh` a `Sprite3D`. `Material` obsahuje informace nezbytné pro nasvětlování objektu, jako jsou barva povrchu či jeho lesk. `PolygonMode` uchovává atributy zpracování jednotlivých polygonů. Mezi důležité patři volba nasměrování polygonu, odstranění jedné z jeho stran, stínování a další. `CompositingMode` určuje atributy pro zpracování na úrovni pixelů. Definuje způsob řešení potíží s polygony, jejichž plochy se navzájem prolínají a způsobovali by tak nežádoucí problikávání. Rovněž se stará o atributy nezbytné pro výpočty průhlednosti (blending) a umožňuje aplikaci *spritů* neboli obtisků (např. nečistoty, krev, průstřely na stěnách apod.). `Fog` je třída nesoucí atributy pro vytváření iluze mlhy. Lze nastavit její barvu, hustotu, vzdálenost a lineární či exponenciální charakter.

Dostali jsme se k poslední třídě této kategorie s názvem `Texture2D`. Ta uchovává informace o dvojrozměrném obrázku představujícím materiál. Dále uchovává jeho vlastnosti, jako je způsob aplikace na povrch (submesh), průhlednost, filtrování a další. Obrázek samotný je uchován třídou `Image2D`, jeho rozměry musí být mocninou 2 ale nemusí být čtvercový.

`Loader`

Třída slouží k nahrávání zdrojů (např. z balíčku *.jar). Nelze ji instanciovat, poskytuje však dvě metody `load()` pro nahrávání zdrojů.

## 3.2 MascotCapsule

*MascotCapsule 3D Rendering Engine*[14] (dále jen MascotCapsule) je vyvíjen japonskou společností HI CORPORATION[6]. Jedná se o knihovnu umožňující poměrně snadnou implementaci 3D aplikací nejen v prostředí Java ME. Společnost HI CORPORATION poskytuje tuto knihovnu výrobcům nejrůznějších přenosných zařízení. Z výrobců mobilních telefonů podporujících technologii MascotCapsule je nejznámější firma Sony Ericsson. Naprostá většina přístrojů tohoto výrobce, počínaje modely podporujícími JP-3 [2], má vestavěnou podporu jak JSR 184 tak i MascotCapsule V3. Verze V3 této knihovny nevyžaduje a ani nepodporuje žádný speciální hardware pro akceleraci grafiky. Nová verze MascotCapsule V4 přináší i podporu čipů pro grafickou akceleraci, jako je např. ATI IMAGEON 2300 určený pro mobilní zařízení. V následujícím textu se budeme zabývat dnes nejrozšířenější verzí V3. Ta přináší proti předchozí značně limitované a nepříliš rozšířené verzi V2 řadu vylepšení.

Velmi významnou vlastností enginu je práce výhradně s celočíselnými datovými typy. Všechny operace jako nastavování transformačních matic, zadávání úhlů a pozic bodů probíhá na celočíselné úrovni. Pozice v rámci 3D světa jsou pak v rozmezí hodnot -32768 až +32767 pro každou z os, pozice na obrazovce v rozmezí -2048 až +2047 a úhly jsou zadávány v rozsahu -2880 až +2880. Jako drobná nepříjemnost se může jevit občasná nutnost přepočítávat hodnoty (zejména úhly). Použití celočíselné logiky se však velmi pozitivně projevuje na výkonu celého enginu.

Stejně jako JSR 184 i MascotCapule poskytuje nízkoúrovňový a vysokoúrovňový režim pro práci s grafickými objekty. Oba režimy je rovněž možné kombinovat.

### 3.2.1 Modely

Ve vysokoúrovňovém režimu MascotCapsule umožňuje sestavení scény z více modelů ve formátu *.mbac. Model může být doplněn animacemi ve formátu *.mtra. Model i animace si máme možnost připravit v některé z aplikací pro 3D modelování. Příslušné pluginy pro export modelu do formatu *.bac a animací *.tra jsou dostupné pro aplikace 3ds Max, Maya, LightWave3D, Softimage, Blender, Photoshop a IIIustrator. Soubory získané exportem jsou v tzv. *intermediate formátu* a je potřeba je ještě překonvertovat utilitou M3DConverter do formátů *.mbac a *.mtra, které již dokáže MascotCapsule číst.

Engine podporuje *dynamické polygony* (někdy též nazývané *submodely*) umožňující za běhu aplikace přepínat mezi několika variantami určité části modelu. Tím lze dosáhnout např. efektu mrkajících očí apod.

Protože se MascotCapsule zaměřuje zejména na mobilní zařízení, jsou animace implementovány jako *skeletální*. Což znamená, že se uchovávají pouze informace o změnách pozice *kostí* pomyslné kostry modelu a nikoliv o pohybu všech polygonů modelu. Tento způsob animací vede ke značným úsporám paměti, zejména v okamžiku kdy máme na scéně více modelů. Dalším pozitivem je fakt, že můžeme jednu a tu samou animaci (či sadu animací)

[2] Java Platform 3 je konkrétní implementace Javy poskytující specifikou množinu knihoven

aplikovat na více podobných modelů, ať už v modelovacím programu nebo v přímo v mobilní aplikaci, což je podporováno již zmíněnými dynamickými polygony.

### 3.2.2 Textury

Textury jsou načítány v bitmapovém formátu *.bmp. Obrázek musí mít barevnou hloubku 8bitů (256 barev) a maximální rozměry $256 \times 256$ bodů pro běžné textury, kterých může být načteno až 16. Textura pro odrazy prostředí (enviroment map) může mít rozměry max $64 \times 64$ bodů a může být pouze jedna. I když formát *.bmp průhlednost nepodporuje, pokud ji v MascotCapsule povolíme, je automaticky barva 0 z palety 256 barev reprezentována jako zcela průhledná. Nekomprimovaný formát *.bmp by se nemusel zdát pro použití v mobilních zařízeních s omezenou pamětí vhodný, avšak je potřeba si uvědomit, že při nasazení aplikace dochází ke kompresi všech zdorjů do balíčku *.jar. Další komprese by tedy byla poměrně zbytečná.

Oproti předchozím verzím je také příjemný fakt, že můžeme na jeden model aplikovat textury z více souborů.

### 3.2.3 Zobrazení

MascotCapsule V3 nově podporuje oba dva běžné typy projekce, mezi kterými je možné libovolně přepínat. Jedná se o projekci paralelní a perspektivní.

Dále podporuje následující techniky:

- nasvětlování scény
- částečná průhlednost (semi-transparency) s pevně nastavenou hodnotou 50%
- aditivní zobrazení, neboli sčítání barev RGB modelu
- substraktivní zobrazení, neboli odčítání barev RGB modelu
- odrazy prostředí (specular) s využitím již zmínění enviroment mapy
- barevné polygony bez použití textury
- přepínání jednostranných/oboustranných polygonů

### 3.2.4 Třídy rozhraní MascotCapsule V3

V následujícím textu se budeme zabývat popisem 10 tříd, které rozhraní MascotCapsule V3 poskytuje. Podrobný popis API rozhraní se může lišit dle konkrétní implementace. Varianta pro mobilní telefony Sony Ericsson je popsána v dokumentu *API reference of com.mascotcapsule package for Sony Ericsson*, který je spolu s ostatními dokumenty dostupný na síti *MascotCapsule Developer Network (MCDN)*[14].

`Graphics3D`

Jedná se o třídu poskytující metody pro kreslení objektů a primitiv. Před vlastním kreslením je potřeba specifikovat objekt, na který bude kreslení probíhat, metodou `bind()`. Po ukončení renderování je potřeba tento objekt opět uvolnit metodou `release()`.

Pro vykreslení modelu ve formátu *.mbac slouží metoda `renderFigure()` a pro kreslení primitiv metoda `renderPrimitives`. V případě komplikovanějšího projektu je možné použít metodu `drawCommandList()`, která mimo vykreslení modelu, poskytuje současně i možnost měnit řadu nastavení, čímž lze redukovat počet volaných metod v renderovacím procesu. Kromě samotného modelu či specifikace primitiv očekávají tyto metody alespoň další dva parametry. Jedním z nich je objekt třídy `FigureLayout` určující transformaci objektu v rámci scény. Dalším důležitým parametrem je objekt třídy `Effect3D` umožňující nastavení světelných podmínek, stínů a průhlednosti.

Vykreslení stávající scény vyvoláme metodou `flush()`. Metodu obvykle voláme pouze jednou po přípravě všech prvků scény, avšak pokud potřebujeme vynutit vykreslení prvků scény v určitém pořadí můžeme tuto metodu v renderovacím procesu použít i vícekrát.

`Util3D`

Třída poskytuje statické metody pro výpočty odmocniny `sqrt()` a goniometrických funkcí sinus `sin()` a kosinus `cos()`. Jedná se o upravené celočíselné varianty uzpůsobené pro snadnou spolupráci s ostatními třídami tohoto rozhraní.

`Vector3D`

Jak název napovídá, třída reprezentuje vektor v trojrozměrném prostoru. Opět jde o celočíselnou variantu. Jednotkovému vektoru (s délkou 1.0) zde odpovídá vektor s délkou 4096. Normalizace vektoru na jednotkový můžeme dosáhnout voláním metody `unit()`.

`AffineTrans`

Třída `AffineTrans` představuje celočíselnou transformační matici o rozměrech $3 \times 4$. Poskytuje operace, jako rotaci `setRotation()`, násobení `mul()` nastavení na jednotkou matici `setIdentity()`, transformaci vektoru danou maticí `transform()` a další.

`Texture`

Tato třída umožňuje načtení textury ze souboru ve formátu *.bmp případně z pole bytů. Texturu je pak možné použít jako materiál nebo enviroment mapu na libovolný model nebo modely. Soubor či pole s texturou se specifikuje již při vytváření objektu této třídy, jako parametr konstruktoru, a dále je již neměnný.

`Figure`

Třída `Figure` nese informace o 3D modelu. Model je možné načíst ze souboru ve formátu *.mbac nebo z pole a to již při vytváření objektu. K vytvořenému modelu můžeme přiřadit texturu či sadu textur metodou `setTexture()` s příslušnými parametry. Výběr jedné konkretní sady je pak kdykoliv možné provést metodou `selectTexture()`. Metodou `setPattern()` pak můžeme vybrat konkrétní skupinu dynamických polygonů (submodel).

`FigureLayout`

`FigureLayout` ukrývá informace nezbytné pro vykreslení modelu reprezentovaného

třídou `Figure` nebo sestaveného pomocí primitiv. Můžeme specifikovat základní pozici na dvourozměrné obrazovce metodou `setCenter()`. Dále důležité vlastnosti jako pozici, rotaci a měřítko v rámci 3D světa metodou `setAffineTrans()` s příslušnou transformační maticí nebo maticemi jako parametrem. Dále máme možnost individuálně nastavit vlastnosti modelu vztažené k perspektivnímu zobrazení metodami `setPerspektive()`. Obdobně máme možnost nastavit i vlastnosti zobrazení paralelního metodou `setParalelSize()`.

`ActionTable`

Třída zapouzdřuje animace načtené ze souboru ve formátu *.mtra nebo z pole. Do animace jako takové nemáme možnost zasahovat můžeme však získat důležité parametry, jako je počet akcí, které animace obsahuje, metodou `getNumActions()`. Rovněž máme možnost zjistit počet snímků konkrétní akce použitím metody `getNumFrames()`.

`Effect3D`

Slouží pro nastavení efektů aplikovaných na objekt (objekty) při renderování. Nastavit můžeme vlastnosti stínování metodami `setShadingType()` a `setToonParams()`. Máme možnost povolit či zakázat částečnou průhlednost objektu metodou `setTransparency()`. Rovněž můžeme nastavit nasvětlení konkrétním světelným zdrojem `Light`.

`Light`

Objekt této třídy reprezentuje světelný zdroj v prostoru. Možnosti nasvětlení scény jsou bohužel poměrně omezené. Můžeme použít současně pouze jedno světlo okolí (ambient) a jedno světlo směrové. U směrového světla můžeme specifikovat vektor vyzařování `setParallelLightDirection()` a intenzitu vyzařování `setParallelLightIntensity()`. Světlo okolí umožňuje nastavit pouze druhou jmenovanou vlastnost metodou `setAmbientIntensity()`.

## 3.3 Srovnání

Volbu rozhraní pro práci s trojrozměrnou grafikou ovlivňuje několik aspektů. Především se jedná o typ zařízení, pro které aplikaci vyvíjíme.

V předchozí kapitole jsme získali přehled o knihovnách JSR 184 a MascotCapsule. Nyní si tedy můžeme ukázat přímé srovnání, jak by vypadala implementace jednoduché aplikace a řešení dílčích problémů užitím jedné i druhé knihovny.

### 3.3.1 Porovnání práce s primitivy

#### Vertexy a normály

Při sestavování objektů z grafických primitiv nejprve musíme specifikovat pozice jednotlivých vertexů, které budou tvořit vrcholy polygonů. Hodnoty zadáváme do jednorozměrného pole, kde vždy trojce hodnot tvoří souřadnice jednoho vertexu. V praxi bychom asi tyto souřadnice získávali z datového souboru, obsahujícího geometrii jednotlivých objektů (např. herní mapy). Obdobným způsobem musíme specifikovat i normálové vektory jednotlivých polygonů.

### a) MascotCapsule

O vykreslovaní se v MascotCapsule stará třída `Graphics3D`. Pro kreslení primitiv je určena její metoda `renderPrimitives()` s parametry:

```
renderPrimitives( Texture tex, int x, int y, FigureLayout layout,
  Effect3D effect, int cmd, int nPrim, int[] nVtx, int[] nNorm,
  int[] nTex, int[] nCol )
```

Je vidět že metoda přímo očekává jednorozměrné pole `nVtx` s vertexy a jednorozměrné pole normál `nNorm`. Jediné o co ve spojitosti s geometrií objektu musíme postarat je příprava těchto polí:

```
int [] nVtx = {
  +32, +32, 0, -32, +32, 0, +32, -32, 0,
  -32, +32, 0, +32, -32, 0, -32, -32, 0 };
int [] nNorm = {
  0, 0, 4096, 0, 0, 4096, 0, 0, 4096,
  0, 0, 4096, 0, 0, 4096, 0, 0, 4096 };
```

### b) JSR 184

V JSR 184 slouží k vykreslovaní objektů metody `render()` třídy `Graphics3D`. Varianta pro renderování primitiv má následující parametry:

```
render( VertexBuffer bufVtx, IndexBuffer bufIdx,
  Appearance appearence, Transform trans )
```

Pro JSR 184 tedy musíme vertexy a normály nejprve uložit do samostatných objektů třídy `VertexArray`. Potom vytvoříme společný `VertexBuffer`, který zapouzdřuje obě tato pole, která k sobě logicky patří.

Navíc se musíme postarat o vytvoření `IndexBufferu`, který, jak již bylo řečeno, slouží k definici, jakým způsobem budou z vertexů vytvářeny primitiva. Jediným implementovaným potomkem této virtuální třídy je `TriangleStripArray`.

```
short [] nVtx = {
  +32, +32, 0, -32, +32, 0, +32, -32, 0,
  -32, +32, 0, +32, -32, 0, -32, -32, 0 };

int nVtxComp = 3;
int nVtxCnt = nVtx.length / nVtxComp;
VertexArray arrayVtx = new VertexArray( nVtxCnt, nVtxComp, 2 );
arrayVtx.set( 0, nVtxCnt, nVtx );

byte [] nNorm = {
  0, 0, 4096, 0, 0, 4096, 0, 0, 4096,
  0, 0, 4096, 0, 0, 4096, 0, 0, 4096 };
```

```
int nNormComp = 3;
int nNormCnt = nNorm.length / nNormComp;
VertexArray arrayNorm = new VertexArray( nNormCnt, nNormComp, 1 );
arrayNorm.set( 0, nNormCnt, nNorm );

VertexBuffer bufVtx = new VertexBuffer();
bufVtx.setPositions( arrayVtx, 1.0f, null );
bufVtx.setNormals( arrayNorm );

int [] nStripVtxCnt = { 4 };
IndexBuffer bufIdx = new TriangleStripArray( 0, nStripVtxCnt );
```

### Textury

Pro aplikaci textur na skupinu primitiv musíme učinit dva hlavní kroky. Vhodným způsobem musíme načíst požadovaný obrázek s texturou a nastavit jeho umístění na jednotlivých polygonech pomocí pole s patřičnými hodnotami.

### a) MascotCapsule

V případě MascotCapsule tedy vytvoříme obdobné pole jako při specifikaci vertexů a normál v minulém kroku. Rozdíl spočívá v tom že pozice textury je pro každý vertex určena pouze souřadnicemi x a y v souřadném systému textury. Počet prvků pole je tedy dán vztahem *počet primitiv* $\times$ *počet vrcholů primitiva* $\times$ *2*. Pro naše dva polygony se třemi vrcholy bude počet hodnot $2 \times 3 \times 2 = 12$. Obrázek načteme jednoduše tak, že jeho název uvedeme jako první parametr při vytváření objektu třídy `Texture`.

```
int [] nTex = {
  192, 64, 128, 64, 192, 128,
  128, 64, 192, 128, 128, 128 };
Texture tex = new Texture( "/texture01.bmp", true );
```

### b) JSR 184

U JSR 184 jsme v předchozím kroku použili třídu `TriangleStripArray`, která při vykreslování skládá polygony tak, že dva vertexy jsou pro sousední polygony společné. Počet prvků pole je tak redukován na $4 \times 2 = 8$. Opět následuje nezbytné zapouzdření pole třídou `VertexArray`. Tuto třídu můžeme již přímo použít pro nastavení pozice textury `VertexBuffer`u metodou `setTexCoords()`.

Obrázek načteme při vytváření objektu třídy `Image`. Třída `Image2D` definuje v jakém formátu má být obrázek reprezentován. V našem případě jde o obyčejný formát RGB bez alpha kanálu. Rovněž texturu `Texture2D` vytvoříme bez speciálních požadavků na filtrování či průhlednost.

Finální vzhled našeho objektu složeného z polygonů je při renderování definován objektem třídy `Appearance`. Jeho metodou `setTexture()` nastavíme aby při zobrazení byla použita právě vytvořená textura. Nakonec musíme vytvořit objekt třídy `Material` definující chování povrchu našeho objektu při nasvětlování scény. Tento materiál rovněž nastavíme u objektu `Apearance` tentokrát metodou `setMaterial()`.

U materiálu jsme metodou `setVertexColorTrackingEnable()` nastavili aby ambientní složka při vykreslování byla brána nikoliv z textury ale z příslušného `VetexBufferu`. Dále jsme nastavili barvu odrazivé (specular) složky a světlost textury. Nyní již má objekt třídy `Apearance` dostatečné informace pro námi požadované zobrazení objektu.

```
short [] nTex = {
  192, 64, 128, 64, 192, 128, 128, 128 };

int nTexComp = 2;
int nTexCnt = nTex.length / nTexComp;
VertexArray texArray = new VertexArray( nTexCnt, nTexComp, 2 );
texArray.set( 0, nTexCnt, nTex );
bufVtx.setTexCoords( 0, texArray, 128.0f, null );

Image img = Image.createImage( "/texture01.png" );

Image2D img2D = new Image2D( Image2D.RGB, img );
Texture2D tex2D = new Texture2D( img2D );

Material mat = new Material();
mat.setVertexColorTrackingEnable( true );
mat.setColor( Material.SPECULAR, 0xFFFFFFFF );
mat.setShininess( 100.0f );

Appearance appearance = new Appearance();
appearance.setTexture( 0, tex2D );
appearance.setMaterial( mat );
```

### Kamera a perspektiva

#### a) MascotCapsule

MascotCapsule nedisponuje třídou reprezentující kameru, místo toho patřičně transformujeme objekty scény. Každý objekt na scéně má své rozložení definované objektem třídy `FigureLayout`. Jeho konečné umístění definujeme skládáním transformací `AffineTrans`. V praxi se obvykle setkáme se skládáním alespoň 2 transformací. Jedna bude reprezentovat posunutí objektu vůči ostatním objektům scény – např. rozmístění předmětů po herní mapě. Další trasformace bude představovat pohled na scénu neboli polohu kamery. V našem případě máme pouze jeden objekt a vystačíme si pouze s transformací scény (objekt `transView`). Nepřítomnost objektu zastupující kameru je v MascotCapsule vynahrazena speciální transformační metodou `AffineTrans.lookAt()`. Ta nám jako své parametry umožní zadat místo, ze kterého se na scénu "díváme", vektor udávající směr pohledu a natočení kamery kolem osy procházející "objektivem" (podobný efekt jako bychom nakláněli hlavu ze strany na stranu).

U vlastního rozložení objektu `FigureLayout` nejprve nastavíme perspektivní zobrazení metodou `setPerspektive()`. První dva parametry udávají vzdálenost přední a zadní ořezávací roviny. Poslední parametr specifikuje úhel pohledu neboli FOV (Field Of View), který jsme nastavili na hodnotu 512 což odpovídá 45°. Metodou `setCenter()` jsme vycentrovali rozložení na střed dvojrozměrné obrazovky. Nakonec jsme aplikovali nachystanou transformaci voláním `setAffineTrans()`.

```
Vector3D vecPos = new Vector3D( 0, 0, 128 );
Vector3D vecLook = new Vector3D( 0, -2048, -4096 );
Vector3D vecUp = new Vector3D( 0, 4096, 0 );

AffineTrans transView = new AffineTrans();
transView.setIdentity();
transView.lookAt( vecPos, vecLook, vecUp );

FigureLayout lay = new FigureLayout();
lay.setPerspective( 16, 4096, 512 );
lay.setCenter( getWidth()/2, getHeight()/2 );
lay.setAffineTrans( transView );
```

### b) JSR 184

V JSR 184 máme k dispozici třídu `Camera` obstarávající jednorázovou transformaci scény a nastavení perspektivy. K nastavení perspektivní kamery slouží metoda `setPerspektive()`. Jako první zadáme FOV opět s hodnotou 45 stupňů. Další parametr je poměr stran (aspect ratio), který se v naší ukázce bude měnit v závislosti na poměru stran displaye. Poslední dva parametry udávají vzdálenost přední a zadní ořezávací roviny.

```
Camera cam = new Camera();
cam.setPerspective( 45.0f,
  ( float )getWidth()/ ( float )getHeight(), 1.0f, 1000.0f );
```

### Nasvětlení scény

### a) MascotCapsule

Světlo je zde reprezentováno třídou `Light`. Všechny potřebné parametry máme možnost nastavit při volání konstruktoru. Zadáme tedy směr vyzařování, intenzitu směrové i ambientní složky. Aplikace světla a dalších efektů probíhá při renderování předáním objektu třídy `Effect3D`. Metodou `setLight()` nastavíme aktivní světlo a pomocí `setShadingType()` zvolíme běžné stínování objektu.

```
Vector3D vecLightDir = new Vector3D( -128, 256, 512 );
int nLightIntenDir = 3730;
int nLightIntenAmb = 1626;

Light light =
  new Light( vecLightDir, nLightIntenDir, nLightIntenAmb );

Effect3D effect = new Effect3D();
effect.setShadingType( Effect3D.NORMAL_SHADING );
effect.setLight( light );
```

### b) JSR 184

Stejně je světlo pojmenované i v JSR 184. Není potřeba vytvářet speciální efekt pro světla a stínování, nastavení světelných podmínek totiž neprobíhá po objektech ale vztahuje se na celou scénu.

```
Light  light = new Light();
light.setColor( 0xFFFFFF );
light.setIntensity( 1.0f );
```

### Vykreslení scény z primitiv

Vlastní vykreslení probíhá v MascotCapsule i JSR 184 obdobně – voláním příslušné metody třídy `Graphics3D`. V JSR 184 obsluhujeme kameru a světla zvlášť, při použití MascotCapsule je nastavení kamery zahrnuto v parametru `lay` a nasvětlení v parametru `effect`.

### a) MascotCapsule

```
static final int nCmd =
  Graphics3D.PRIMITVE_TRIANGLES |
  Graphics3D.PDATA_NORMAL_PER_VERTEX |
  Graphics3D.PDATA_TEXURE_COORD |
  Graphics3D.PDATA_COLOR_NONE |
  Graphics3D.ENV_ATTR_LIGHTING |
  Graphics3D.PATTR_BLEND_HALF;

Graphics3D g3d = new Graphics3D();

// Kreslici smycka
g3d.bind( g );
g3d.renderPrimitives( tex, 0, 0, lay, effect, nCmd, 12,
  nVert, nNorm, nTex, { 0 } );
g3d.flush();
g3d.release( g );
```

### b) JSR 184

```
Transform trans3D = new Transform();
trans3D.postTranslate( 0.0f, 0.0f, 30.0f );

Graphics3D g3d = Graphics3D.getInstance();

// Kreslici smycka
g3d.setCamera( cam, tran3D );
g3d.resetLights();
g3d.addLight( light, trans3D );
g3d.render( bufVtx, bufIdx, appearance, null );
g3d.releaseTarget();
```

### 3.3.2 Porovnání práce s modely

Práce s modely je v obou rozhraních velmi jednoduchá.

#### a) MascotCapsule

V MascotCapsule slouží pro reprezentaci modelu třída `Figure`. Ta nám umožňuje přímé načtení modelu jehož název uvedeme jako parametr konstruktoru. Kreslení probíhá metodou `Graphics3D.renderFigure`. Jako první parametr předáme vykreslovaný model. Druhý a třetí parametr definuje pozici modelu na 2D obrazovce. Poslední dva parametry určují rozložení a efekty modelu, které jsme si popsali u práce s grafickými primitivy. Vykreslení více modelů současně dosáhneme opakovaným voláním této metody s příslušnými parametry.

```
Figure figure = new Figure( "/model01.mbac" );
Texture texture = new Texture( "/image01.bmp", true );
figure.setTexture( texture );

Graphics3D g3d = new Graphics3D();

// Kreslici smycka
g3d.bind( g );
g3d.renderFigure( figure, 0, 0, layout, effect );
g3d.flush();
g3d.release( g );
```

#### b) JSR 184

JSR 184 využívá pro nahrávání zdrojů (modely, textury apod.) globální `Loader`. K uchování načtených modelů slouží třída `Object3D`. Narozdíl od MascotCapsule nemusíme při vykreslování více modelů volat opakovaně metodu pro vykreslování. Můžeme použít třídu `World`, která slouží jako kontejner pro všechny modely scény. Objekt třídy `World` můžeme potom jednorázově vykreslit voláním přetížené metody `Graphics3D.render()`.

```
Object3D[] roots = Loader.load("/model01.m3g");
World myWorld = ( World )roots[ 0 ];

Graphics3D g3d = Graphics3D.getInstance();

// Kreslici smycka
g3d.bindTarget( g );
g3d.render( myWorld );
g3d.releaseTarget();
```

### 3.3.3 Závěrečné srovnání

Jak je patrné z předchozích kapitol, JSR 184 nám proti MascotCapsule nabízí širší paletu možností pro práci s 3D grafikou. Poskytuje nám více možností pro práci s texturami, grafickými efekty (např. průhlednost), nasvětlováním (jak počet světel tak možnost nastavit jejich parametry). Pohodlnější je také nastavení a práce s kamerou. Díky bohatšímu

rozhraní máme možnosti seskupovat příbuzné elementy (např. vertexy a normály polygonového povrchu), což jistě napomáhá přehlednosti složitějších projektů.

MascotCapsule v diskutované verzi V3 poskytuje poměrně jednoduché rozhraní. Nejvíce omezující jsou proti JSR 184 možnosti práce se světly (současná aplikace pouze jednoho ambientního a směrového světla) a práce s efekty (např. pouze 50% průhlednost). Poměrně komplikovaný se rovněž může zdát způsob transformací objektů a celé scény. Mezi hlavní výhody MascotCapsule však patří vysoký výkon, kterého dosahuje zejména díky celočíselné logice. Vystačíme-li s poskytovanými možnostmi, můžeme jako pozitivum chápat rovněž jednoduchost rozhraní, které si lze velmi rychle osvojit a soustředit se na tvorbu vlastní aplikace. Verze V4 některé nedostatky řeší a navíc jde s poskytovaným výkonem ještě dále – přináší totiž podporu grafických akcelerátorů.

# Kapitola 4

# Návrh

Jako ukázku práce s 3D rozhraním pro platformu Java ME jsem se rozhodl implementovat jednoduchou hru. Konkrétně se jedná o remake legendární hry Dyna Blaster do trojrozměrné podoby.

## 4.1 Dyna Blaster

Původní hra byla vytvořena společností Hudson Soft Company, Ltd. v letech 1991 a 1992 pro platformy Amiga, Atari ST a PC. Verze pro osobní počítač byla určená pro operační systém DOS.

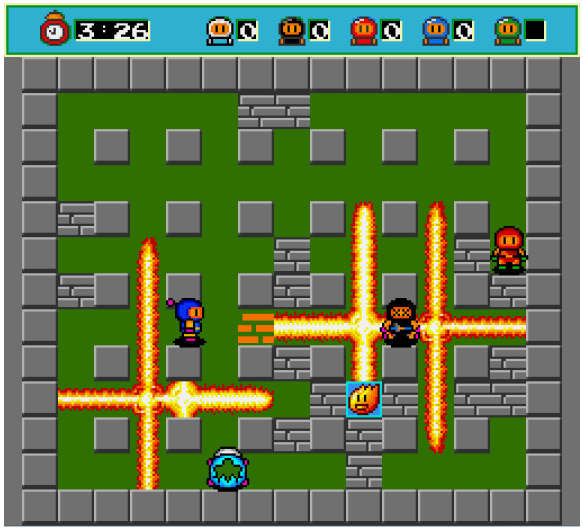

Obrázek 4.1: Hlavní menu hry Dyna Blaster

Jednalo se o hru v bludišti, kdy hráč měl za úkol projít celkem 8 úrovní po 8 kolech. Každá úroveň se odehrávala v prostoru s pevně danými rozměry a rozmístěnými statickými objekty. Navíc byla plocha vyplněna náhodně rozmístěnými objekty, které hráč mohl zničit umístěním výbušniny na sousední volná políčka. V bludišti se pohybovali počítačem řízení nepřátelé, které bylo potřeba v časovém limitu usmrtit rovněž pomocí výbušnin. Nepřátelé nemají k dispozici žádné zbraně, avšak hráč umírá pokud nestihne v čas uprchnout a nepřítel se dostane na stejné políčko. Úroveň bylo možné opustit pouze bránou skrytou pod některým ze zničitelných objektů a otevřela se až po eliminování všech nepřátel. Pod zničitelnými

překážkami bylo rovněž možné nalézt speciální vylepšení (takzvané power-upy), které hráči poskytli lepšení schopností. Hra nabízela vzhledem k datu vydání velmi jednoduché avšak přehledné rozhraní.


Obrázek 4.2: Dyna Blaster (PC): Ukázka ze hry jednoho hráče

Kromě hry jednoho hráče proti počítačovým protivníkům, byl k dispozici i multiplayer až pro 4 hráče na jednom počítači. Cílem pochopitelně bylo zabít protihráče šikovně nastraženou výbušninou. Multiplayer se stal ve své době velmi oblíbený, zejména kvůli možnosti hrát na jednom počítači bez nutnosti vytvářet sériové nebo síťové spojení mezi více stroji. Hrálo se na předem daný počet kol – hráč s nejvyšším score vyhrál.




Obrázek 4.3: Dyna Blaster (PC): Ukázka z multiplayeru 4 hráčů na jednom počítači

Hra je nyní obvykle řazena do kategorie *abandonware* a je možné ji získat na některém z portálů, které se zabývají abandonware hrami[1]. Pro běh v jiném systému než DOS či Windows 9x je potřeba použít některý z emulačních nástrojů (např. DOSBox[3] dostupný pro Windows série NT, Linux, Unix, MacOS a další).

## 4.2 Volba rozhraní

Při volbě rozhraní byl důležitý zejména poskytovaný výkon. Převedení hry typu Dyna Blaster do trojrozměrné podoby bude vyžadovat současné vykreslení poměrně velkého množství objektů. U závodních či FPS (First Person Shoter) her máme možnost napevno omezit dohlednost do dálky nastavením ořezávání v určité vzdálenosti před kamerou. Pokud není tato vzdálenost příliš krátká a překážky "nevyskakují" těsně před kamerou, hráči omezený výhled nijak nevadí. Z hlediska vykreslování scény by toto opatření znamenalo výrazné snížení počtu viditelných objektů. Hra Dyna Blaster se odehrává v bludišti, nad kterým (nebo jeho velkou částí) potřebuje mít hráč stálý přehled. Ořezávání nebo přílišné přiblížení kamery tak nepřichází v úvahu. Spojení všech objektů do jednoho celku není rovněž možné, potřebujeme totiž generovat bludiště různých rozměrů s náhodně rozmístěnými překážkami.

Protože výkon je silnou stránkou MascotCapsule a omezení v oblasti nasvělování scény a efektů není pro hru typu Dyna Blaster nijak kritické, rozhodnul jsem se právě pro toto rozhraní. Dostupnost MascotCapsule téměř výhradně na mobilních telefonech značky Sony Ericsson nepředstavovalo překážku, protože telefon tohoto výrobce vlastním a mám k dispozici i více rozdílných modelů pro testování.

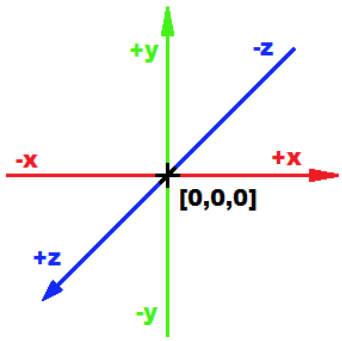
## 4.3 Návrh hry Blaster3D

Remake hry jsem pojmenoval *Blaster3D* aby byl patrný odkaz na původní hru a přitom nedošlo ke konfliktu s autorskými právy, pokud by byl název chráněn copyrightem či se jednalo o obchodní známku. Na základě volby rozhraní bude hra dostupá pro mobilní telefony Sony Ericsson s podporou MascotCapsule.

Základní principy Dyna Blasteru zůstanou zachovány. Pohyb bludištěm bude stále možný pouze ve 2 osách, avšak objekty tvořící prostředí budou trojrozměrné. Rovněž nepřátelé, postava hráče a ostatní předměty budou reprezentovány skutečnými 3D modely.

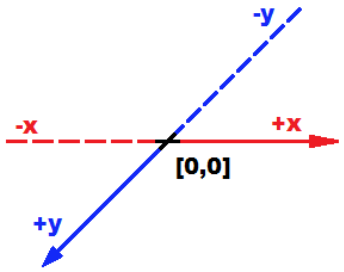
### 4.3.1 Rozložení scény

Svět vytvořený užitím MascotCapsule využívá souřadný systém reprezentovaný obrázkem 4.4. Pro usnadnění orientace bude pro herní elementy využita pouze kladná část os souřadného systému.




Obrázek 4.4: Systém souřadnic 3D světa

Tento systém bude však použit pouze pro vykreslování a transformace scény. Herní logika bude využívat pouze dvourozměrný diskrétní systém souřadnic. Tím usnadníme a zpřehledníme interakci mezi objekty scény. Jeho rozložení odpovídající původnímu 3D sytému vidíme na obrázku 4.5.




Obrázek 4.5: Systém souřadnic "diskrétního" světa

Záporné části os jsou vyznačeny čárkovaně, protože i herní logika bude využívat zejména části kladné. Vztah obou souřadných systémů si můžeme představit tak, že necháme splynou jejich počátky [0, 0, 0] a [0, 0]. Osy X navzájem splývají. Osa Z směřující v 3D systému k pozorovateli, je v diskrétním systému nahrazena osou Y. Diskrétní systém jednoznačně určuje polohu objektu na pomyslné šachovnici, zatímco systém trojrozměrný určuje skutečnou polohu objektu na scéně pro dosažení plynulých pohybů. Rozměr jednoho políčka šachovnice bude konstantní (`VELIKOST_POLE`). Rozměry podstavy jednotlivých modelů budou maximálně dosahovat rozměrů `VELIKOST_POLE` × `VELIKOST_POLE`. Výška objektu nebude omezena avšak je potřeba uvážit že model nesmí hráči překážet ve výhledu.

Přepočty z 3D systému do diskrétního systému (např. při určení diskrétní pozice pohybujícího se hráče) zachycuje následující vztah:

```
diskretni_pozice.x = skutecna_pozice.x \ VELIKOST_POLE
diskretni_pozice.y = skutecna_pozice.z \ VELIKOST_POLE
```

Výpočet opačným směrem, tedy z diskrétní pozice na pozici skutečnou (např. při vytváření objektů) definujeme jako:

```
skutecna_pozice.x = diskretni_pozice.x * VELIKOST_POLE + VELIKOST_POLE \ 2
skutecna_pozice.y = 0
skutecna_pozice.z = diskretni_pozice.y * VELIKOST_POLE + VELIKOST_POLE \ 2
```

Jak je vidět souřadnice Y bude vždy nulová, všechny objekty tedy budou svisle ležet v jedné rovině. Přírůstek `VELIKOST_POLE \ 2` je zde pro vycentrování objektu na střed políčka.

### 4.3.2 Herní prostředí

Objekty tvořící herní svět lze rozdělit do několika kategorií:

- statické objekty herní plochy
- zničitelné objekty herní plochy
- spawn objekty
- výbušniny
- pohyblivé objekty
- statické okolí herní scény

*Statické objekty herní plochy* představují překážku pro hráče, kterou nelze nijak zničit. Na pole s touto překážkou nemůžou pohyblivé objekty (hráč, NPC) za žádných okolností vstoupit.

*Zničitelné objekty herní plochy* se chovají naprosto stejně, až do okamžiku, kdy jsou zničeny výbušninou. Po zničení je objekt tohoto typu odstraněn z herní plochy a pole, na kterém se nacházel, stává přístupné pro pohyblivé objekty. Jeho zničením může dojít také k odhalení tzv. spawn objektu.

*Spawn objekt* je speciální typ objektu, který je skryt pod zničitelným objektem. Spawn objekt je pro hráče neviditelný až do okamžiku, kdy zaniká zničitelný objekt na daném poli. Spawn objektem může být *brána pro opuštění levelu* nebo *power-up*.

*Výbušnina* je objekt který může hráč vytvořit na jeho současné pozici stiskem příslušného tlačítka. jakmile hráč pole opustí, stává až do výbuchu nepřístupné pro pohyblivé objekty.

*Pohyblivé objekty* jsou objekty, které se mohou plynule pohybovat z políčka na políčko. Mezi tyto objekty se řadí hráčova postava a NPC (Non-Player Character).

*Statické herní okolí* jsou objekty, které se nacházejí mimo herní plochu. Tyto objekty nijak nezasahují do herní logiky, pouze vytváří okolí, kterým je herní plocha vizuálně omezena.

Rozložení statických objektů je bude načítáno ze speciálního souboru *.b3dm s mapou herního levelu. V mapě se rovněž bude nacházet startovací pozice pro postavu hráče (typicky levý horní roh). Zničitelné objekty budou po načtení mapy, náhodně rozmístěny na některé volné pozice. Počet takto rozmístěných objektů zaplní $\frac{1}{3}$ z celkového volného místa. Náhodně bude rovněž zvolena pozice brány a několika power-upů. Statické okolí vytvoříme automaticky tak, aby vizuálně připomínalo hradby. V rozích, kde hráči nebudou překážet ve výhledu, budou umístěny věže.

### 4.3.3 Soubor obsahující herní mapu

Mapa ponese informace o rozmístění statických objektů a počáteční pozici hráče. Uložena bude ve speciálním souboru s příponou *.b3dm (Blaster3D Map), jehož formát si nyní popíšeme. Soubor začíná 4 byty dlouhým identifikátorem obsahujícím ascii hodnoty znaků `B3DM`. Následuje 1 byte nesoucí číselné označení verze souboru (pozn. aktuálně popisována a implementována je verze 3). Dále jsou na řadě rozměry mapy X × Y uložené jako 2 čísla typu `short`. Hlavička je zakončena 1 bytem udávajícím počet nepřátel. Hlavička souboru je tedy celkem 10bytů dlouhá a může vypadat např. takto:

```
42 33 44 4D 03 00 0B 00 0B 03
```

Výše uvedený příklad je uvozen identifikátorem B3DM, následuje informace že daný *.b3dm soubor je verze 3. Dále jsou uvedeny rozměry herní mapy $11 \times 11$ polí a počet náhodně vygenerovaných nepřátel je nastaven na 3.

Po hlavičce souboru je očekáván blok dat o délce přesně $X \times Y$ bytů. Tento blok již obsahuje vlastní informace pro rozmístění objektů. Postupuje se po řádcích od levého horního rohu mapy. Jedno herní políčko je v souboru reprezentováno 1bytem. Řádků je celkem Y a každý má délku X bytů. Hodnoty kterých může políčko nabývat a jejich význam zahrnuje tabulka 4.1.

| Hodnota (hex) | Význam |
|---|---|
| `0x00` | Volné políčko (může být dodatečně obsazeno generátorem objektů) |
| `0x01` | Statická překážka |
| `0x02` | Zničitelný objekt (nepoužito, generováno automaticky) |
| `0x30` | Startovní pozice hráče |
| `0x31` | Startovní pozice nepřítele (nepoužito, generováno automaticky) |

Tabulka 4.1: Datové hodnoty pro jednotlivá herní políčka mapy *.b3dm

### 4.3.4 Pravidla hry

Cílem každého kola hry je zabít všechny nepřátele a opustit level bránou skrytou pod některým ze zničitelných objektů. Kromě tohoto úspěšného zakončení může nastat několik situací, které vedou k prohře hráče v daném kole:

- vypršení časového limitu
- usmrcení nepřítelem
- usmrcení výbušninou

Hráč má po spuštění nové hry k dispozici 3 životy. Prohra znamená že hráč přichází o jeden život a musí opakovat daný level. Při opakování je level uveden do výchozího stavu, včetně nového generování zničitelných objektů a restartování časomíry. Navíc jsou hráči odebrány všechny doposud získané power-upy. V okamžiku kdy je vyčerpán poslední život hra končí.

### 4.3.5 Výbušniny

Kolem výbušnin se točí všechny hry založené na Dyna Blasteru. Jsou jediným prostředkem pro ničení překážek a nepřátel. Ve hře Blast3D budou k dispozici celkem 2 typy výbušnin, jejichž vlastnosti lze navíc vylepšovat sbíráním power-upů. Hráč stiskem příslušné klávesy umístí bombu na vhodné políčko a vzdálí se, aby nebyl zasažen výbuchem. Výbuch zasáhne políčko, na které byla výbušnina umístěna a několik políček sousedních. Na sousední pole se výbuch šíří pouze vodorovně a svisle, nikoliv úhlopříčně. Počet sousedních polí ovlivněných výbuchem závisí na aktuálním dosahu výbušniny. Pokud výbuch v určitém směru zasáhne překážku, tak na daném poli zanikne, a dále se v tomto směru nešíří. Je-li výbuchem zasažena jiná výbušnina dojde rovněž i k její explozi.

První typ výbušnin je *časovaná bomba*. Exploze výbušniny proběhne automaticky po uplynutí konstantního časového okamžiku (2,5 s).

Sebráním příslušného power-upu jsou časované bomby nahrazeny pokročilejším typem výbušniny – *minami s dálkovým detonátorem*. K explozi nedochází automaticky ale až po stisku příslušné klávesy. Hráč tak může "na dálku" odpálit minu v pravý okamžik, kdy je nepřítel v jejím dosahu. V případě že bylo položeno více min, jeden stisk klávesy odpálí vždy jednu minu. Pořadí odpalování je od nejdříve položené po nejpozději položenou minu.

### 4.3.6 Power-upy

Power-upy slouží k vylepšování schopností hráče. Jsou skryty pod zničitelnými objekty. K jejich získání dojde v okamžiku, kdy hráč vstoupí na políčko kde se power-up nachází. Pokud je odhalený power-up zasažen výbuchem, dojde k jeho zničení. Blast3D nabízí celkem 3 různé power-upy a všechny vylepšují vlastnosti výbušnin, které má hráč k dispozici.

*Zvětšení rozsahu výbušniny* je power-up po jehož sebrání se zvětší rozsah exploze o jeden dílek ve všech 4 směrech. Po spuštění nové hry je rozsah aktuální pole + 1 dílek v každém ze 4 směrů. Maximální dosažitelná hodnota není nijak omezena.

*Zvýšení počtu výbušnin* udává kolik výbušnin může hráč současně umístit. Sebráním power-upu naroste tento počet o jednu výbušninu. Po spuštění nové hry může hráč umístit pouze jednu výbušninu současně. Počet rovněž není nijak limitován.

*Miny s dálkovým detonátorem* jsou posledním power-upem, po jehož sebrání jsou časované bomby nahrazeny zmíněnými minami.

Do každého levelu bude na náhodně zvolenou pozici umístěn jeden power-up náhodného typu. nalezení a sebrání power-upu není podmínkou pro úspěšné dokončení levelu. Rovněž jeho zničení není nijak kritické, hráč se pouze připraví o dodatečné schopnosti.

### 4.3.7 Brána

Brána slouží pro úspěšné opuštění levelu a po spuštění je skrytá pod některým ze zničitelných objektů. Bránou hráč projde automaticky, pokud vstoupí na její pole. Nejprve je však potřeba usmrtit všechny protivníky v daném kole, jinak hráč nebude schopen postoupit do dalšího levelu. Bránu nelze žádným způsobem zničit.

# Kapitola 5

# Implementace

Vývoj programové části proběhl v prostředí *Eclipse*[4] (celý název zní *Eclipse IDE for Java Developers*) s nainstalovaným pluginem *EclipseME*[5] pro přímou podporu platformy Java ME. Na systém bylo nutné nainstalovat vývojářský baliček *Java SE Development Kit*[7] (Verze 6 Update16). Instalace *Sony Ericsson SDK*[8] (verze 2.5.0.5) poskytnula podporu jednotlivých modelů mobilních telefonů této značky, včetně možnosti emulace a debugování na vzdáleném zařízení.

K osvojení konstrukcí jazyka Java a získání správných návyků mě pomohla kniha "Učebnice jazyka Java"[16]

Aplikace je určena pro zařízení splňující následující specifikaci:

- Java ME konfugurace CLDC verze 1.1
- profil MIDP verze 2.0
- podpora MascotCapsule verze v3
- display s rozlišením alespoň $176 \times 220$ bodů
- numerická klávesnice s klasickým rozložením a/nebo joystick (podporující pohyb ve 4 směrech a stisknutí)

## 5.1 MIDlet aplikace – třída Blast3D

Program je rozložen do několika zdrojových souborů, z nichž každý obsahuje implementaci jedné třídy. Základem je zde jako u každé Java ME aplikace tzv. MIDlet. `MIDlet` je třída jejíž kostruktor je volán po spuštění aplikace. V závislosti na událostech z vnějšku se může nacházet v jednom ze tří základních stavů. Po vytvoření se MIDlet nachází v *pasivním režimu* a čeká na automatickou aktivaci. K té dochází voláním metody `startApp()`. Při opětovném pozastavení je volána metoda `pauseApp`. Poslední stav je ukončení aplikace, kterému odpovídá metoda `destroyApp`.

Třída odvození od MIDletu je nazvána `Blast3D`. V jejím konstruktoru získáme instanci třídy `Display`, prostřednictvím které bude probíhat veškeré zobrazování. Dále vytvoříme rozšířené plátno `MainCanvas` v jehož kódu se nachází hlavní část aplikace (viz dále).

Při aktivaci MIDletu uvnitř metody `startApp()` přiřadíme instanci třídy `Display` naše plátno `MainCanvas`. Pro plátno vytvoříme vlastní vlákno `Thread`, které metodou `start()` spustíme.

## 5.2 Třída MainCanvas

`MainCanvas` je založen na třídě `GameCanvas`, která proti obyčejnému plátnu poskytuje rozšířené možnosti pro hry (celoobrazovkový režim, bufferované kreslení apod.). MainCanvas dále implementuje rozhraní `Runnable`, abychom běh plátna mohli spustit ve vlastním vlákně.

Mezi hlavní činnosti prováděné v konstruktoru patří načítání všech zdrojů, které aplikace vyžaduje ke svému běhu. Zejména se jedná o modely ve formátu *.mbac a textury *.bmp. Následuje přiřazení textur k jednotlivým modelům. Tento proces pro jeden model s jednou texturou zachycuje úryvek kódu 5.1.

```
// Pole pro ulozeni textur jednotlivych modelu
Texture [] texWallBrick = new Texture[ 2 ];
...

// Pokus o nacteni jednotlivych textur a modelu
try
{
  texWallBrick[ 0 ] = new Texture( "/brick01.bmp", true );
  ...

  m_figures[ Shared.FIGURE_WALL_BRICK ] =
    Figure( "/wall03x.mbac" );
  ...

}
catch ( IOException e )
{
  e.printStackTrace();
}

// Prirazeni textur k modelum
m_figures[ Shared.FIGURE_WALL_BRICK ].setTexture( texWallBrick );
...
```

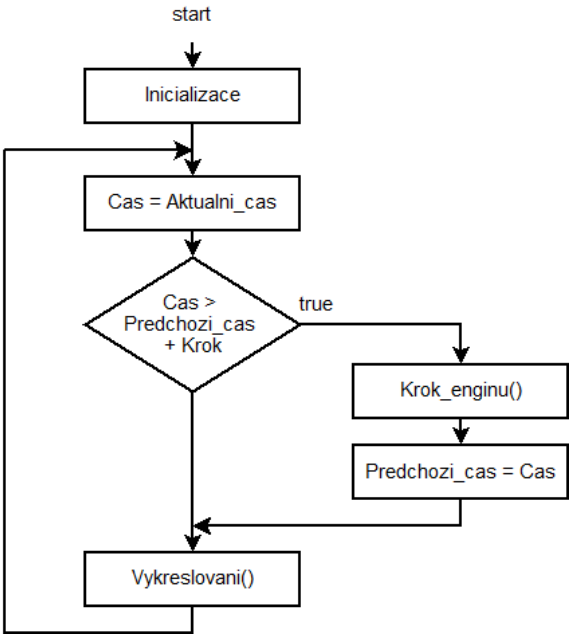
Tabulka 5.1: Načítání modelů a textur

MainCanvas dále obsahuje hlavní smyčku, ve které probíhá vykreslování 2D i 3D grafiky, obsluha vstupu uživatele, krok herního enginu a další.

### 5.2.1 Hlavní smyčka programu

Hlavní smyčka se nachází uvnitř metody `run()`. Schématicky je průběh smyčky znázorněn na obrázku 5.1.

V rámci inicializace je program přepnut do režimu zobrazení hlavního menu. Potom již začíná "nekonečná" smyčka. Prvním krokem je získání a uložení aktuálního času zařízení. K provedení kroku herního enginu dochází v pravidelných intervalech, aby se omezil vliv výkonnosti zařízení na rychlost hry. O provedení kroku je rozhodnuto na základě podmínky `Cas > Predchozi_cas + Krok`. Hodnota `Cas` zde představuje aktuální čas, `Predchozi_cas`




Obrázek 5.1: Hlavní smyčka programu

je čas kdy proběhl předchozí krok enginu a `Krok` je konstanta udávající časový interval mezi jednotlivými kroky [1]. Je-li podmínka splněna dochází k provedení dalšího kroku a k aktualizaci hodnoty `Predchozi_cas`. Nezávisle na tom, zda byl proveden krok enginu, dochází v každé iteraci k vykreslování 2D a 3D grafiky.

### 5.2.2 Krok herního enginu

Krok enginu v metodě `EngineStep()` začíná inkrementací herního času. Herním čas běží od 0 a jsou jím řízeny všechny časově závislé akce na obrazovce. Dále dochází k volání metody `HandleMovements()`, která se stará o aktualizaci pozice hráče, v závislosti na stisknutých směrových klávesách.

Ke slovu se dostává metoda `TransformScene()`. Jejím úkolem je nastavit transformační matici herního prostředí. Matice je nastavena dle pozice hráčovi postavy tak, aby po její aplikaci na herní svět, byla viditelná potřebná část herní plochy kolem hráče. Pozice kamery se počítá dle následujících vztahů:

```
Cam.x = Player.x + ( SceneSize.x / 2 - Player.x ) / 4
Cam.y = SceneSize.y
Cam.z = SceneSize.z / 2 + Player.x + ( SceneSize.z / 2 - Player.z ) / 3
```

`Cam` je pozice kamery, `Player` pozice hráče a `SceneSize` představuje skutečné rozměry herní plochy. Pokud by x-ová souřadnice kamery byla rovna přímo x-ové souřadnici hráče, docházelo by k nepříjemnému efektu, kdy hráč stojí v blízkosti levého či pravého kraje herní plochy. Byla by totiž viditelná pouze její malá část a přibližně polovinu obrazovky by

[1] V rámci ladění byla její hodnota stanovena na 25 ms.

zabírala prázdná plocha. Proto je kamera posunuta o přírůstek daný $\frac{1}{4}$ vzdálenosti hráče od středu scény. Aby bylo dosaženo "smysluplného" pohledu na scénu je z-ová souřadnice kamery posunuta o $\frac{1}{2}$ velikosti herní plochy před pozici hráče. I zde by ovšem docházelo k malému rozhledu hráče, pokud by se nacházel u horního či dolního okraje. Proto je navíc přičtena $\frac{1}{2}$ z odchylky pozice hráče od středu herní plochy. K vhodným hodnotám – $\frac{1}{4}$ pro x-ovou a $\frac{1}{2}$ pro z-ovou souřadnici – se dospělo při ladění programu. Y-ovoá souřadnice je konstatní s hodnotou 1600. Jak je vidět na obrázku 5.2, hráč má přehled nad velkou částí herní plochy, i pokud se nachází v jejím rohu.


Obrázek 5.2: Pohled na scénu s hráčem stojícím v rohu

Při spuštění nového kola je použit jednoduchý efekt, kdy je kamera nastavena velmi blízko k postavě hráče a plynule oddalována. Tohoto efektu je dosaženo násobením všech souřadnic kamery hodnotou rostoucí od 0,4 do 1,0. Efekt je aplikován uvnitř metody `TransformScene()`.

Po získání globální transformační matice dochází k aktualizaci všech objektů scény ve skaldu `ObjectStorage` (viz dále) voláním jeho metody `Update()`. Metoda mimo jiné zajistí patřičnou transformaci jednotlivých objektů. Transformační matice je samostatně aplikována na "podlahu" herní scény vytvořenou z grafických primitiv.

Na konci metody `EngineStep()` ještě dochází k aktualizaci hodin, které měří čas zbývající do konce kola. Pokud čas vypršel je volána metoda `GameOver()`, která zastaví herní engine a vyvolá zobrazení nápisu "GAME OVER".

### 5.2.3 Vykreslování grafiky

O vykreslování grafiky se stará metoda `paint()`. Metoda rozlišuje dva základní stavy programu – zobrazení menu a zobrazení herní scény.

V režimu zobrazení menu dochází pouze k volání metody `m_menu.Draw()`, která se postará o vykreslení aktuální nabídky `m_menu`, všech položek a zvýraznění položky aktuální. Pro texty menu je použit vlastní bitmapový font (viz dále).

Postup vykreslování trojrozměrné scény zachycuje kód 5.2.

```
// nastaveni barvy pozadi (mazani sceny)
g.setColor( m_nColBg );
g.fillRect( 0, 0, m_nWidth, m_nHeight );

g3d.bind( g );

// Kresleni "zeme" sestavene z primitiv
g3d.renderPrimitives( null, 0, 0, m_layGround, m_effect, COMMAND, 1,
  m_nVertGround, m_nNormGround, new int[ 0 ], m_nColorGround );

// Prenos do grafikceho bufferu
g3d.flush();

// Kresleni jednotlivych modelu sceny
for ( int nCnt = 0; nCnt < m_storage.m_nObjectCnt; nCnt++ )
{
  g3d.renderFigure(
    // Z ObjectStorage ziskame ID modelu
    m_figures[ m_storage.m_objects[ nCnt ].GetFigureID() ],
    0,
    0,
    // Z ObjectStorage ziskame rovnez rozlozeni jednotlivych modelu
    m_storage.m_objects[ nCnt ].GetLayout(),
    // efekt je spolecny pro celou scenu
    m_effect );
}

// Prenos sceny slozene z modelu do grafickeho bufferu
g3d.flush();

// uvolneni cile vykreslovani
g3d.release( g );
```

Tabulka 5.2: Vykreslování 3D scény

Před vykreslením každého snímku dojde ke smazání obrazovky. Mazání je realizováno nastavením renderovací barvy na barvu pozadí `m_nColBg` a následným vykreslením obdélníku přes celou obrazovku. Dalším krokem je vykreslení vlastní 3D scény. Hlavní část scény tvoří předpřipravené modely avšak v malé míře se uplatňuje i práce s grafickými primitivy. Konkrétně spodní část scény ("podlaha") je vytvořena z primitiv. K jejímu vykreslení je použita metoda `g3d.renderPrimitives()`. Podlaha je samostatně přenesena do videobufferu voláním `g3d.flush()`, čímž je omezeno její "problikávání" přes ostatní prvky scény. Následuje nejdůležitější část renderovacího procesu – vykreslení jednotlivých modelů. Ty jsou dostupné v poli `m_figures`. Index modelu v poli je pro jednotlivé objekty scény získán ze skladu objektů voláním `m_storage.m_objects[ nCnt ].GetFigureID()`. Rozložení objektů (layout) je rovněž získáno ze skladu metodou `m_storage.m_objects[ nCnt ].GetLayout()`. Po přípravě vykreslení všech modelů dochází k jejich jednorázovému přenosu do videobufferu `g3d.flush()` a k uvolnění cíle vykreslování `g3d.release( g )`.

Po dokončení 3D renderovacího procesu se dostává ke slovu zobrazení dvojrozměrných prvků. Metoda `DrawHUD()` obstarává vykreslení HUDu (Head-Up Display). HUD zobrazuje hráči informace o probíhající hře. Zobrazení probíhá v textové podobě doplněné malými ikonami s využitím bitmapového fontu. V rámci `DrawHUD()` je volána vnořená metoda `DrawMessage()` pro zobrazení důležitých zpráv na střed obrazovky (nápisy "PAUSED", "GAME OVER" a "STAGE N-M").

### 5.2.4 Zachycení a zpracování vstupu uživatele

Hru je možné ovládat prostřednictvím kláves telefonu a/nebo joystickem. K zachycení vstupních událostí slouží metody `keyPressed()` a `keyReleased()`.

Při stisku klávesy je volána metoda `keyPressed()` s celočíselným parametrem reprezentujícím kód klávesy.

Pokud se hra nachází v režimu zobrazení menu, je kód klávesy předán metodě `menuKeyPressed()`. Chování metody je závislé na předaném kódu. Při stisku klávesy 8 nebo vychýlení joysticku směrem dolů dochází k výběru následující položky menu voláním metody `m_menu.SelectNext()`. Obdobně při stlačení klávesy 2 nebo vychýlení joysticku nahoru, dochází k výběru předchozí položky metodou `m_menu.SelectPrev()`. Aktivace vybrané položky je možná stiskem klávesy 5, stiskem joysticku nebo stiskem levé akční klávesy. Tato událost je aktuálnímu menu ohlášena voláním metody `m_menu.Action()`. Skutečná událost potom závisí na funkci konkrétní položky menu a je implementována metodou `Action()` u příslušného potomka třídy `MenuItem`.

Není-li zobrazeno menu, jsou události předány metodě `gameKeyPressed()`. Ta nejprve testuje, zda není aktivní "tvrdá pauza", znamenající konec hry. Pokud ano, dochází po stisku libovolné klávesy[2] k návratu do hlavního menu voláním `ShowMenu( m_mainMenu )`. Pokud byla uživatelem aktivována běžná pauza, je po stisku libovolné klávesy běh hry obnoven. Pokud nedošlo k žádné ze dvou uvedených situací a probíhá tedy normálním způsobem hra, jsou uloženy změny stavu směrových kláves do pole `m_bKeyStates` voláním metody `setGameKeyState()`. Toto pole je čteno při každém kroku herního enginu, a podle jeho obsahu je vyvolán pohyb hráče v určitém směru. Pokud byla stisknuta klávesa 5 nebo joystick dochází k pokusu umístit výbušninu na aktuální hrací pole metodou `PlantExplosive()`. Stisk levé akční klávesy způsobí voláním `ShowMenu( m_gameMenu )` zobrazení hlavního menu. Hvězdička pomocí `Pause( true )` pozastaví hru. Funkci má přiřazenu i klávesa mřížka, po jejímž stisku dojde k aktivaci miny [3]. Informaci o umístěných minách si udržuje sklad objektů. Odpálení "nejstarší" z nich, je realizováno metodou `m_storage.ActivateMine()`.

Pokud uživatel uvolnil některou z kláves, je volána metoda `keyReleased()`, jejímž parametrem je opět kód příslušné klávesy. Pokud se hra nachází v režimu zobrazení menu, je uvolnění klávesy ignorováno. Uvolnění klávesy v režimu hry vyvolá pouze modifikaci pole stavu směrových kláves `m_bKeyStates` voláním metody `setGameKeyState()`.

### 5.2.5 Načítání herního prostředí

Načítání levelu probíhá v metodě `LoadLevel()`. Nejprve je provedeno uvolnění zdrojů z případného předchozího levelu. Následuje vytvoření objektu třídy `Map` a načtení mapy ze

[2] Libovolnou klávesou rozumíme takovou klávesu, jejíž stisknutí vyvolá metodu `keyPressed()`. Některé telefony jsou vybaveny speciálními klávesami (např. spoušť fotoaparátu) které jsou pro Java aplikace transparentní.

[3] pokud hráč tento typ výbušniny umístil na některé hrací pole

souboru *.b3dm metodou `map.Read()`. Třída `Map` načítá informace ze souboru do vnitřního pole, odkud jsou potom dostupné přes příslušné metody. Je možné získat rozměry mapy `GetSizeX()` a `GetSizeY`, obsah políčka na konkrétní souřadnici `GetData()` a počáteční počet nepřátel `GetEnemyCnt()`. Dle získaných rozměrů jsou nastaveny parametry grafických primitiv pro vytvoření "podlahy" herní plochy. Rozměry mapy jsou použity také pro inicializaci skladu objektů. Do skladu jsou metodou `m_storage.CreateObject()` v cyklu přidávány jednotlivé objekty, jejichž typ je získán metodou `map.GetData()`. Dále dochází k náhodnému rozmístění nepřátel a skrytých objektů (power-upů a brány) metodou `m_storage.ObjectFieldGenerateRandom()`. Nakonec je zobrazen nápis o začátku nového levelu.

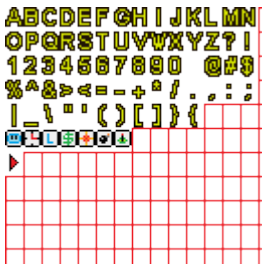
## 5.3 Sklad objektů

Třída `ObjectStorage` představuje sklad pro uchování jednotlivých objektů. Objektem rozumíme entitu virtuálního světa založenou na třídě `BaseObject`. Každý objekt má svůj vizuální model, rozložení (layout) specifikující přesnou pozici, informaci zda je objekt validní nebo má být smazán a některé další atributy. Sklad obsahuje pole `BaseObject [] m_objects` pro sekvenční uložení těchto objektů. Sekvenční uložení umožňuje snadný přístup k prvkům při renderování, bez volných pozic apod. Dále je k dispozici trojrozměrné pole `int [][][] m_nObjectField` představující 2D herní plochu s možností umístění i více objektů na jedno pole současně. Třetí rozměr pole `nObjectField` obsahuje indexy prvků uložených v sekvenčním poli `m_objects`. Toto trojrozměrné pole je využito pro pohodlné vyhodnocování herní logiky, kde by sekvenční přístup nebyl příliš vhodný.

`ObjectStorage` tedy poskytuje dva typy přístupu k objektům scény. Efektivní přístup pro renderování jsme si uvedli v části "Vykreslování grafiky". Pro herní logiku je určena sada metod umožňující operace nad konkrétní pozicí v mapě.

## 5.4 Bitmapový font

Bitmapový font je písmo, jehož jednotlivé znaky jsou uloženy v bitmapovém obrázku. Odtud jsou při výpisu textu vybírány a kopírovány na plátno. V aplikaci je použit jeden font uložený v obrázku `Font01.png` 5.3.




Obrázek 5.3: Obrázek obsahující znaky bitmapového fontu

Aby bylo možné s fontem pracovat, je potřeba znát dodatečné informace, jako je pozice jednotlivých znaků a rozměry znaku v pixelech. Pro určení pozice je použito pole znaků 5.3, kde pořadí prvků koresponduje s pořadím znaků v bitmapovém obrázku. Rozměry jsou pro všechny znaky stejné.

Obrázek fontu i s těmito informacemi jsou zapouzdřeny ve třídě `BitmapFont`. Vlastní vykreslování bitmapového fontu je realizováno uvnitř třídy `Graphics2D` metodou `DrawBitmapString()`. Hlavní část metody tvoří cyklus 5.4, probíhající v intervalu `<0..POCET_ZNAKU_RETEZCE)`. Nejprve jsou vypočítány souřadnice daného znaku v obrázku. Metodou `setClip()` je nastavena ořezávací oblast v rámci obrazovky. Od tohoto okamžiku se modifikace obrazu projeví pouze v rámci této oblasti. Souřadnice oblasti jsou dány výslednou pozicí znaku na obrazovce. Velikost oblasti je pevně dána rozměry jednoho znaku bitmapového fontu. Následuje vykreslení obrázku s fontem. Ten je posunut tak, aby do výřezu v obrazovce "zapadnul" požadovaný znak. Po skončení cyklu je obnovena původní ořezávací oblast, která umožňuje kreslení po celé obrazovce.

```
final static public String STR_CHARS =
  "ABCDEFGHIJKLMN" +
  "OPQRSTUVWXYZ?!" +
  "1234567890␣@#$" +
  "%^&><=-+*/.,:;" +
  "|_\\\"'()[]{}␣␣␣";
```

Tabulka 5.3: Pole určující pozice jednotlivých znaků v bitmapě

```
for ( int nCharCnt = 0; nCharCnt < cTextChars.length; nCharCnt++ )
{
  // Ziskani indexu znaku
  int nCharIndex =
    BitmapFont.STR_CHARS.indexOf( cTextChars[ nCharCnt ] );
  ...
  // Vyhledani znaku v bitmape
  int nCharBitmapX = ( nCharIndex % nCharsPerRow ) * nCharW;
  int nCharBitmapY = ( nCharIndex / nCharsPerRow ) * nCharH;

  // Nastaveni clip area na jeden znak a vyresleni znaku z bitmapy
  g.setClip( nCharX, nCharY, nCharW, nCharH );
  g.drawImage(imageFont, nCharX - nCharBitmapX,
    nCharY - nCharBitmapY, Graphics.TOP | Graphics.LEFT );

  nCharX += nCharW + 1;
}
```

Tabulka 5.4: Smyčk pro výpis znaků bitmapového fontu

Texty vytvořené užitím bitmapových fontů, jsou použity pro vykreslení položek menu a pro zobrazení HUDu. Náhled na menu je vidět na obrázku 6.1.

## 5.5 Objekty herní scény

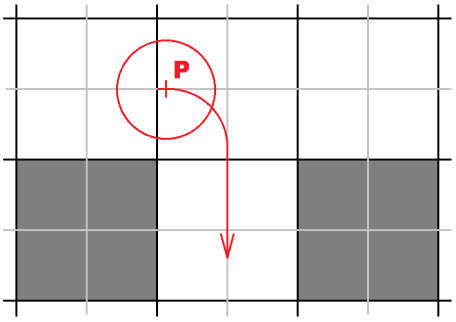
Kód jednotlivých objektů (entit) se nachází ve složce `Objects`. Základ každé entity tvoří třída `BaseObject`. Ta obsahuje proměnné pro uložení základních atributů objektu. Patří sem `Vector3D m_vecPos` pro uložení "diskrétní" polohy objektu a `Vector`

`m_vecGridPos` pro uložení polohy skutečné. Každý objekt scény má svůj model, jehož vlastnosti jsou definovány pomocí rozložení `FigureLayout m_layout` a celočíselné proměnné `int m_nFigureID` nesoucí index modelu.

Třída `BaseObject` poskytuje některé základní metody pro práci s objektem. Metodami `SetPos()` a `GetPos()` lze nastavit resp. zjistit přesnou pozici objektu a `GetGridPos()` vrátí diskrétní polohu v rámci hrací plochy. Metoda `Invalidate()` označí objekt jako neplatný. Neplatné objekty jsou při dalším kroku enginu odstraněny. Platnost objektu lze testovat voláním `IsValid()`. `GetType()` vrací typ objektu. Typy objektů jsou celočíselné konstanty s prefixem `OBJECT_` definované v souboru `Shared.java` (např. `OBJECT_WALL` značí nezničitelný objekt – "zeď"). Typ objektu je nastaven v konstruktoru potomka třídy `BaseObject`. Důležitá je metoda `Update()` volaná v každém kroku herního enginu. V jejím rámci dochází k aktualizaci rozložení modelu (layoutu) v závislosti na změně polohy. Tato metoda může být rozšířena v potomkovi. Metoda `Kill()` implementuje chování objektu při pokusu o jeho destrukci (např. při zasažení explozí).

### 5.5.1 Hráč

Postava hráče je reprezentována objektem třídy `Player`. Třída je rozšířena o metodu `Move( Vector vecDir )`, která je volána z při každém kroku herního enginu. Metodě je předán vektor reprezentující směr pohybu hráče. Volání probíhá uvnitř metody `MainCanvas.setGameKeyState()` (viz "Ovládání hry a menu"). Před samotným vykonáním pohybu je volním `m_storage.IsObstacle()` testováno zda se v daném směru nenachází překážka. Hráči je usnadněno ovládání tak, aby se při změně směru pohybu nemusel nacházet přesně v ose pole, na které hodlá vstoupit. Situaci znázorňuje obrázek 5.4. Hráč se snaží pohybovat směrem dolů, nenachází se ovšem přesně nad volným polem. V takovém případě je tolerována odchylka `ROZMER_POLE / 2` od středu pole. Postava hráče se tedy nepohybuje pouze v požadovaném směru, ale současně je provedena postupná korekce odchylky od osy pole. Na událost `Kill()` objekt reaguje odebráním jednoho života a restartováním levelu nebo ukončením hry (přišel-li hráč o poslední život).




Obrázek 5.4: Korekce pohybu hráče (označen písmenem P)

### 5.5.2 Nepřátelé

Ve hře Blaster3D je implementován jeden typ nepřítele `BaseNPC`. V rámci každého kroku enginu je volána metoda `Move()`, uvnitř které je řízen pohyb nepřítele. Předpokládejme že se pohybuje určitým směrem. Před vstupem na každé další políčko v daném směru, je metodou `m_storage.IsObstacle()` testováno, zda se na něm nenachází překážka. Pokud

je pole volné, nepřítel pokračuje plynule stejným směrem. V případě že se na poli nachází překážka proběhne hledání možných směrů pohybu. Pokud je více možností, než návrat na předchozí pole, je náhodně vybrána jedna z nich. Pokud není jiná možnost nepřítel se vrátí na pole ze kterého přišel. Při spuštění nové hry může nastat situace, kdy je nepřítel zablokován ve všech 4 směrech. V takovém případě jsou v každém kroku enginu testovány okolní pole, zda se některé z nich neuvolnilo. Při usmrcení nepřítele proběhne v metodě `Kill()` jeho invalidace a navýšení score o 100 bodů.

### 5.5.3 Zničitelné objekty

Zničitelné objekty jsou instancí třídy `Breakable`. Její implementace je velmi jednoduchá. Reimplementována je metoda `Kill()`. Při jejím volání dochází k zneplatnění objektu, takže bude v dalším kroku enginu ze scény odstraněn.

### 5.5.4 Statické objekty

Statické objekty nelze v průběhu hry nijak ovlivnit. Na událost `Kill()` nereagují a liší se pouze vizuálním modelem nastaveným v konstruktoru. Statické objekty použité v rámci herní plochy jsou třídy `Wall`. Okolí scény je tvořeno objekty tříd `OutWall` a `Tower`.

### 5.5.5 Výbušniny

Výbušniny jsou reprezentovány třídou `Explosive`. Chování výbušniny závisí na jejím typu. Pokud byl při vytváření typ výbušniny nastaven na `OBJECT_BOMB`, je v konstruktoru nastaven časovač na hodnotu `CASOVAC = AKTUALNI_CAS_ENGINU + 2500 ms`. V metodě `Update()` je pravidelně porovnávána hodnota časovače s časem enginu. Jakmile je splněna podmínka `AKTUALNI_CAS_ENGINU >= CASOVAC` je volána metoda `Explode()`. Pokud je výbušnina typu `OBJECT_MINE` je metoda `Explode()` vyvolána externě stiskem klávesy mřížka. Metoda `Explode()` zajistí volání metody `Kill()` pro všechny objekty v dosahu výbušniny (dosah je podrobněji popsán v části věnující se návrhu aplikace). Rovněž jsou na ovlivněných polích vytvářeny objekty třídy `Exp()`, představující plameny exploze. `Exp()` má podobně jako výbušnina omezenou životnost na 500 ms. Nakonec je volána metoda `Invalidate()`, čímž vynutíme odstranění výbušniny.

# Kapitola 6

# Ovládání aplikace

Po spuštění aplikace se na obrazovce zobrazí hlavní menu 6.1.


Obrázek 6.1: Hlavní menu

Menu lze ovládat pomocí numerických kláves nebo joystickem. Funkci jednotlivých ovládacích prvků telefonu shrnuje tabulka 6.1.

| Ovládací prvek | Funkce |
|---|---|
| Joystick dolů, NUM8 | Výběr následující položky |
| Joystick nahoru, NUM2 | Výběr předchozí položky |
| Stisk joysticku, NUM5, levá akční klávesa | Aktivace vybrané položky |

Tabulka 6.1: Funkce ovládacích prvků v rámci menu.

Položka “NEW GAME” vyvolá spuštění nové hry od prvního levelu (STAGE 1-1). Položka “EXIT GAME” slouží k ukončení celé aplikace. Prostřední volba “SELECT

LEVEL” vyvolá podmenu pro výběr levelu 6.2.


Obrázek 6.2: Menu pro výběr levelu

Výběrem některé z položek “ROUND1-STAGE1” – “ROUND1-STAGE8” dojde ke spuštění příslušného levelu. Aktivace položky “BACK” povede k návratu do hlavního menu.

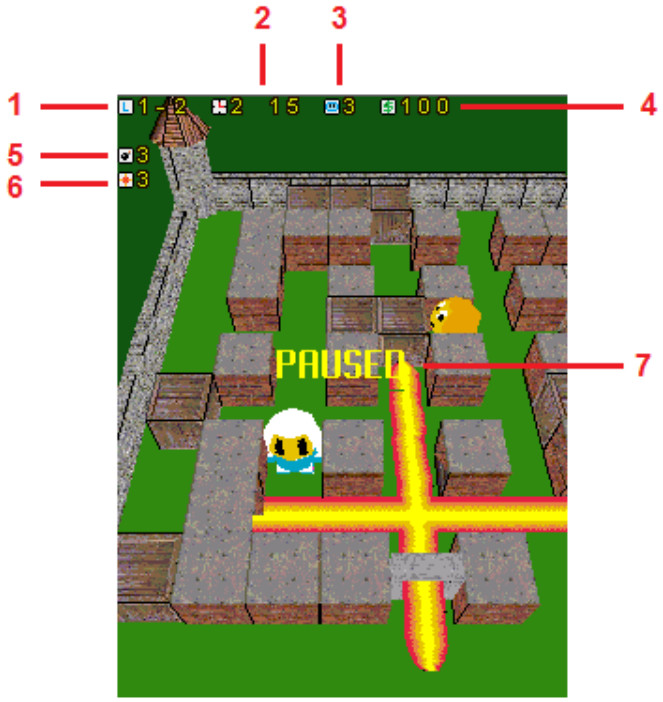
Po spuštění nového levelu některým z uvedených způsobů se hra přepíná do režimu zobrazení 3D scény. Konfigurace ovládání v tomto modu je uvedena v tabulce 6.2.

| Ovládací prvek | Funkce |
|---|---|
| Joystick dolů, NUM8 | Pohyb postavy hráče dopředu (dolů) |
| Joystick nahoru, NUM2 | Pohyb postavy hráče dozadu (nahoru) |
| Joystick vlevo, NUM4 | Pohyb postavy hráče vlevo |
| Joystick vpravo, NUM6 | Pohyb postavy hráče vpravo |
| Stisk joysticku, NUM5 | Umístění výbušniny na aktuální pole |
| Levá akční klávesa | Vyvolání herního (in-game) menu |
| NUM* (hvězdička) | Pozastavení hry (pauza) |
| NUM# (mřížka) | Odpálení dříve umístěné miny |

Tabulka 6.2: Funkce ovládacích prvků v rámci menu.

Při pozastavení hry klávesou NUM* zůstává zobrazena herní scéna doplněná o nápis “PAUSED”. Pauzu lze zrušit stiskem libovolné klávesy. Levou akční klávesou je možné v průběhu hry vyvolat herní (in-game) menu. Při vyvolání tohoto menu je aktuální hra automaticky pozastavena. In-game menu je vizuálně shodné s hlavním menu, navíc je zde pouze položka “RESUME GAME” pro návrat do rozehrané hry. Pokud je z herního menu spuštěna nová hra, je dosavadní postup v rozehrané hře bez upozornění ztracen.

O stavu běžící hry je hráč informován prostřednictvím informačního displaye – HUDu. Náhled na HUD včetně popisu jednotlivých elementů je uveden na obrázku 6.3.




Obrázek 6.3: Informační display – HUD, *Popis: 1) Číslo aktuálně rozehraného levelu, 2) Čas do konce kola, 3) Počet životů, 4) Score, 5) Počet současně umístitelných výbušnin, 6) Rozsah exploze výbušniny, 7) Zpráva zobrazená na středu displaye*

# Kapitola 7

# Závěr

Práce poskytla čtenáři úvod do architektury platformy Java ME. Popis rozhraní MascotCapsule a JSR 184. Uvedeno bylo jejich vzájemného srovnání včetně příkladu.

Navržená aplikace Blast3D byla zdárně implementována. Testování proběhlo na přístrojích Sony Ericsson k790i, k800i a k750i. Na modelech k790i a k800i hra běžela dostatečně plynule – rychlost vykreslování přesáhla 20 snímků za sekundu. Model k750i je vybaven výrazně pomalejším procesorem a již "od oka" bylo patrné, že běh aplikace není plynulý.

Hra proti předloze poskytuje sice z herního hlediska omezené možnosti, primárním cílem však byla realizace principů 3D grafiky na mobilních zařízeních. Mezi možná rozšíření by mohlo patřit doplnění všech vlastností původní hry, tedy $8 \times 8$ levelů odehrávajících se v rozličném prostředí, více typů nepřátel, zvukovou stránku hry apod. Z hlediska srovnání rozhraní pro 3D grafiku by jistě bylo zajímavé implementovat obdobnou aplikaci s využitím rozhraní JSR 184. Následovat by mohlo porovnáním výkonu, složitosti kódu, velikosti aplikace a dalších aspektů.

# Literatura


[1] Abandonia [online]. http://www.abandonia.com/games/75/DynaBlaster.htm.

[2] Building and Strengthening the Java Brand (dostupné na archive.com) [online]. http://web.archive.org/web/20051207112353/ http://java.sun.com/developer/technicalArticles/JavaOne2005/naming.html.

[3] DOSBox [online]. http://www.dosbox.com.

[4] Eclipse [online]. http://eclipse.org/.

[5] EclipseME [online]. http://eclipseme.org/.

[6] HI CORPORATION. http://www.hicorp.co.jp.

[7] Java Development Kit (JDK) [online]. http://java.sun.com/javase/downloads/index.jsp.

[8] Sony Ericsson - Developer World [online]. http://developer.sonyericsson.com/.

[9] 3D Engine: Gem Xenotime 3D [online]. http://www.mobygames.com/game-group/3d-engine-gem-xenotime-3d, 2005 [cit. 19. května 2010].

[10] Connected Limited Device Configuration 1.1 [online]. http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html, 2010 [cit. 19. května 2010].

[11] J2ME Building Blocks for Mobile Devices [online]. java.sun.com/products/cldc/wp/KVMwp.pdf, 2010 [cit. 19. května 2010].

[12] Java ME Technology [online]. http://java.sun.com/javame/technology/index.jsp, 2010 [cit. 19. května 2010].

[13] JSRs: Java Specification Requests - JSR 184 [online]. http://jcp.org/aboutJava/communityprocess/mrel/jsr184/index.html, 2010 [cit. 19. května 2010].

[14] MascotCapsule Developer Network (MCDN) [online]. http://www.mascotcapsule.com/en/index.php, 2010 [cit. 19. května 2010].

[15] Mobile Information Device Profile for Java$^{TM}$2 Micro Edition [online]. http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html, 2010 [cit. 19. května 2010].

[16] Herout, P.: *Učebnice jazyka Java.* Kopp, 2001, iSBN 80-7232-115-3.

[17] Mahmoud, Q. H.: *Naučte se Java 2 Micro Edition.* Grada Publishing, 2002, iSBN 80-247-0444-7.

[18] Topley, K.: *J2ME v kostce - Pohotová referenční příručka.* Grada Publishing, 2004, iSBN 80-247-0426-9.

# Příloha A

# Obsah CD

- Tento text ve formátu PDF
- Zdrojové soubory ve formátu LaTeX a ostatní materiály potřebné pro sazbu tohoto textu
- Spustitelná aplikace Blast3D (soubory *.jar a *.jad)
- Soubory obsahující zdrojový kód aplikace Blast3D (*.java)
- Aplikací využívané zdroje (textury *.bmp 8bpp, transparentní obrázky *.png, modely *.mbac, mapy levelů *.b3dm...)
- Zdroje pro aplikaci v původním formátu (textury *.bmp 24bpp, modely ve formátu eclipse a intermediate formátu *.bac)
- Nepoužité a ukázkové zdroje pro aplikaci